

filepath: promptflow/SECURITY.md content:

Security

Microsoft takes the security of our software products and services seriously, which includes all source code repositories managed through our GitHub organizations, which include [Microsoft](https://github.com/microsoft) (<https://github.com/microsoft>), [Azure](https://github.com/Azure) (<https://github.com/Azure>), [DotNet](https://github.com/dotnet) (<https://github.com/dotnet>), [AspNet](https://github.com/aspnet) (<https://github.com/aspnet>), [Xamarin](https://github.com/xamarin) (<https://github.com/xamarin>), and our GitHub organizations (<https://opensource.microsoft.com/>).

If you believe you have found a security vulnerability in any Microsoft-owned repository that meets [Microsoft's definition of a security vulnerability](https://aka.ms/opensource/security/definition) (<https://aka.ms/opensource/security/definition>), please report it to us as described below.

Reporting Security Issues

Please do not report security vulnerabilities through public GitHub issues.

Instead, please report them to the Microsoft Security Response Center (MSRC) at <https://msrc.microsoft.com/create-report> (<https://aka.ms/opensource/security/create-report>).

If you prefer to submit without logging in, send email to secure@microsoft.com (<mailto:secure@microsoft.com>). If possible, encrypt your message with our PGP key; please download it from the [Microsoft Security Response Center PGP Key page](https://aka.ms/opensource/security/pgpkey) (<https://aka.ms/opensource/security/pgpkey>).

You should receive a response within 24 hours. If for some reason you do not, please follow up via email to ensure we received your original message. Additional information can be found at [microsoft.com/msrc](https://aka.ms/opensource/security/msrc) (<https://aka.ms/opensource/security/msrc>).

Please include the requested information listed below (as much as you can provide) to help us better understand the nature and scope of the possible issue:

- Type of issue (e.g. buffer overflow, SQL injection, cross-site scripting, etc.)
- Full paths of source file(s) related to the manifestation of the issue
- The location of the affected source code (tag/branch/commit or direct URL)
- Any special configuration required to reproduce the issue
- Step-by-step instructions to reproduce the issue
- Proof-of-concept or exploit code (if possible)
- Impact of the issue, including how an attacker might exploit the issue

This information will help us triage your report more quickly.

If you are reporting for a bug bounty, more complete reports can contribute to a higher bounty award. Please visit our [Microsoft Bug Bounty Program](https://aka.ms/opensource/security/bounty) (<https://aka.ms/opensource/security/bounty>) page for more details about our active programs.

Preferred Languages

We prefer all communications to be in English.

Policy

Microsoft follows the principle of [Coordinated Vulnerability Disclosure](https://aka.ms/opensource/security/cvd) (<https://aka.ms/opensource/security/cvd>).

filepath: promptflow/CODE_OF_CONDUCT.md content: # Microsoft Open Source Code of Conduct

This project has adopted the [Microsoft Open Source Code of Conduct](https://opensource.microsoft.com/codeofconduct/) (<https://opensource.microsoft.com/codeofconduct/>).

Resources:

- [Microsoft Open Source Code of Conduct](https://opensource.microsoft.com/codeofconduct/) (<https://opensource.microsoft.com/codeofconduct/>)
- [Microsoft Code of Conduct FAQ](https://opensource.microsoft.com/codeofconduct/faq/) (<https://opensource.microsoft.com/codeofconduct/faq/>)
- Contact opencode@microsoft.com (<mailto:opencode@microsoft.com>) with questions or concerns

filepath: promptflow/CONTRIBUTING.md content: # Contributing to Prompt Flow You can contribute to prompt flow with issues and pull requests (PRs). Simply filing issues for problems you encounter is a great way to contribute. Contributing code is greatly appreciated.

Reporting Issues

We always welcome bug reports, API proposals and overall feedback. Here are a few tips on how you can make reporting your issue as effective as possible.

Where to Report

New issues can be reported in our [list of issues](https://github.com/microsoft/promptflow/issues) (<https://github.com/microsoft/promptflow/issues>).

Before filing a new issue, please search the list of issues to make sure it does not already exist.

If you do find an existing issue for what you wanted to report, please include your own feedback in the discussion. Do consider upvoting (👍 reaction) the original post, as this helps us prioritize popular issues in our backlog.

Writing a Good Bug Report

Good bug reports make it easier for maintainers to verify and root cause the underlying problem. The better a bug report, the faster the problem will be resolved. Ideally, a bug report should contain the following information:

- A high-level description of the problem.
- A *minimal reproduction*, i.e. the smallest size of code/configuration required to reproduce the wrong behavior.
- A description of the *expected behavior*, contrasted with the *actual behavior* observed.
- Information on the environment: OS/distribution, CPU architecture, SDK version, etc.
- Additional information, e.g. Is it a regression from previous versions? Are there any known workarounds?

Contributing Changes

Project maintainers will merge accepted code changes from contributors.

DOs and DON'Ts

DO's:

- **DO** follow the standard coding conventions: [Python \(https://pypi.org/project/black/\)](https://pypi.org/project/black/).
- **DO** give priority to the current style of the project or file you're changing if it diverges from the general guidelines.
- **DO** include tests when adding new features. When fixing bugs, start with adding a test that highlights how the current behavior is broken.
- **DO** add proper docstring for functions and classes following [API Documentation Guidelines \(/docs/dev/documentation_guidelines.md\)](#).
- **DO** keep the discussions focused. When a new or related topic comes up it's often better to create new issue than to side track the discussion.
- **DO** clearly state on an issue that you are going to take on implementing it.
- **DO** blog and tweet (or whatever) about your contributions, frequently!

DON'Ts:

- **DON'T** surprise us with big pull requests. Instead, file an issue and start a discussion so we can agree on a direction before you invest a large amount of time.
- **DON'T** commit code that you didn't write. If you find code that you think is a good fit to add to prompt flow, file an issue and start a discussion before proceeding.
- **DON'T** submit PRs that alter licensing related files or headers. If you believe there's a problem with them, file an issue and we'll be happy to discuss it.
- **DON'T** make new APIs without filing an issue and discussing with us first.

Breaking Changes

Contributions must maintain API signature and behavioral compatibility. Contributions that include breaking changes will be rejected. Please file an issue to discuss your idea or change if you believe that a breaking change is warranted.

Suggested Workflow

We use and recommend the following workflow:

1. Create an issue for your work, or reuse an existing issue on the same topic.
 - Get agreement from the team and the community that your proposed change is a good one.
 - Clearly state that you are going to take on implementing it, if that's the case. You can request that the issue be assigned to you. Note: The issue filer and the implementer don't have to be the same person.
2. Create a personal fork of the repository on GitHub (if you don't already have one).
3. In your fork, create a branch off of main (`git checkout -b my_branch`).
 - Name the branch so that it clearly communicates your intentions, such as "issue-123" or "githubhandle-issue".
4. Make and commit your changes to your branch.
5. Add new tests corresponding to your change, if applicable.
6. Run the relevant scripts in [the section below \(https://github.com/microsoft/promptflow/blob/main/CONTRIBUTING.md#dev-scripts\)](https://github.com/microsoft/promptflow/blob/main/CONTRIBUTING.md#dev-scripts) to ensure that your build is clean and all tests are passing.
7. Create a PR against the repository's **main** branch.
 - State in the description what issue or improvement your change is addressing.
 - Link the PR to the issue in step 1.
 - Verify that all the Continuous Integration checks are passing.
8. Wait for feedback or approval of your changes from the code maintainers.
 - If there is no response for a few days, you can create a new issue to raise awareness. Promptflow team has triage process toward issues without assignee, then you can directly contact the issue owner to follow up (e.g. loop related internal reviewer).
9. When area owners have signed off, and all checks are green, your PR will be merged.

Development scripts

The scripts below are used to build, test, and lint within the project.

- see [doc/dev/dev_setup.md](https://github.com/microsoft/promptflow/blob/main/docs/dev/dev_setup.md) (https://github.com/microsoft/promptflow/blob/main/docs/dev/dev_setup.md).

PR - CI Process

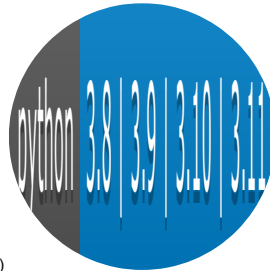
The continuous integration (CI) system will automatically perform the required builds and run tests (including the ones you are expected to run) for PRs. Builds and test runs must be clean.

If the CI build fails for any reason, the PR issue will be updated with a link that can be used to determine the cause of the failure.

filepath: promptflow/README.md content: # Prompt flow



<https://pypi.org/project/promptflow/>



<https://pypi.python.org/pypi/promptflow/>



<https://pypi.org/project/promptflow/>



[https://microsoft.github.io/promptflow/reference/pf-command-](https://microsoft.github.io/promptflow/reference/pf-command-reference.html)



[reference.html\)](https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow)

[https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow\)](https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow)



<https://microsoft.github.io/promptflow/index.html>



<https://github.com/microsoft/promptflow/issues/new/choose>

<https://github.com/microsoft/promptflow/issues/new/choose>



<https://github.com/microsoft/promptflow/blob/main/CONTRIBUTING.md>

<https://github.com/microsoft/promptflow/blob/main/LICENSE>



Welcome to join us to make prompt flow better by participating [discussions](https://github.com/microsoft/promptflow/discussions) (<https://github.com/microsoft/promptflow/discussions>), opening [issues](https://github.com/microsoft/promptflow/issues/new/choose) (<https://github.com/microsoft/promptflow/issues/new/choose>), submitting [PRs](https://github.com/microsoft/promptflow/pulls) (<https://github.com/microsoft/promptflow/pulls>).

Prompt flow is a suite of development tools designed to streamline the end-to-end development cycle of LLM-based AI applications, from ideation, prototyping, testing, evaluation to production deployment and monitoring. It makes prompt engineering much easier and enables you to build LLM apps with production quality.

With prompt flow, you will be able to:

- **Create and iteratively develop flow**
 - Create executable [flows](https://microsoft.github.io/promptflow/concepts/concept-flows.html) (<https://microsoft.github.io/promptflow/concepts/concept-flows.html>) that link LLMs, prompts, Python code and other [tools](https://microsoft.github.io/promptflow/concepts/concept-tools.html) (<https://microsoft.github.io/promptflow/concepts/concept-tools.html>) together.
 - Debug and iterate your flows, especially the [interaction with LLMs](https://microsoft.github.io/promptflow/concepts/concept-connections.html) (<https://microsoft.github.io/promptflow/concepts/concept-connections.html>) with ease.
- **Evaluate flow quality and performance**
 - Evaluate your flow's quality and performance with larger datasets.
 - Integrate the testing and evaluation into your CI/CD system to ensure quality of your flow.
- **Streamlined development cycle for production**
 - Deploy your flow to the serving platform you choose or integrate into your app's code base easily.
 - (Optional but highly recommended) Collaborate with your team by leveraging the cloud version of [Prompt flow in Azure AI](https://learn.microsoft.com/en-us/azure/machine-learning/prompt-flow/overview-what-is-prompt-flow?view=azureml-api-2) (<https://learn.microsoft.com/en-us/azure/machine-learning/prompt-flow/overview-what-is-prompt-flow?view=azureml-api-2>).

Installation

To get started quickly, you can use a pre-built development environment. **Click the button below** to open the repo in GitHub Codespaces, and then continue the readme!

Open in GitHub Codespaces

<https://codespaces.new/microsoft/promptflow?quickstart=1>

If you want to get started in your local environment, first install the packages:

Ensure you have a python environment, python=3.9 is recommended.

```
pip install promptflow promptflow-tools
```

Quick Start ↗

Create a chatbot with prompt flow

Run the command to initiate a prompt flow from a chat template, it creates folder named `my_chatbot` and generates required files within it:

```
pf flow init --flow ./my_chatbot --type chat
```

Setup a connection for your API key

For OpenAI key, establish a connection by running the command, using the `openai.yaml` file in the `my_chatbot` folder, which stores your OpenAI key (override keys and name with `--set` to avoid yaml file changes):

```
pf connection create --file ./my_chatbot/openai.yaml --set api_key=<your_api_key> --name open_ai_connection
```

For Azure OpenAI key, establish the connection by running the command, using the `azure_openai.yaml` file:

```
pf connection create --file ./my_chatbot/azure_openai.yaml --set api_key=<your_api_key> api_base=<your_api_base> --name open_ai_connection
```

Chat with your flow

In the `my_chatbot` folder, there's a `flow.dag.yaml` file that outlines the flow, including inputs/outputs, nodes, connection, and the LLM model, etc

Note that in the `chat` node, we're using a connection named `open_ai_connection` (specified in `connection` field) and the `gpt-35-turbo` model (specified in `deployment_name` field). The `deployment_name` field is to specify the OpenAI model, or the Azure OpenAI deployment resource.

Interact with your chatbot by running: (press `Ctrl + C` to end the session)

```
pf flow test --flow ./my_chatbot --interactive
```

Core value: ensuring "High Quality" from prototype to production

Explore our [15-minute tutorial \(examples/tutorials/flow-fine-tuning-evaluation/promptflow-quality-improvement.md\)](#) that guides you through prompt tuning ➡ batch testing ➡ evaluation, all designed to ensure high quality ready for production.

Next Step! Continue with the **Tutorial** ☐ section to delve deeper into prompt flow.

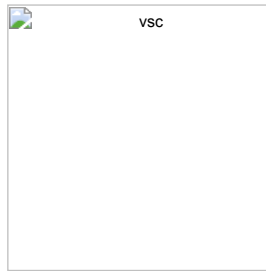
Tutorial ☐ ↗

Prompt flow is a tool designed to **build high quality LLM apps**, the development process in prompt flow follows these steps: develop a flow, improve the flow quality, deploy the flow to production.

Develop your own LLM apps

VS Code Extension

We also offer a VS Code extension (a flow designer) for an interactive flow development experience with UI.



You can install it from the [visualstudio marketplace \(https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow\)](https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow).

Deep delve into flow development

[Getting started with prompt flow \(.docs/cloud/azureai/quick-start/index.md\)](#): A step by step guidance to invoke your first flow run.

Learn from use cases

[Tutorial: Chat with PDF \(https://github.com/microsoft/promptflow/blob/main/examples/tutorials/e2e-development/chat-with-pdf.md\)](https://github.com/microsoft/promptflow/blob/main/examples/tutorials/e2e-development/chat-with-pdf.md): An end-to-end tutorial on how to build a high quality chat application with prompt flow, including flow development and evaluation with metrics.

More examples can be found [here \(https://microsoft.github.io/promptflow/tutorials/index.html#samples\)](https://microsoft.github.io/promptflow/tutorials/index.html#samples). We welcome contributions of new use cases!

Setup for contributors

If you're interested in contributing, please start with our dev setup guide: [dev_setup.md \(.docs/dev/dev_setup.md\)](#).

Next Step! Continue with the **Contributing** ☐ section to contribute to prompt flow.

Contributing

This project welcomes contributions and suggestions. Most contributions require you to agree to a Contributor License Agreement (CLA) declaring that you have the right to, and actually do, grant us the rights to use your contribution. For details, visit <https://cla.opensource.microsoft.com> (<https://cla.opensource.microsoft.com>).

When you submit a pull request, a CLA bot will automatically determine whether you need to provide a CLA and decorate the PR appropriately (e.g., status check, comment). Simply follow the instructions provided by the bot. You will only need to do this once across all repos using our CLA.

This project has adopted the [Microsoft Open Source Code of Conduct \(https://opensource.microsoft.com/codeofconduct/\)](https://opensource.microsoft.com/codeofconduct/). For more information see the [Code of Conduct FAQ \(https://opensource.microsoft.com/codeofconduct/fag/\)](https://opensource.microsoft.com/codeofconduct/fag/) or contact opencode@microsoft.com (<mailto:opencode@microsoft.com>), with any additional questions or comments.

Trademarks

This project may contain trademarks or logos for projects, products, or services. Authorized use of Microsoft trademarks or logos is subject to and must follow [Microsoft's Trademark & Brand Guidelines \(https://www.microsoft.com/en-us/legal/intellectualproperty/trademarks/usage/general\)](https://www.microsoft.com/en-us/legal/intellectualproperty/trademarks/usage/general). Use of Microsoft trademarks or logos in modified versions of this project must not cause confusion or imply Microsoft sponsorship. Any use of third-party trademarks or logos are subject to those third-party's policies.

Code of Conduct

This project has adopted the [Microsoft Open Source Code of Conduct \(https://opensource.microsoft.com/codeofconduct/\)](https://opensource.microsoft.com/codeofconduct/). For more information see the [Code of Conduct FAQ \(https://opensource.microsoft.com/codeofconduct/fag/\)](https://opensource.microsoft.com/codeofconduct/fag/) or contact opencode@microsoft.com (<mailto:opencode@microsoft.com>), with any additional questions or comments.

Data Collection

The software may collect information about you and your use of the software and send it to Microsoft if configured to enable telemetry. Microsoft may use this information to provide services and improve our products and services. You may turn on the telemetry as described in the repository. There are also some features in the software that may enable you and Microsoft to collect data from users of your applications. If you use these features, you must comply with applicable law, including providing appropriate notices to users of your applications together with a copy of Microsoft's privacy statement. Our privacy statement is located at <https://go.microsoft.com/fwlink/?LinkID=824704> (<https://go.microsoft.com/fwlink/?LinkID=824704>). You can learn more about data collection and use in the help documentation and our privacy statement. Your use of the software operates as your consent to these practices.

Telemetry Configuration

Telemetry collection is on by default.

To opt out, please run `pf config set telemetry.enabled=false` to turn it off.

License

Copyright (c) Microsoft Corporation. All rights reserved.

Licensed under the [MIT \(LICENSE\)](#) license.

filepath: promptflow/SUPPORT.md content: # Support

How to file issues and get help

This project uses GitHub Issues to track bugs and feature requests. Please search the existing issues before filing new issues to avoid duplicates. For new issues, file your bug or feature request as a new Issue.

Microsoft Support Policy

Support for this **PROJECT** or **PRODUCT** is limited to the resources listed above.

filepath: promptflow/.github/PULL_REQUEST_TEMPLATE.md content: # Description

Please add an informative description that covers that changes made by the pull request and link all relevant issues.

All Promptflow Contribution checklist:

- ☐ The pull request does not introduce [breaking changes](#).
- ☐ CHANGELOG is updated for new features, bug fixes or other significant changes.
- ☐ I have read the [contribution guidelines \(./CONTRIBUTING.md\)](#).
- ☐ Create an issue and link to the pull request to get dedicated review from promptflow team. Learn more: [suggested workflow \(./CONTRIBUTING.md#suggested-workflow\)](#).

General Guidelines and Best Practices

- ☐ Title of the pull request is clear and informative.
- ☐ There are a small number of commits, each of which have an informative message. This means that previously merged commits do not appear in the history of the PR. For more information on cleaning up the commits in your PR, [see this page \(https://github.com/Azure/azure-powershell/blob/master/documentation/development-docs/cleaning-up-commits.md\)](https://github.com/Azure/azure-powershell/blob/master/documentation/development-docs/cleaning-up-commits.md).

Testing Guidelines

- ☐ Pull request includes test coverage for the included changes.

filepath: promptflow/.github/ISSUE_TEMPLATE/vsc_bug_report.md content: --- name: Bug report for Prompt flow VS Code extension about: Bug report for Prompt flow VS Code extension title: "[BUG] [VSCode Extension]" labels: bug assignees: "

Describe the bug A clear and concise description of the bug.

How To Reproduce the bug Steps to reproduce the behavior, how frequent can you experience the bug: 1.

Screenshots

- On the VSCode primary side bar > the Prompt flow pane > quick access section. Find the "install dependencies" action. Please it and attach the screenshots there.
- Please provide other snapshots about the key steps to repro the issue.

Environment Information

- Promptflow Package Version using `pf -v`: [e.g. 0.0.102309906]
- Operating System: [e.g. Ubuntu 20.04, Windows 11]
- Python Version using `python --version`: [e.g. python==3.10.12]
- VS Code version.
- Prompt Flow extension version.
- On the VS Code bottom pane > Output pivot > "prompt flow" channel, find the error message could be relevant to the issue and past them here. That would be helpful for our trouble shooting.
- If your code to repro the issue is public and you want to share us, please share the link.

Additional context Add any other context about the problem here.

filepath: promptflow/.github/ISSUE_TEMPLATE/bug_report.md content: --- name: Bug report about: Create a report to help us improve title: "[BUG]" labels: bug assignees: "

Describe the bug A clear and concise description of the bug.

How To Reproduce the bug Steps to reproduce the behavior, how frequent can you experience the bug: 1.

Expected behavior A clear and concise description of what you expected to happen.

Screenshots If applicable, add screenshots to help explain your problem.

Running Information(please complete the following information):

- Promptflow Package Version using `pf -v`: [e.g. 0.0.102309906]
- Operating System: [e.g. Ubuntu 20.04, Windows 11]
- Python Version using `python --version`: [e.g. python==3.10.12]

Additional context Add any other context about the problem here.

filepath: promptflow/.github/ISSUE_TEMPLATE/contribution_request.md content: --- name: Contribution request about: Create a request of review to contribution title: "[Contribution Request]" labels: enhancement assignees: "

Is your contribution request related to a problem? Please describe. A clear and concise description of what you will contribute.

Detail the functionality and value. A clear and concise description of the contribution's functionality and value.

Additional context Add any other context or screenshots about the contribution request here.

filepath: promptflow/.github/ISSUE_TEMPLATE/feature_request.md content: --- name: Feature request about: Suggest an idea for this project title: "[Feature Request]" labels: enhancement assignees: "

Is your feature request related to a problem? Please describe. A clear and concise description of what the problem is.

Describe the solution you'd like A clear and concise description of what you want to happen.

Describe alternatives you've considered A clear and concise description of any alternative solutions or features you've considered.

Additional context Add any other context or screenshots about the feature request here.

filepath: promptflow/scripts/readme/README.md content: # Readme Workflow Generator

These tools is used to generate workflows from README.md and python notebook files in the [examples \(./examples/\)](#) folder.

- Generated workflows will be placed in `.github/workflows/samples` * (`./../github/workflows/`) folder.
- The script will also generate a new explanation [README.md \(./examples/README.md\)](#) for all the examples.

1. Install dependencies

```
pip install -r ../../examples/requirements.txt
pip install -r ../../examples/dev_requirements.txt
```

2. Generate workflows

(Option 1) One Step Generation

At the **root** of the repository, run the following command:

```
python scripts/readme/readme.py
```

(Option 2) Step by Step Generation

At the **root** of the repository, run the following command:


```
# Generate workflow from README.md inside examples folder
python scripts/readme/readme_generator.py -g "examples/**/*.ipynb"

# Generate workflow from python notebook inside examples folder
python scripts/readme/workflow_generator.py -g "examples/flows/**/*.README.md"
```

Multiple inputs are supported.

3. Options to control generations of examples README.md (../examples/README.md)

3.1 Notebook Workflow Generation

- Each workflow contains metadata area, set `.metadata.description` area will display this message in the corresponding cell in README.md (../examples/README.md) file.
- When set `.metadata.no_readme_generation` to value `true`, the script will stop generating for this notebook.

3.2 README.md Workflow Generation

- For README.md files, only `bash` cells will be collected and converted to workflow. No cells will produce no workflow.
- Readme descriptions are simply collected from the first sentence in the README.md file just below the title. The script will collect words before the first `.` of the first paragraph. Multi-line sentence is also supported

- A supported description sentence: This is a sample workflow for testing.
- A not supported description sentence: Please check www.microsoft.com for more details.

filepath: `promptflow/scripts/installer/windows/install_from_msi.md` content: # Install prompt flow MSI installer on Windows Prompt flow is a suite of development tools designed to streamline the end-to-end development cycle of LLM-based AI applications, that can be installed locally on Windows computers.

For Windows, the prompt flow is installed via an MSI, which gives you access to the CLI through the Windows Command Prompt (CMD) or PowerShell.

Install or update

The MSI distributable is used for installing or updating the prompt flow on Windows. You don't need to uninstall current versions before using the MSI installer because the MSI updates any existing version.

:sync: Microsoft Installer (MSI)

Latest version

Download and install the latest release of the prompt flow. When the installer asks if it can make changes to your computer, select the "Yes" box.

[Latest release of the promptflow \(64-bit\) \(https://aka.ms/installpromptflowwindowsx64\)](https://aka.ms/installpromptflowwindowsx64)

Specific version

If you prefer, you can download a specific version of the promptflow by using a URL. To download the MSI installer for a specific version, change the version segment in URL [https://promptflowartifact.blob.core.windows.net/msi-installer/promptflow-](https://promptflowartifact.blob.core.windows.net/msi-installer/promptflow-1.0.0-promptflow-windowsx64.msi) ([https://promptflowartifact.blob.core.windows.net/msi-installer/promptflow-](https://promptflowartifact.blob.core.windows.net/msi-installer/promptflow-1.0.0-promptflow-windowsx64.msi)) msi

:sync: Microsoft Installer (MSI) with PowerShell

PowerShell

To install the prompt flow using PowerShell, start PowerShell and run the following command:

```
$ProgressPreference = 'SilentlyContinue'; Invoke-WebRequest -Uri https://aka.ms/installpromptflowwindowsx64 -OutFile .\promptflow.msi
```

This will download and install the latest 64-bit installer of the prompt flow for Windows.

To install a specific version, replace the `-Uri` argument with the URL like below. Here is an example of using the 64-bit installer of the promptflow version 1.0.0 in PowerShell:

```
$ProgressPreference = 'SilentlyContinue'; Invoke-WebRequest -Uri https://promptflowartifact.blob.core.windows.net/msi-installer/promptflow-1.0.0-promptflow-windowsx64.msi -OutFile .\promptflow.msi
```

Run the prompt flow

You can now run the prompt flow with the `pf` or `pfazure` command from either Windows Command Prompt or PowerShell.

Upgrade the prompt flow

Beginning with version 1.4.0, the prompt flow provides an in-tool command to upgrade to the latest version.

```
pf upgrade
```

For prompt flow versions prior to 1.4.0, upgrade by reinstalling as described in [Install the prompt flow](#).

Uninstall

You uninstall the prompt flow from the Windows "Apps and Features" list. To uninstall:

Platform	Instructions
Windows 11	Start > Settings > Apps > Installed apps
Windows 10	Start > Settings > System > Apps & Features
Windows 8 and Windows 7	Start > Control Panel > Programs > Uninstall a program

Once on this screen type `_promptflow` into the program search bar. The program to uninstall is listed as **promptflow (64-bit)**. Select this application, then select the **Uninstall** button.

FAQ

Where is the prompt flow installed?

In Windows, the 64-bit prompt flow installs in `C:\Users**\AppData\Local\Apps\promptflow` by default.

What version of the prompt flow is installed?

Type `pf --version` in a terminal window to know what version of the prompt flow is installed. Your output looks like this:

```
promptflow          x.x.x

Executable '***\python.exe'
Python (Windows) 3.*.* | packaged by conda-forge | *
```

filepath: `promptflow/scripts/installer/windows/README.md` content: `# Building the Windows MSI Installer`

This document provides instructions on creating the MSI installer.

Option1: Building with Github Actions

Trigger the [workflow \(https://github.com/microsoft/promptflow/actions/workflows/build_msi_installer.yml\)](https://github.com/microsoft/promptflow/actions/workflows/build_msi_installer.yml) manually.

Option2: Local Building

Prerequisites

1. Turn on the '.NET Framework 3.5' Windows Feature (required for WIX Toolset).
2. Install 'Microsoft Build Tools 2015': <https://www.microsoft.com/download/details.aspx?id=48159> (<https://www.microsoft.com/download/details.aspx?id=48159>)
3. You need to have `curl.exe`, `unzip.exe` and `msbuild.exe` available under `PATH`.
4. Install 'WIX Toolset build tools' following the instructions below.
 - Enter the directory where the README is located (`cd scripts/installer/windows`), `mkdir wix` and `cd wix`.
 - `curl --output wix-archive.zip https://azurecliprod.blob.core.windows.net/msi/wix310-binaries-mirror.zip`
 - `unzip wix-archive.zip` and `del wix-archive.zip`
5. We recommend creating a clean virtual Python environment and installing all dependencies using `src/promptflow/setup.py`.
 - `python -m venv venv`
 - `venv\Scripts\activate`
 - `pip install promptflow[azure,executable] promptflow-tools`

Building

1. Update the version number \$(env.CLI_VERSION) and \$(env.FILE_VERSION) in product.wxs, promptflow.wixproj and version_info.txt.
2. cd scripts/installer/windows/scripts and run pyinstaller promptflow.spec.
3. cd scripts/installer/windows and Run msbuild /t:rebuild /p:Configuration=Release /p:Platform=x64 promptflow.wixproj.
4. The unsigned MSI will be in the scripts/installer/windows/out folder.

Notes

- If you encounter "Access is denied" error when running promptflow. Please follow the [link \(https://learn.microsoft.com/en-us/microsoft-365/security/defender-endpoint/attack-surface-reduction-rules-deployment-implement?view=o365-worldwide#customize-attack-surface-reduction-rules\)](https://learn.microsoft.com/en-us/microsoft-365/security/defender-endpoint/attack-surface-reduction-rules-deployment-implement?view=o365-worldwide#customize-attack-surface-reduction-rules) to add the executable to the Windows Defender Attack Surface Reduction (ASR) rule.

filepath: promptflow/scripts/installer/curl_install_pypi/README.md content: # Curl Install Script Information

The scripts in this directory are used for installing through curl and they point to the packages on PyPI.

Install or update promptflow

curl [https://promptflowartifact.blob.core.windows.net/linux-install-scripts/install_\(https://promptflowartifact.blob.core.windows.net/linux-install-scripts/install\)](https://promptflowartifact.blob.core.windows.net/linux-install-scripts/install_(https://promptflowartifact.blob.core.windows.net/linux-install-scripts/install)) | bash

The script can also be downloaded and run locally. You may have to restart your shell in order for the changes to take effect.

Uninstall promptflow

Uninstall the promptflow by directly deleting the files from the location chosen at the time of installation.

1. Remove the installed CLI files.

```
# The default install/executable location is the user's home directory ($HOME).
rm -r $HOME/lib/promptflow
rm $HOME/bin/pf
rm $HOME/bin/pfs
rm $HOME/bin/pfazure
```

2. Modify your \$HOME/.bash_profile or \$HOME/.bashrc file to remove the following line:

```
export PATH=$PATH:$HOME/bin
```

3. If using bash or zsh, reload your shell's command cache.

```
hash -r
```

filepath: promptflow/scripts/release/promptflow-release-note.md content: We are pleased to announce the release of promptflow {{VERSION}}.

This release includes some new features, bug fixes, and improvements. We recommend that all users upgrade to this version.

See the [CHANGELOG \(https://github.com/microsoft/promptflow/blob/release/promptflow%7B%7BVERSION%7D%7D/src/promptflow/CHANGELOG.md\)](https://github.com/microsoft/promptflow/blob/release/promptflow%7B%7BVERSION%7D%7D/src/promptflow/CHANGELOG.md) for a list of all the changes.

The release will be available via PyPI:

```
pip install --upgrade promptflow
```

Please report any issues with the release on the [promptflow issue tracker \(https://github.com/microsoft/promptflow/issues\)](https://github.com/microsoft/promptflow/issues).

Thanks to all the contributors who made this release possible.

filepath: promptflow/.devcontainer/README.md content: # Devcontainer for promptflow To facilitate your promptflow project development and empower you to work on LLM projects using promptflow more effectively, we've configured the necessary environment for developing promptflow projects and utilizing flows through the dev container feature. You can seamlessly initiate your promptflow project development and start leveraging flows by simply using the dev container feature via VS Code or Codespaces.

Use Github Codespaces

Use codespaces to open promptflow repo, it will automatically build the dev containers environment and open promptflow with dev containers. You can just click:



[.\(https://codespaces.new/microsoft/promptflow?quickstart=1\)](https://codespaces.new/microsoft/promptflow?quickstart=1)

Use local devcontainer



Use vscode to open promptflow repo, and install vscode extension: Dev Containers and then open promptflow with dev containers.

About dev containers please refer to: [dev containers \(https://code.visualstudio.com/docs/devcontainers/containers\)](https://code.visualstudio.com/docs/devcontainers/containers)

filepath: promptflow/examples/CONTRIBUTING.md content: # Contributing to examples folder

Thank you for your interest in contributing to the examples folder. This folder contains a collection of Python notebooks and selected markdown files that demonstrate various usage of this promptflow project. The script will automatically generate a README.md file in the root folder, listing all the notebooks and markdown files with their corresponding workflows.

Guidelines for notebooks and markdown files in examples folder

When creating or modifying a notebook or markdown file, please follow these guidelines:

- Each notebook or markdown file should have a clear and descriptive title as the first line
- Each notebook or markdown file should have a brief introduction that explains the purpose and scope of the example. For details, please refer to the readme workflow generator manual [README.md \(./scripts/readme/README.md\)](#) file.
 - The first sentence of first paragraph of the markdown file is important. The introduction should be concise and informative, and end with a period.
 - Each notebook file should have a metadata area when the file is opened as a big JSON file. The metadata area may contain the following fields:
 - `.metadata.description:` (Mandatory) A short description of the example that will be displayed in the README.md file. The description should be concise and informative, and end with a period.
 - `.metadata.stage:` (Optional) A value that indicates whether the script should skip generating a workflow for this notebook or markdown file. If set to `development`, the script will ignore this file. If set to other values or omitted, the script will generate a workflow for this file.
- Each notebook or markdown file should have a clear and logical structure, using appropriate headings, subheadings, comments, and code cells. The code cells should be executable and produce meaningful outputs.
- Each notebook or markdown file should follow the [PEP 8 \(https://peps.python.org/pep-0008/\)](https://peps.python.org/pep-0008/) style guide for Python code, and use consistent and readable variable names, indentation, spacing, and punctuation.
- Each notebook or markdown file should include relevant references, citations, and acknowledgements.
- If you are contributing to [tutorial \(./tutorials/\)](#), each notebook or markdown file should declare its dependent resources in its metadata, so that the auto generated workflow can listen to the changes of these resources to avoid unexpected breaking. Resources should be declared with relative path to the repo root, and here are examples for [notebook \(./tutorials/get-started/quickstart.ipynb\)](#) and [markdown \(./tutorials/e2e-development/chat-with-pdf.md\)](#).

Generate workflows, update README.md and submit pull requests

To run the `readme.py` script, you need to have Python 3 installed on your system. You also need to install the required packages by running:

```
# At this examples folder
pip install -r requirements.txt
pip install -r dev_requirements.txt
```

Then, you can run the script by:

```
# At the root of this repository
python scripts/readme/readme.py
```

For detailed usage of `readme.py`, please refer to the `readme` workflow generator manual [README.md](#) (`./scripts/readme/README.md`)

Update [README.md](#) (`./README.md`) in [examples](#) (`./`) folder

The `readme.py` script will scan all the notebooks and markdown files in the `examples` folder, and generate a `README.md` file in the root folder. The `README.md` file will contain a table of contents with links to each notebook and markdown file, as well as their descriptions and workflows.

Generations in the [workflows](#) (`./github/workflows/`) folder

This contains two parts:

- For notebooks, we'll prepare standard workflow running environment to test the notebook to the end.
- For markdowns, The workflows are generated by extracting the `bash` cells from markdown file. The workflows will prepare standard workflow running environment and test these cells to the end.

The script will also save workflows in the [workflows](#) (`./github/workflows/`) folder, where each notebook or markdown file will have a corresponding workflow file with the `.yml` extension. The workflow files can be triggered by creating a new pull request or pushing a new commit to the repository. The workflow will run the notebook or markdown file, and you could check the outputs afterwards.

Feedback and Support

If you have any feedback or need any support regarding this folder, submit an issue on GitHub. We appreciate your contribution and hope you enjoy using our project.

filepath: `promptflow/examples/README.md` content: `# Promptflow examples`



(<https://github.com/psf/black>)



([./LICENSE](#))

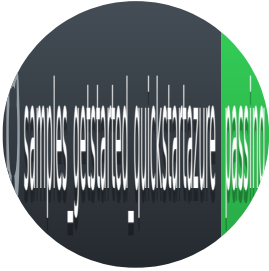
Get started

Install dependencies

- Bootstrap your python environment.
 - e.g: create a new [conda](https://conda.io/projects/conda/en/latest/user-guide/getting-started.html) (<https://conda.io/projects/conda/en/latest/user-guide/getting-started.html>) environment. `conda create -n pf-examples python=3.9`.
 - install required packages in python environment: `pip install -r requirements.txt`
 - show installed sdk: `pip show promptflow`

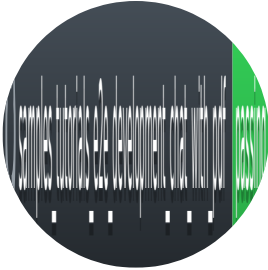


Quick start

path	status	descriptio
quickstart.ipynb (tutorials/get-started/quickstart.ipynb)	A circular logo with a dark grey background on the left and a green background on the right. The text "samples generated quickstart session" is in white on the grey part, and "session" is in white on the green part.	A quickstart tutorial to run a flow and evaluate it.
(https://github.com/microsoft/promptflow/actions/workflows/samples_getstarted_quickstart.yml)		

path		status	descriptio
quickstart-azure.ipynb (tutorials/get-started/quickstart-azure.ipynb)			A quickstart tutorial to run a flow in Azure AI and evaluate it.
https://github.com/microsoft/promptflow/actions/workflows/samples_getstarted_quickstartazure.yml			

CLI examples

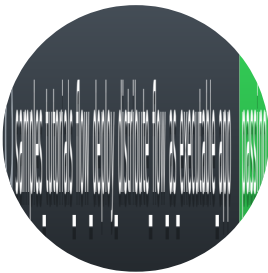
Tutorials ([tutorials](#) ([tutorials](#)))

path		status	
chat-with-pdf (tutorials/e2e-development/chat-with-pdf.md)			https://github.com/microsoft/promptflow/actions/workflows/samples_tutor
azure-app-service (tutorials/flow-deploy/azure-app-service/README.md)			https://github.com/microsoft/promptflow/actions/workflows/samples_tutor
create-service-with-flow (tutorials/flow-deploy/create-service-with-flow/README.md)			https://github.com/microsoft/promptflow/actions/workflows/samples_tutorials_flow_deploy_create_service_with_flow.yml

path

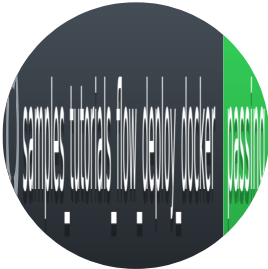
status

[distribute-flow-as-executable-app](#)
([tutorials/flow-deploy/distribute-flow-as-executable-app/README.md](#))



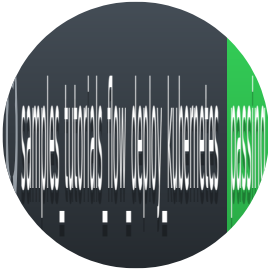
(https://github.com/microsoft/promptflow/actions/workflows/samples_tutorials_flow_deploy_distribute_f

[docker](#) ([tutorials/flow-deploy/docker/README.md](#))



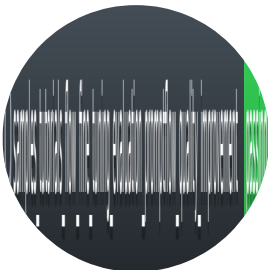
(https://github.com/microsoft/promptflow/actions/workflows/samples_tutor

[kubernetes](#) ([tutorials/flow-deploy/kubernetes/README.md](#))



(https://github.com/microsoft/promptflow/actions/workflows/samples_tutor

[promptflow-quality-improvement](#)
([tutorials/flow-fine-tuning-evaluation/promptflow-quality-improvement.md](#))



(https://github.com/microsoft/promptflow/actions/workflows/samples_tutorials_flow_fine_tuning_evalua

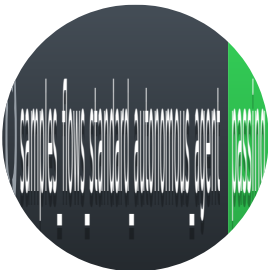
Flows ([flows](#)([flows](#)))

[Standard flows](#) ([flows/standard/](#))

path

status

[autonomous-agent](#)
([flows/standard/autonomous-agent/README.md](#))

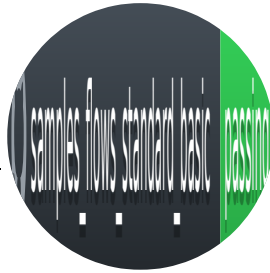


(https://github.com/microsoft/promptflow/actions/workflows/samples_flows_standard_autonomous_

path

status

basic
(flows/standard/basic/README.md)



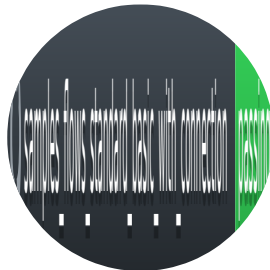
(https://github.com/microsoft/promptflow/actions/workflows/samples_flows_standard_basic.yml).

basic-with-builtin-llm
(flows/standard/basic-with-builtin-llm/README.md)



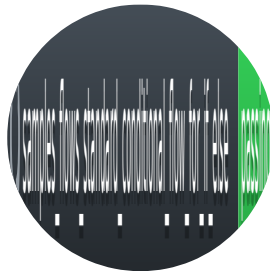
(https://github.com/microsoft/promptflow/actions/workflows/samples_flows_standard_basic_with_bu

basic-with-connection
(flows/standard/basic-with-connection/README.md)



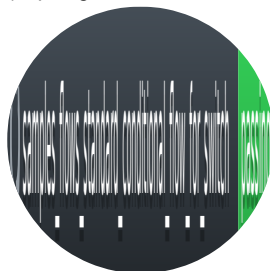
(https://github.com/microsoft/promptflow/actions/workflows/samples_flows_standard_basic_with_co

conditional-flow-for-if-else
(flows/standard/conditional-flow-for-if-else/README.md)



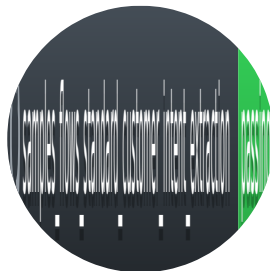
(https://github.com/microsoft/promptflow/actions/workflows/samples_flows_standard_conditional_flow

conditional-flow-for-switch
(flows/standard/conditional-flow-for-switch/README.md)



(https://github.com/microsoft/promptflow/actions/workflows/samples_flows_standard_conditional_flow

customer-intent-extraction
(flows/standard/customer-intent-extraction/README.md)

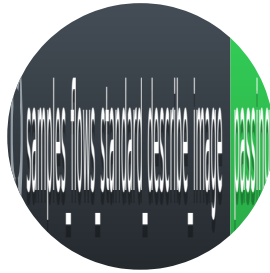


(https://github.com/microsoft/promptflow/actions/workflows/samples_flows_standard_customer_inte

path

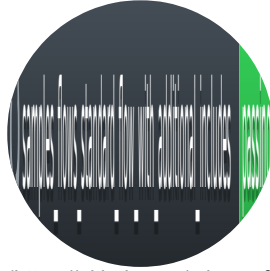
status

[describe-image](#)
([flows/standard/describe-image/README.md](#)).



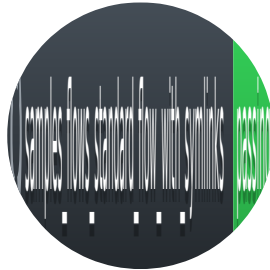
(https://github.com/microsoft/promptflow/actions/workflows/samples_flows_standard_describe_image

[flow-with-additional-includes](#)
([flows/standard/flow-with-additional-includes/README.md](#)).



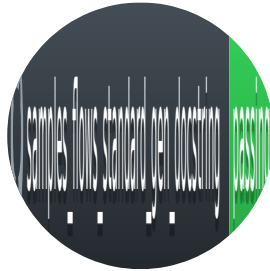
(https://github.com/microsoft/promptflow/actions/workflows/samples_flows_standard_flow_with_additional_includes

[flow-with-symlinks](#)
([flows/standard/flow-with-symlinks/README.md](#)).



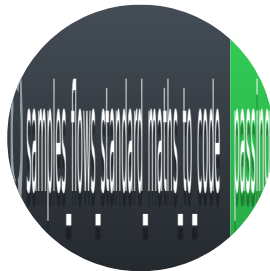
(https://github.com/microsoft/promptflow/actions/workflows/samples_flows_standard_flow_with_symlinks

[gen-docstring](#) ([flows/standard/gen-docstring/README.md](#)).



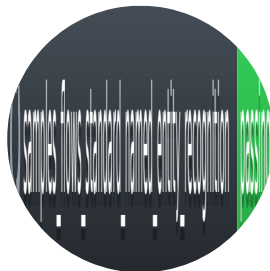
(https://github.com/microsoft/promptflow/actions/workflows/samples_flows_standard_gen_docstring

[maths-to-code](#)
([flows/standard/maths-to-code/README.md](#)).



(https://github.com/microsoft/promptflow/actions/workflows/samples_flows_standard_maths_to_code

[named-entity-recognition](#)
([flows/standard/named-entity-recognition/README.md](#)).

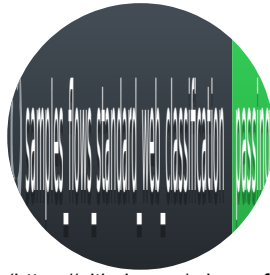


(https://github.com/microsoft/promptflow/actions/workflows/samples_flows_standard_named_entity_recognition

path

status

[web-classification](#)
([flows/standard/web-classification/README.md](#))



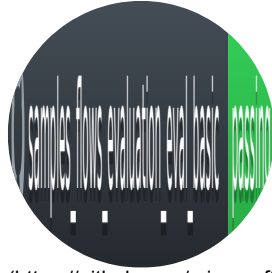
(https://github.com/microsoft/promptflow/actions/workflows/samples_flows_standard_web_classification.yml)

Evaluation flows (flows/evaluation/)

path

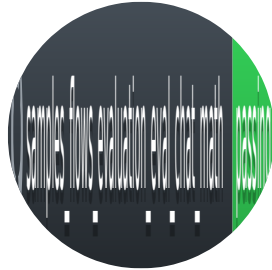
status

[eval-basic](#)
([flows/evaluation/eval-basic/README.md](#))



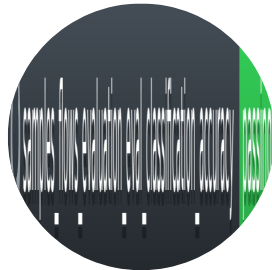
(https://github.com/microsoft/promptflow/actions/workflows/samples_flows_evaluation_eval_basic.yml)

[eval-chat-math](#)
([flows/evaluation/eval-chat-math/README.md](#))



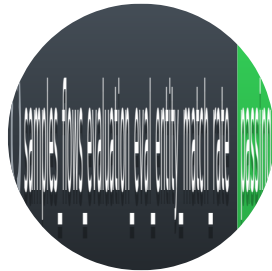
(https://github.com/microsoft/promptflow/actions/workflows/samples_flows_evaluation_eval_chat_math.yml)

[eval-classification-accuracy](#)
([flows/evaluation/eval-classification-accuracy/README.md](#))



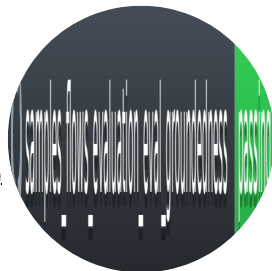
(https://github.com/microsoft/promptflow/actions/workflows/samples_flows_evaluation_eval_classification_accuracy.yml)

[eval-entity-match-rate](#)
([flows/evaluation/eval-entity-match-rate/README.md](#))

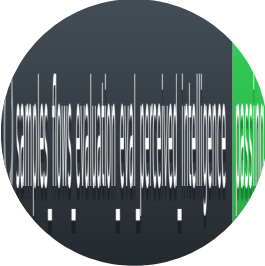
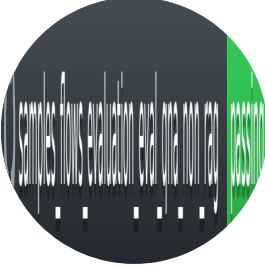
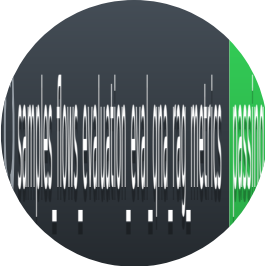


(https://github.com/microsoft/promptflow/actions/workflows/samples_flows_evaluation_eval_entity_match_rate.yml)

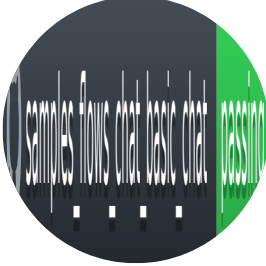
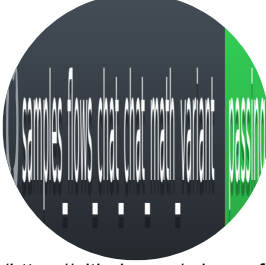
[eval-groundedness](#)
([flows/evaluation/eval-groundedness/README.md](#))



(https://github.com/microsoft/promptflow/actions/workflows/samples_flows_evaluation_eval_groundness.yml)

path		status
eval-perceived-intelligence (flows/evaluation/eval-perceived-intelligence/README.md)		https://github.com/microsoft/promptflow/actions/workflows/samples_flows_evaluation_eval_perceived_inte
eval-qna-non-rag (flows/evaluation/eval-qna-non-rag/README.md)		https://github.com/microsoft/promptflow/actions/workflows/samples_flows_evaluation_eval_qna_non_rag.
eval-qna-rag-metrics (flows/evaluation/eval-qna-rag-metrics/README.md)		https://github.com/microsoft/promptflow/actions/workflows/samples_flows_evaluation_eval_qna_rag_metr

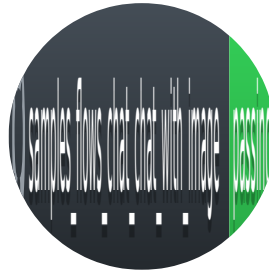
Chat flows (flows/chat/)

path		status
basic-chat (flows/chat/basic-chat/README.md)		https://github.com/microsoft/promptflow/actions/workflows/samples_flows_c
chat-math-variant (flows/chat/chat-math-variant/README.md)		https://github.com/microsoft/promptflow/actions/workflows/samples_flows_c

path

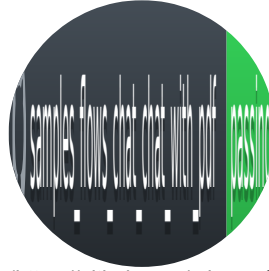
status

[chat-with-image \(flows/chat/chat-with-image/README.md\)](#)



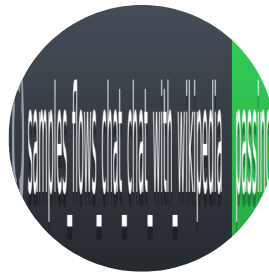
https://github.com/microsoft/promptflow/actions/workflows/samples_flows_c

[chat-with-pdf \(flows/chat/chat-with-pdf/README.md\)](#)



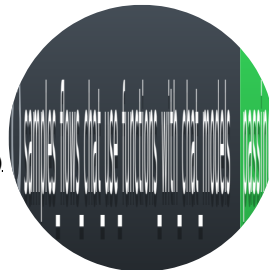
https://github.com/microsoft/promptflow/actions/workflows/samples_flows_c

[chat-with-wikipedia \(flows/chat/chat-with-wikipedia/README.md\)](#)



https://github.com/microsoft/promptflow/actions/workflows/samples_flows_c

[use_functions_with_chat_models \(flows/chat/use_functions_with_chat_models/README.md\)](#)



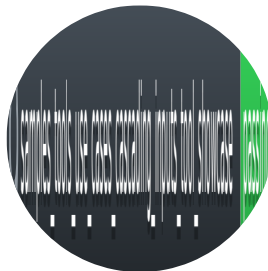
https://github.com/microsoft/promptflow/actions/workflows/samples_flows_c

Tool Use Cases ([Tool Use Cases \(tools/use-cases\)](#))

path

status

[cascading-inputs-tool-showcase \(tools/use-cases/cascading-inputs-tool-showcase/README.md\)](#)




https://github.com/microsoft/promptflow/actions/workflows/samples_tools_use_cases


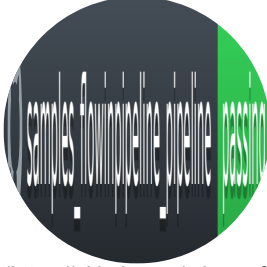
path	status	
custom-strong-type-connection-package-tool-showcase (tools/use-cases/custom-strong-type-connection-package-tool-showcase/README.md)		https://github.com/microsoft/promptflow/actions/workflows/samples_tools_use_cases_
custom-strong-type-connection-script-tool-showcase (tools/use-cases/custom-strong-type-connection-script-tool-showcase/README.md)		https://github.com/microsoft/promptflow/actions/workflows/samples_tools_use_cases_
custom_llm_tool_showcase (tools/use-cases/custom_llm_tool_showcase/README.md)		https://github.com/microsoft/promptflow/actions/workflows/samples_tools_use_cases_
dynamic-list-input-tool-showcase (tools/use-cases/dynamic-list-input-tool-showcase/README.md)		https://github.com/microsoft/promptflow/actions/workflows/samples_tools_use_cases_

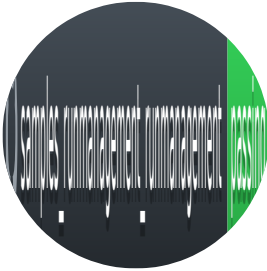
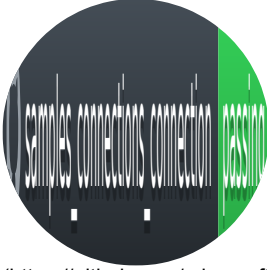
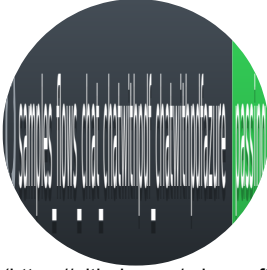
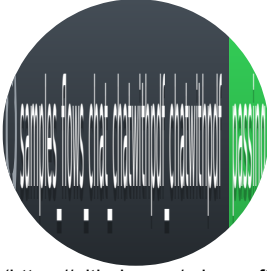
Connections (connections (connections))

path	status	description
------	--------	-------------

path	status	description
connections (connections/README.md)		This folder contains example <code>YAML</code> files for creating connection using <code>pf cli</code>
.https://github.com/microsoft/promptflow/actions/workflows/samples_connections.yml		

SDK examples

path	status
quickstart.ipynb (tutorials/get-started/quickstart.ipynb)	
.https://github.com/microsoft/promptflow/actions/workflows/samples_getstarted_quickstart.yml	
quickstart-azure.ipynb (tutorials/get-started/quickstart-azure.ipynb)	
.https://github.com/microsoft/promptflow/actions/workflows/samples_getstarted_quickstartazure.yml	
pipeline.ipynb (tutorials/flow-in-pipeline/pipeline.ipynb)	
.https://github.com/microsoft/promptflow/actions/workflows/samples_flowinpipeline_pipeline.yml	
flow-as-function.ipynb (tutorials/get-started/flow-as-function.ipynb)	
.https://github.com/microsoft/promptflow/actions/workflows/samples_getstarted_flowasfunction.yml	
cloud-run-management.ipynb (tutorials/run-management/cloud-run-management.ipynb)	
.https://github.com/microsoft/promptflow/actions/workflows/samples_runmanagement_cloudrunmanagen	

path	status
run-management.ipynb (tutorials/run-management/run-management.ipynb)	 (https://github.com/microsoft/promptflow/actions/workflows/samples_runmanagement_runmanagement.y)
connection.ipynb (connections/connection.ipynb)	 (https://github.com/microsoft/promptflow/actions/workflows/samples_connections_connection.yml)
chat-with-pdf-azure.ipynb (flows/chat/chat-with-pdf/chat-with-pdf-azure.ipynb)	 (https://github.com/microsoft/promptflow/actions/workflows/samples_flows_chat_chatwithpdf_chatwithpd)
chat-with-pdf.ipynb (flows/chat/chat-with-pdf/chat-with-pdf.ipynb)	 (https://github.com/microsoft/promptflow/actions/workflows/samples_flows_chat_chatwithpdf_chatwithpd)

Contributing

We welcome contributions and suggestions! Please see the [contributing guidelines](#) ([./CONTRIBUTING.md](#)) for details.

Code of Conduct

This project has adopted the [Microsoft Open Source Code of Conduct](#) (<https://opensource.microsoft.com/codeofconduct/>). Please see the [code of conduct](#) ([./CODE_OF_CONDUCT.md](#)) for details.

Reference

- [Promptflow documentation](#) (<https://microsoft.github.io/promptflow/>).

filepath: promptflow/examples/flows/chat/chat-with-image/README.md content: # Chat With Image

This flow demonstrates how to create a chatbot that can take image and text as input.

Tools used in this flow :

- OpenAI GPT-4V tool

Prerequisites

Install promptflow sdk and other dependencies in this folder:

```
pip install -r requirements.txt
```

What you will learn

In this flow, you will learn

- how to compose a chat flow with image and text as input. The chat input should be a list of text and/or images.

Getting started

1 Create connection for OpenAI GPT-4V tool to use

Go to "Prompt flow" "Connections" tab. Click on "Create" button, and create an "OpenAI" connection. If you do not have an OpenAI account, please refer to [OpenAI](https://platform.openai.com/) (<https://platform.openai.com/>) for more details.

```
# Override keys with --set to avoid yaml file changes
pf connection create --file ../../connections/azure_openai.yaml --set api_key=<your_api_key> api_base=<your_api_base> name=aoai_gpt4v_connection
```

Note in [flow.dag.yaml](#) ([flow.dag.yaml](#)) we are using connection named `aoai_gpt4v_connection`.

```
# show registered connection
pf connection show --name aoai_gpt4v_connection
```

2 Start chatting

```
# run chat flow with default question in flow.dag.yaml
pf flow test --flow .

# run chat flow with new question
pf flow test --flow . --inputs question=["How many colors can you see?", {"data:image/png;url": "https://developer.microsoft.com/_d"}]
```

```
# start a interactive chat session in CLI
pf flow test --flow . --interactive

# start a interactive chat session in CLI with verbose info
pf flow test --flow . --interactive --verbose
```

filepath: promptflow/examples/flows/chat/basic-chat/README.md content: # Basic Chat This example shows how to create a basic chat flow. It demonstrates how to create a chatbot that can remember previous interactions and use the conversation history to generate next message.

Tools used in this flow :

- llm tool

Prerequisites

Install promptflow sdk and other dependencies in this folder:

```
pip install -r requirements.txt
```

What you will learn

In this flow, you will learn

- how to compose a chat flow.
- prompt template format of LLM tool chat api. Message delimiter is a separate line containing role name and colon: "system:", "user:", "assistant:". See [OpenAI Chat](https://platform.openai.com/docs/api-reference/chat/create#chat/create-role) (<https://platform.openai.com/docs/api-reference/chat/create#chat/create-role>) for more about message role.


```
system:
You are a chatbot having a conversation with a human.

user:
{{question}}
```

- how to consume chat history in prompt.

```
{% for item in chat_history %}
user:
{{item.inputs.question}}
assistant:
{{item.outputs.answer}}
{% endfor %}
```

Getting started

1 Create connection for LLM tool to use

Go to "Prompt flow" "Connections" tab. Click on "Create" button, select one of LLM tool supported connection types and fill in the configurations.

Currently, there are two connection types supported by LLM tool: "AzureOpenAI" and "OpenAI". If you want to use "AzureOpenAI" connection type, you need to create an Azure OpenAI service first. Please refer to [Azure OpenAI Service \(https://azure.microsoft.com/en-us/products/cognitive-services/openai-service/\)](https://azure.microsoft.com/en-us/products/cognitive-services/openai-service/) for more details. If you want to use "OpenAI" connection type, you need to create an OpenAI account first. Please refer to [OpenAI \(https://platform.openai.com/\)](https://platform.openai.com/) for more details.

```
# Override keys with --set to avoid yaml file changes
pf connection create --file ../../connections/azure_openai.yaml --set api_key=<your_api_key> api_base=<your_api_base> --name open_ai_connection
```

Note in [flow.dag.yaml \(flow.dag.yaml\)](#) we are using connection named `open_ai_connection`.

```
# show registered connection
pf connection show --name open_ai_connection
```

2 Start chatting

```
# run chat flow with default question in flow.dag.yaml
pf flow test --flow .

# run chat flow with new question
pf flow test --flow . --inputs question="What's Azure Machine Learning?"

# start a interactive chat session in CLI
pf flow test --flow . --interactive

# start a interactive chat session in CLI with verbose info
pf flow test --flow . --interactive --verbose
```

filepath: `promptflow/examples/flows/chat/chat-with-wikipedia/README.md` content: # Chat With Wikipedia

This flow demonstrates how to create a chatbot that can remember previous interactions and use the conversation history to generate next message.

Tools used in this flow :

- llm tool
- custom python Tool

Prerequisites

Install promptflow sdk and other dependencies in this folder:

```
pip install -r requirements.txt
```

What you will learn

In this flow, you will learn

- how to compose a chat flow.
- prompt template format of LLM tool chat api. Message delimiter is a separate line containing role name and colon: "system:", "user:", "assistant:". See [OpenAI Chat \(https://platform.openai.com/docs/api-reference/chat/create#chat/create-role\)](https://platform.openai.com/docs/api-reference/chat/create#chat/create-role) for more about message role.

```
system:
You are a chatbot having a conversation with a human.

user:
{{question}}
```

- how to consume chat history in prompt.

```
{% for item in chat_history %}
user:
{{item.inputs.question}}
assistant:
{{item.outputs.answer}}
{% endfor %}
```

Getting started

1 Create connection for LLM tool to use

Go to "Prompt flow" "Connections" tab. Click on "Create" button, select one of LLM tool supported connection types and fill in the configurations.

Currently, there are two connection types supported by LLM tool: "AzureOpenAI" and "OpenAI". If you want to use "AzureOpenAI" connection type, you need to create an Azure OpenAI service first. Please refer to [Azure OpenAI Service \(https://azure.microsoft.com/en-us/products/cognitive-services/openai-service/\)](https://azure.microsoft.com/en-us/products/cognitive-services/openai-service/) for more details. If you want to use "OpenAI" connection type, you need to create an OpenAI account first. Please refer to [OpenAI \(https://platform.openai.com/\)](https://platform.openai.com/) for more details.

```
# Override keys with --set to avoid yaml file changes
pf connection create --file ../../connections/azure_openai.yml --set api_key=<your_api_key> api_base=<your_api_base>
```

Note in [flow.dag.yaml \(flow.dag.yaml\)](#), we are using connection named open_ai_connection.

```
# show registered connection
pf connection show --name open_ai_connection
```

2 Start chatting

```
# run chat flow with default question in flow.dag.yaml
pf flow test --flow .

# run chat flow with new question
pf flow test --flow . --inputs question="What's Azure Machine Learning?"

# start a interactive chat session in CLI
pf flow test --flow . --interactive

# start a interactive chat session in CLI with verbose info
pf flow test --flow . --interactive --verbose
```

filepath: promptflow/examples/flows/chat/chat-math-variant/README.md content: # Test your prompt variants for chat with math This is a prompt tuning case with 3 prompt variants for math question answering.

By utilizing this flow, in conjunction with the `evaluation/eval-chat-math` flow, you can quickly grasp the advantages of prompt tuning and experimentation with prompt flow. Here we provide a [video \(https://www.youtube.com/watch?v=gcle6nk2gA4\)](https://www.youtube.com/watch?v=gcle6nk2gA4) and a [tutorial \(\(./././tutorials/flow-fine-tuning-evaluation/promptflow-quality-improvement.md\)\)](#) for you to get started.

Tools used in this flow :

- `llm tool`
- `custom python Tool`

Prerequisites

Install promptflow sdk and other dependencies in this folder:

```
pip install -r requirements.txt
```

Getting started

1 Create connection for LLM tool to use

Go to "Prompt flow" "Connections" tab. Click on "Create" button, select one of LLM tool supported connection types and fill in the configurations.

Currently, there are two connection types supported by LLM tool: "AzureOpenAI" and "OpenAI". If you want to use "AzureOpenAI" connection type, you need to create an Azure OpenAI service first. Please refer to [Azure OpenAI Service \(https://azure.microsoft.com/en-us/products/cognitive-services/openai-service/\)](https://azure.microsoft.com/en-us/products/cognitive-services/openai-service/) for more details. If you want to use "OpenAI" connection type, you need to create an OpenAI account first. Please refer to [OpenAI \(https://platform.openai.com/\)](https://platform.openai.com/) for more details.

```
# Override keys with --set to avoid yaml file changes
pf connection create --file ../../connections/azure_openai.yaml --set api_key=<your_api_key> api_base=<your_api_base> --name open_ai_connection
```

Note in [flow.dag.yaml \(flow.dag.yaml\)](#) we are using connection named `open_ai_connection`.

```
# show registered connection
pf connection show --name open_ai_connection
```

2 Start chatting

```
# run chat flow with default question in flow.dag.yaml
pf flow test --flow .

# run chat flow with new question
pf flow test --flow . --inputs question="2+5=?"

# start a interactive chat session in CLI
pf flow test --flow . --interactive

# start a interactive chat session in CLI with verbose info
pf flow test --flow . --interactive --verbose

filepath: promptflow/examples/flows/chat/chat-with-pdf/README.md
content: # Chat with PDF

This is a simple flow that allow you to ask questions about the content of a PDF file and get answers.
You can run the flow with a URL to a PDF file and question as argument.
Once it's launched it will download the PDF and build an index of the content.
Then when you ask a question, it will look up the index to retrieve relevant content and post the question with the relevant content

Learn more on corresponding [tutorials](../../tutorials/e2e-development/chat-with-pdf.md).

Tools used in this flow:
- custom `python` Tool

## Prerequisites

Install promptflow sdk and other dependencies:
```bash
pip install -r requirements.txt
```

# Get started

## Create connection in this folder

```
create connection needed by flow
if pf connection list | grep open_ai_connection; then
 echo "open_ai_connection already exists"
else
 pf connection create --file ../../connections/azure_openai.yml --name open_ai_connection --set api_key=<your_api_key> api_base=
fi
```

## CLI Example

### Run flow

**Note:** this sample uses [predownloaded PDFs \(/chat\\_with\\_pdf/pdfs/\)](#) and [prebuilt FAISS Index \(/chat\\_with\\_pdf/index/\)](#) to speed up execution time. You can remove the folders to start a fresh run.

```
test with default input value in flow.dag.yaml
pf flow test --flow .

test with flow inputs
pf flow test --flow . --inputs question="What is the name of the new language representation model introduced in the document?" pdf_

(Optional) create a random run name
run_name="web_classification_"$(openssl rand -hex 12)

run with multiline data, --name is optional
pf run create --file batch_run.yaml --name $run_name

visualize run output details
pf run visualize --name $run_name
```

### Submit run to cloud

Assume we already have a connection named open\_ai\_connection in workspace.

```
set default workspace
az account set -s <your_subscription_id>
az configure --defaults group=<your_resource_group_name> workspace=<your_workspace_name>
```

```
create run
pfazure run create --file batch_run.yaml --name $run_name
```

Note: Click portal\_url of the run to view the final snapshot.

filepath: promptflow/examples/flows/chat/chat-with-pdf/chat\_with\_pdf/rewrite\_question\_prompt.md content: You are able to reason from previous conversation and the recent question, to come up with a rewrite of the question which is concise but with enough information that people without knowledge of previous conversation can understand the question.

A few examples:

# Example 1

## Previous conversation

user: Who is Bill Clinton? assistant: Bill Clinton is an American politician who served as the 42nd President of the United States from 1993 to 2001.

## Question

user: When was he born?

## Rewritten question

When was Bill Clinton born?

# Example 2

## Previous conversation

user: What is BERT? assistant: BERT stands for "Bidirectional Encoder Representations from Transformers." It is a natural language processing (NLP) model developed by Google.  
user: What data was used for its training? assistant: The BERT (Bidirectional Encoder Representations from Transformers) model was trained on a large corpus of publicly available text from the internet. It was trained on a combination of books, articles, websites, and other sources to learn the language patterns and relationships between words.

## Question

user: What NLP tasks can it perform well?

## Rewritten question

What NLP tasks can BERT perform well?

Now comes the actual work - please respond with the rewritten question in the same language as the question, nothing else.

## Previous conversation

{% for item in history %} {{item["role"]}}: {{item["content"]}} {% endfor %}

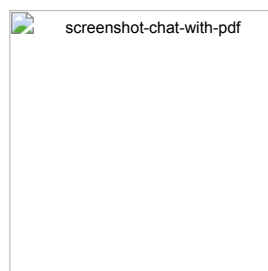
## Question

{{question}}

## Rewritten question

filepath: promptflow/examples/flows/chat/chat-with-pdf/chat\_with\_pdf/README.md content: # Chat with PDF This is a simple Python application that allow you to ask questions about the content of a PDF file and get answers. It's a console application that you start with a URL to a PDF file as argument. Once it's launched it will download the PDF and build an index of the content. Then when you ask a question, it will look up the index to retrieve relevant content and post the question with the relevant content to OpenAI chat model (gpt-3.5-turbo or gpt4) to get an answer.

## Screenshot - ask questions about BERT paper



## How it works?

## Get started

Create .env file in this folder with below content

```
OPENAI_API_BASE=<AOAI_endpoint>
OPENAI_API_KEY=<AOAI_key>
EMBEDDING_MODEL_DEPLOYMENT_NAME=text-embedding-ada-002
CHAT_MODEL_DEPLOYMENT_NAME=gpt-3.5-turbo
PROMPT_TOKEN_LIMIT=3000
MAX_COMPLETION_TOKENS=256
VERBOSE=false
CHUNK_SIZE=1024
CHUNK_OVERLAP=64
```

Note: CHAT\_MODEL\_DEPLOYMENT\_NAME should point to a chat model like gpt-3.5-turbo or gpt-4

## Run the command line

```
python main.py <url-to-pdf-file>
```

filepath: promptflow/examples/flows/chat/chat-with-pdf/chat\_with\_pdf/qna\_prompt.md content: You're a smart assistant can answer questions based on provided context and previous conversation history between you and human.

Use the context to answer the question at the end, note that the context has order and importance - e.g. context #1 is more important than #2.

Try as much as you can to answer based on the provided the context, if you cannot derive the answer from the context, you should say you don't know. Answer in the same language as the question.

# Context

{% for i, c in context %}

## Context #{{i+1}}

{{c.text}} {% endfor %}

# Question

{{question}}

filepath: promptflow/examples/flows/chat/use\_functions\_with\_chat\_models/README.md content: # Use Functions with Chat Models

This flow covers how to use the LLM tool chat API in combination with external functions to extend the capabilities of GPT models.

`functions` is an optional parameter in the [Chat Completion API \(https://platform.openai.com/docs/api-reference/chat/create\)](https://platform.openai.com/docs/api-reference/chat/create) which can be used to provide function specifications. The purpose of this is to enable models to generate function arguments which adhere to the provided specifications. Note that the API will not actually execute any function calls. It is up to developers to execute function calls using model outputs.

If the `functions` parameter is provided then by default the model will decide when it is appropriate to use one of the functions. The API can be forced to use a specific function by setting the `function_call` parameter to `{"name": "<insert-function-name>"}`. The API can also be forced to not use any function by setting the `function_call` parameter to `"none"`. If a function is used, the output will contain `"finish_reason": "function_call"` in the response, as well as a `function_call` object that has the name of the function and the generated function arguments. You can refer to [openai sample \(https://github.com/openai/openai-cookbook/blob/main/examples/How\\_to\\_call\\_functions\\_with\\_chat\\_models.ipynb\)](https://github.com/openai/openai-cookbook/blob/main/examples/How_to_call_functions_with_chat_models.ipynb) for more details.

## What you will learn

In this flow, you will learn how to use functions with LLM chat models and how to compose function role message in prompt template.

## Tools used in this flow

- LLM tool
- Python tool

## Prerequisites

Install prompt-flow sdk and other dependencies:

```
pip install -r requirements.txt
```

# Getting started

## 1 Create connection for LLM tool to use

Go to "Prompt flow" "Connections" tab. Click on "Create" button, select one of LLM tool supported connection types and fill in the configurations.

Currently, there are two connection types supported by LLM tool: "AzureOpenAI" and "OpenAI". If you want to use "AzureOpenAI" connection type, you need to create an Azure OpenAI service first. Please refer to [Azure OpenAI Service \(https://azure.microsoft.com/en-us/products/cognitive-services/openai-service/\)](https://azure.microsoft.com/en-us/products/cognitive-services/openai-service/) for more details. If you want to use "OpenAI" connection type, you need to create an OpenAI account first. Please refer to [OpenAI \(https://platform.openai.com/\)](https://platform.openai.com/) for more details.

```
Override keys with --set to avoid yaml file changes
pf connection create --file ../../connections/azure_openai.yaml --set api_key=<your_api_key> api_base=<your_api_base> --name open_ai_connection
```

Note in [flow.dag.yaml \(flow.dag.yaml\)](#), we are using connection named open\_ai\_connection.

```
show registered connection
pf connection show --name open_ai_connection
```

## 2 Start chatting

```
run chat flow with default question in flow.dag.yaml
pf flow test --flow .

run chat flow with new question
pf flow test --flow . --inputs question="How about London next week?"

start a interactive chat session in CLI
pf flow test --flow . --interactive

start a interactive chat session in CLI with verbose info
pf flow test --flow . --interactive --verbose
```

# References

- [OpenAI cookbook example \(https://github.com/openai/openai-cookbook/blob/main/examples/How\\_to\\_call\\_functions\\_with\\_chat\\_models.ipynb\)](https://github.com/openai/openai-cookbook/blob/main/examples/How_to_call_functions_with_chat_models.ipynb)
- [OpenAI function calling announcement \(https://openai.com/blog/function-calling-and-other-api-updates?ref=upstract.com\)](https://openai.com/blog/function-calling-and-other-api-updates?ref=upstract.com)
- [OpenAI function calling doc \(https://platform.openai.com/docs/guides/gpt/function-calling\)](https://platform.openai.com/docs/guides/gpt/function-calling)
- [OpenAI function calling API \(https://platform.openai.com/docs/api-reference/chat/create\)](https://platform.openai.com/docs/api-reference/chat/create)

filepath: promptflow/examples/flows/integrations/README.md content: # Integrations Folder

This folder contains flow examples contributed by various contributors. Each flow example should have a README.md file that provides a comprehensive introduction to the flow and includes contact information for the flow owner.

# Guideline for README.md of flows

To ensure consistency and clarity, please follow the guidelines below when creating the README.md file for your flow example. You can also refer to the [README.md \(../standard/web-classification/README.md\)](#) file in the [web-classification \(../standard/web-classification\)](#) flow example as a reference.

Note: Above sample README.md may not have contact information because it's a shared example and people can open issues to this repository if they have any questions about the flow example. For integration samples, **please make sure to include contact information in your README.md file**.

## Introduction (Required)

Provide a detailed description of the flow, including its components, inputs, outputs, and any dependencies. Explain how the flow works and what problem it solves. This section should give users a clear understanding of the flow's functionality and how it can be used.

# Tools Used in this Flow (Required)

List all the tools (functions) used in the flow. This can include both standard tools provided by prompt flow and any custom tools created specifically for the flow. Include a brief description of each tool and its purpose within the flow.

# Pre-requisites (Required)

List any pre-requisites that are required to run the flow. This can include any specific versions of prompt flow or other dependencies. If there are any specific configurations or settings that need to be applied, make sure to mention them in this section.

# Getting Started (Required)

Provide step-by-step instructions on how to get started with the flow. This should include any necessary setup or configuration steps, such as installing dependencies or setting up connections. If there are specific requirements or prerequisites, make sure to mention them in this section.

# Usage Examples

Include usage examples that demonstrate how to run the flow and provide input data. This can include command-line instructions or code snippets. Show users how to execute the flow and explain the expected output or results.

# Troubleshooting

If there are any known issues or troubleshooting tips related to the flow, include them in this section. Provide solutions or workarounds for common problems that users may encounter. This will help users troubleshoot issues on their own and reduce the need for support.

# Contribution Guidelines

If you would like to encourage other users to contribute to your flow or provide guidelines for contributing to the integration folder, include a section with contribution guidelines. This can include instructions on how to submit pull requests, guidelines for code formatting, or any other relevant information.

# Contact (Required)

Specify the flow owner and provide contact information in the README.md file. This can include an email address, GitHub username, or any other preferred method of contact. By including this information, users will be able to reach out to the owner with any questions or issues related to the flow.

# Conclusion

By following these guidelines, you can create a well-structured and informative README.md file for your flow example. This will help users understand and utilize your flow effectively. If you have any further questions or need assistance, please don't hesitate to reach out. Happy contributing!

filepath: promptflow/examples/flows/integrations/azure-ai-language/multi\_intent\_conversational\_language\_understanding/README.md content: # Multi Intent Conversational Language Understanding

A flow that can be used to determine multiple intents in a user query leveraging an LLM with Conversational Language Understanding.

This sample flow utilizes Azure AI Language's Conversational Language Understanding to perform various analyses on text or documents. It performs:

- Breakdown of compound multi intent user queries into single user queries using an LLM.
- [Conversational Language Understanding \(https://learn.microsoft.com/en-us/azure/ai-services/language-service/conversational-language-understanding/overview\)](https://learn.microsoft.com/en-us/azure/ai-services/language-service/conversational-language-understanding/overview) on each of those single user queries.

See the [promptflow-azure-ai-language \(https://github.com/microsoft/promptflow/blob/main/docs/integrations/tools/azure\\_ai\\_language\\_tool.md\)](https://github.com/microsoft/promptflow/blob/main/docs/integrations/tools/azure_ai_language_tool.md) tool package reference documentation for further information.

Tools used in this flow:

- `LLM tool`
- `conversational_language_understanding tool from the promptflow-azure-ai-language package`

Connections used in this flow:

- `Custom connection`



# Prerequisites

Install promptflow sdk and other dependencies:

```
pip install -r requirements.txt
```

## Setup connection

Prepare your [Azure AI Language Resource \(https://azure.microsoft.com/en-us/products/ai-services/ai-language\)](https://azure.microsoft.com/en-us/products/ai-services/ai-language), first, and [create a Language Resource \(https://portal.azure.com/#create/Microsoft.CognitiveServicesTextAnalytics\)](https://portal.azure.com/#create/Microsoft.CognitiveServicesTextAnalytics), if necessary. Import the accompanying MediaPlayer.json into a CLU app, train the app and deploy. From your Language Resource, obtain its `api_key` and `endpoint`.

Create a connection to your Language Resource. The connection uses the `CustomConnection` schema:

```
Override keys with --set to avoid yaml file changes
pf connection create -f ../connections/azure_ai_language.yml --set secrets.api_key=<your_api_key> configs.endpoint=<your_endpoint> n
```

Ensure you have created the `azure_ai_language_connection`:

```
pf connection show -n azure_ai_language_connection
```

## Run flow

```
Test with default input values in flow.dag.yaml:
pf flow test --flow .
```

## Flow description

The flow uses a `llm` node to break down compound user queries into simple user queries. For example, "Play some blues rock and turn up the volume" will be broken down to "[Play some blues rock", "Turn Up the volume"]". This is then passed into the CLU tool to recognize intents and entities in each of the utterances.

## Contact

Please reach out to Abhishek Sen ([absen@microsoft.com](mailto:absen@microsoft.com) (<mailto:absen@microsoft.com>)) or [taincidents@microsoft.com](mailto:taincidents@microsoft.com) (<mailto:taincidents@microsoft.com>) with any issues.

filepath: `promptflow/examples/flows/integrations/azure-ai-language/analyze_documents/README.md` content: `# Analyze Documents`

A flow that analyzes documents with various language-based Machine Learning models.

This sample flow utilizes Azure AI Language's pre-built and optimized language models to perform various analyses on text or documents. It performs:

- [Translation \(https://learn.microsoft.com/en-us/rest/api/cognitiveservices/translator/translator/translate?view=rest-cognitiveservices-translator-v3.0&tabs=HTTP\)](https://learn.microsoft.com/en-us/rest/api/cognitiveservices/translator/translator/translate?view=rest-cognitiveservices-translator-v3.0&tabs=HTTP)
- [Personally Identifiable Information \(PII\) detection \(https://learn.microsoft.com/en-us/azure/ai-services/language-service/personally-identifiable-information/overview\)](https://learn.microsoft.com/en-us/azure/ai-services/language-service/personally-identifiable-information/overview)
- [Named Entity Recognition \(NER\) \(https://learn.microsoft.com/en-us/azure/ai-services/language-service/named-entity-recognition/overview\)](https://learn.microsoft.com/en-us/azure/ai-services/language-service/named-entity-recognition/overview)
- [Document Summarization \(https://learn.microsoft.com/en-us/azure/ai-services/language-service/summarization/overview?tabs=document-summarization\)](https://learn.microsoft.com/en-us/azure/ai-services/language-service/summarization/overview?tabs=document-summarization)
- [Sentiment Analysis & Opinion Mining \(https://learn.microsoft.com/en-us/azure/ai-services/language-service/sentiment-opinion-mining/overview?tabs=prebuilt\)](https://learn.microsoft.com/en-us/azure/ai-services/language-service/sentiment-opinion-mining/overview?tabs=prebuilt)

See the [promptflow-azure-ai-language \(https://github.com/microsoft/promptflow/blob/main/docs/integrations/tools/azure\\_ai\\_language\\_tool.md\)](https://github.com/microsoft/promptflow/blob/main/docs/integrations/tools/azure_ai_language_tool.md) tool package reference documentation for further information.

Tools used in this flow:

- `python` tool
- `translator` tool from the `promptflow-azure-ai-language` package
- `pii_entity_recognition` tool from the `promptflow-azure-ai-language` package
- `abstractive_summarization` tool from the `promptflow-azure-ai-language` package
- `extractive_summarization` tool from the `promptflow-azure-ai-language` package
- `entity_recognition` tool from the `promptflow-azure-ai-language` package
- `sentiment_analysis` tool from the `promptflow-azure-ai-language` package

Connections used in this flow:

- `Custom` connection (Azure AI Language)
- `Custom` connection (Azure AI Translator)

# Prerequisites

Install promptflow sdk and other dependencies:

```
pip install -r requirements.txt
```

## Setup connection

Prepare your [Azure AI Language Resource \(https://azure.microsoft.com/en-us/products/ai-services/ai-language\)](https://azure.microsoft.com/en-us/products/ai-services/ai-language) first, and [create a Language Resource \(https://portal.azure.com/#create/Microsoft.CognitiveServicesTextAnalytics\)](https://portal.azure.com/#create/Microsoft.CognitiveServicesTextAnalytics) if necessary. From your Language Resource, obtain its `api_key` and `endpoint`.

Create a connection to your Language Resource. The connection uses the `CustomConnection` schema:

```
Override keys with --set to avoid yaml file changes
pf connection create -f ../connections/azure_ai_language.yml --set secrets.api_key=<your_api_key> configs.endpoint=<your_endpoint> n
```

Ensure you have created the `azure_ai_language_connection`:

```
pf connection show -n azure_ai_language_connection
```

To use the `translator` tool, you must have an existing [Azure AI Translator resource \(https://azure.microsoft.com/en-us/products/ai-services/ai-translator\)](https://azure.microsoft.com/en-us/products/ai-services/ai-translator). [Create a Translator resource \(https://learn.microsoft.com/en-us/azure/ai-services/translator/create-translator-resource\)](https://learn.microsoft.com/en-us/azure/ai-services/translator/create-translator-resource) first, if necessary. From your Translator Resource, obtain its `api_key`, `endpoint`, and `region` (if applicable).

Create a connection to your Translator Resource. The connection uses the `CustomConnection` schema:

```
Override keys with --set to avoid yaml file changes
pf connection create -f ../connections/azure_ai_translator.yml --set secrets.api_key=<your_api_key> configs.endpoint=<your_endpoint>
```

Ensure you have created the `azure_ai_translator_connection`:

```
pf connection show -n azure_ai_translator_connection
```

## Run flow

### Run with single line input

```
Test with default input values in flow.dag.yaml:
pf flow test --flow .
Test with specific input:
pf flow test --flow . --inputs document_path=<path_to_txt_file> language=<document_language_code>
```

### Run with multiple lines of data

```
pf run create --flow . --data ./data.jsonl --column-mapping document_path='${data.document_path}' language='${data.language}' --stre
```

You can also skip providing column-mapping if provided data has same column name as the flow. Reference [here \(https://microsoft.github.io/promptflow/how-to-guides/run-and-evaluate-a-flow/use-column-mapping.html\)](https://microsoft.github.io/promptflow/how-to-guides/run-and-evaluate-a-flow/use-column-mapping.html) for default behavior when column-mapping not provided in CLI.

## Flow description

The flow first uses a `python` node to read in the provided `.txt` file into a string. This string is passed to a `pii_entity_recognition` node where Personally Identifiable Information (PII) is redacted. The redacted text is passed to `abstractive_summarization`, `extractive_summarization` and `entity_recognition` nodes, where summaries and named-entities are obtained. Finally, the generated abstractive summary is forwarded to a `sentiment_analysis` node to analyze its general sentiment.

## Contact

Please reach out to Sean Murray ([murraysean@microsoft.com](mailto:murraysean@microsoft.com) (<mailto:murraysean@microsoft.com>)) or [taincidents@microsoft.com](mailto:taincidents@microsoft.com) (<mailto:taincidents@microsoft.com>) with any issues.

filepath: promptflow/examples/flows/standard/flow-with-additional-includes/README.md content: # Flow with additional\_includes

User sometimes need to reference some common files or folders, this sample demos how to solve the problem using additional\_includes. The file or folders in additional includes will be copied to the snapshot folder by promptflow when operate this flow.

## Tools used in this flow

- LLM Tool
- Python Tool

## What you will learn

In this flow, you will learn

- how to add additional includes to the flow

## Prerequisites

Install promptflow sdk and other dependencies:

```
pip install -r requirements.txt
```

## Getting Started

### 1. Add additional includes to flow

You can add this field additional\_includes into the `flow.dag.yaml` (`flow.dag.yaml`). The value of this field is a list of the relative file/folder path to the flow folder.

```
additional_includes:
- ../web-classification/classify_with_llm.jinja2
- ../web-classification/convert_to_dict.py
- ../web-classification/fetch_text_content_from_url.py
- ../web-classification/prepare_examples.py
- ../web-classification/summarize_text_content.jinja2
- ../web-classification/summarize_text_content__variant_1.jinja2
```

### 2. Test & run the flow with additional includes

In this sample, this flow will references some files in the `web-classification` (`../web-classification/README.md`) flow. You can execute this flow with additional\_include or submit it to cloud. The snapshot generated by Promptflow contains additional include files/folders.

#### Test flow with single line data

```
test with default input value in flow.dag.yaml
pf flow test --flow .
test with user specified inputs
pf flow test --flow . --inputs url='https://www.microsoft.com/en-us/d/xbox-wireless-controller-stellar-shift-special-edition/94fbjc7l'
```

#### Run with multi-line data

```
create run using command line args
pf run create --flow . --data ./data.jsonl --column-mapping url='${data.url}' --stream
create run using yaml file
pf run create --file run.yml --stream
```

You can also skip providing column-mapping if provided data has same column name as the flow. Reference [here \(https://aka.ms/pf/column-mapping\)](https://aka.ms/pf/column-mapping) for default behavior when column-mapping not provided in CLI.

#### Submit run to cloud

Assume we already have a connection named `open_ai_connection` in workspace.

```
set default workspace
az account set -s <your_subscription_id>
az configure --defaults group=<your_resource_group_name> workspace=<your_workspace_name>
```

```
create run
pfazure run create --flow . --data ./data.jsonl --column-mapping url='${data.url}' --stream
pfazure run create --file run.yml
```

Note: Click `portal_url` of the run to view the final snapshot.

filepath: `promptflow/examples/flows/standard/gen-docstring/README.md` content: # Generate Python docstring This example can help you automatically generate Python code's docstring and return the modified code.

Tools used in this flow :

- `load_code` tool, it can load code from a file path.
  - Load content from a local file.
  - Loading content from a remote URL, currently loading HTML content, not just code.
- `divide_code` tool, it can divide code into code blocks.
  - To avoid files that are too long and exceed the token limit, it is necessary to split the file.
  - Avoid using the same function (such as `init(self)`) to generate docstrings in the same one file, which may cause confusion when adding docstrings to the corresponding functions in the future.
- `generate_docstring` tool, it can generate docstring for a code block, and merge docstring into origin code.

## What you will learn

In this flow, you will learn

- How to compose an auto generate docstring flow.
- How to use different LLM APIs to request LLM, including synchronous/asynchronous APIs, chat/completion APIs.
- How to use asynchronous multiple coroutine approach to request LLM API.
- How to construct a prompt.

## Prerequisites

Install promptflow sdk and other dependencies:

```
pip install -r requirements.txt
```

### Create connection for LLM to use

```
Override keys with --set to avoid yaml file changes
pf connection create --file ../../connections/azure_openai.yml --set api_key=<your_api_key> api_base=<your_api_base>
```

Note:

The `azure_openai.yml` (`../../connections/azure_openai.yml`) file is located in connections folder.  
We are using connection named `open_ai_connection` in `flow.dag.yaml` (`flow.dag.yaml`).

## Execute with Promptflow

### Execute with SDK

```
python main.py --source <your_file_path>
```

**Note:** the file path should be a python file path, default is `./azure_open_ai.py`.



A webpage will be generated, displaying diff:

## Execute with CLI

```
run flow with default file path in flow.dag.yaml
pf flow test --flow .

run flow with file path
pf flow test --flow . --inputs source="./azure_open_ai.py"
```

```
run flow with batch data
pf run create --flow . --data ./data.jsonl --name auto_generate_docstring --column-mapping source='${data.source}'
```

Output the code after add the docstring.

You can also skip providing `column-mapping` if provided data has same column name as the flow. Reference [here \(https://aka.ms/pf/column-mapping\)](https://aka.ms/pf/column-mapping) for default behavior when `column-mapping` not provided in CLI.

filepath: promptflow/examples/flows/standard/web-classification/README.md content: # Web Classification

This is a flow demonstrating multi-class classification with LLM. Given an url, it will classify the url into one web category with just a few shots, simple summarization and classification prompts.

## Tools used in this flow

- LLM Tool
- Python Tool

## What you will learn

In this flow, you will learn

- how to compose a classification flow with LLM.
- how to feed few shots to LLM classifier.

## Prerequisites

Install promptflow sdk and other dependencies:

```
pip install -r requirements.txt
```

## Getting Started

### 1. Setup connection

If you are using Azure Open AI, prepare your resource follow this [instruction \(https://learn.microsoft.com/en-us/azure/cognitive-services/openai/how-to/create-resource?pivots=web-portal\)](https://learn.microsoft.com/en-us/azure/cognitive-services/openai/how-to/create-resource?pivots=web-portal) and get your `api_key` if you don't have one.

```
Override keys with --set to avoid yaml file changes
pf connection create --file ../../connections/azure_openai.yaml --set api_key=<your_api_key> api_base=<your_api_base> --name open_
```

If you using OpenAI, sign up account [OpenAI website \(https://openai.com/\)](https://openai.com/), login and [find personal API key \(https://platform.openai.com/account/api-keys\)](https://platform.openai.com/account/api-keys).

```
pf connection create --file ../../connections/openai.yml --set api_key=<your_api_key>
```

## 2. Configure the flow with your connection

flow.dag.yaml is already configured with connection named open\_ai\_connection.

## 3. Test flow with single line data

```
test with default input value in flow.dag.yaml
pf flow test --flow .
test with user specified inputs
pf flow test --flow . --inputs url='https://www.youtube.com/watch?v=kYqRtjDBci8'
```

## 4. Run with multi-line data

```
create run using command line args
pf run create --flow . --data ./data.jsonl --column-mapping url='${data.url}' --stream

(Optional) create a random run name
run_name="web_classification_"$(openssl rand -hex 12)
create run using yaml file, run_name will be used in following contents, --name is optional
pf run create --file run.yml --stream --name $run_name
```

You can also skip providing column-mapping if provided data has same column name as the flow. Reference [here \(https://aka.ms/pf/column-mapping\)](https://aka.ms/pf/column-mapping) for default behavior when column-mapping not provided in CLI.

```
list run
pf run list
show run
pf run show --name $run_name
show run outputs
pf run show-details --name $run_name
```

## 5. Run with classification evaluation flow

create evaluation run:

```
(Optional) save previous run name into variable, and create a new random run name for further use
prev_run_name=$run_name
run_name="classification_accuracy_"$(openssl rand -hex 12)
create run using command line args
pf run create --flow ../../evaluation/eval-classification-accuracy --data ./data.jsonl --column-mapping groundtruth='${data.answer}'
create run using yaml file, --name is optional
pf run create --file run_evaluation.yml --run $prev_run_name --stream --name $run_name
```

```
pf run show-details --name $run_name
pf run show-metrics --name $run_name
pf run visualize --name $run_name
```

## 6. Submit run to cloud

```

set default workspace
az account set -s <your_subscription_id>
az configure --defaults group=<your_resource_group_name> workspace=<your_workspace_name>

create run
pfazure run create --flow . --data ./data.jsonl --column-mapping url='${data.url}' --stream

(Optional) create a new random run name for further use
run_name="web_classification_$(openssl rand -hex 12)"

create run using yaml file, --name is optional
pfazure run create --file run.yaml --name $run_name

pfazure run stream --name $run_name
pfazure run show-details --name $run_name
pfazure run show-metrics --name $run_name

(Optional) save previous run name into variable, and create a new random run name for further use
prev_run_name=$run_name
run_name="classification_accuracy_$(openssl rand -hex 12)"

create evaluation run, --name is optional
pfazure run create --flow ../../evaluation/eval-classification-accuracy --data ./data.jsonl --column-mapping groundtruth='${data.answer}' --stream
pfazure run create --file run_evaluation.yaml --run $prev_run_name --stream --name $run_name

pfazure run stream --name $run_name
pfazure run show --name $run_name
pfazure run show-details --name $run_name
pfazure run show-metrics --name $run_name
pfazure run visualize --name $run_name

```

filepath: `promptflow/examples/flows/standard/basic-with-builtin-llm/README.md` content: # Basic flow with builtin llm tool A basic standard flow that calls Azure OpenAI with builtin llm tool.

Tools used in this flow :

- `prompt tool`
- `builtin llm tool`

Connections used in this flow:

- `azure_open_ai connection`

## Prerequisites

Install promptflow sdk and other dependencies:

```
pip install -r requirements.txt
```

## Setup connection

Prepare your Azure Open AI resource follow this [instruction \(https://learn.microsoft.com/en-us/azure/cognitive-services/openai/how-to/create-resource?pivots=web-portal\)](https://learn.microsoft.com/en-us/azure/cognitive-services/openai/how-to/create-resource?pivots=web-portal) and get your `api_key` if you don't have one.

Note in this example, we are using [chat api \(https://learn.microsoft.com/en-us/azure/ai-services/openai/how-to/chatgpt?pivots=programming-language-chat-completions\)](https://learn.microsoft.com/en-us/azure/ai-services/openai/how-to/chatgpt?pivots=programming-language-chat-completions), please use `gpt-35-turbo` or `gpt-4` model deployment.

Ensure you have created `open_ai_connection` connection before.

```
pf connection show -n open_ai_connection
```

Create connection if you haven't done that. Ensure you have put your azure open ai endpoint key in `azure_openai.yml (./../connections/azure_openai.yml)` file.

```
Override keys with --set to avoid yaml file changes
pf connection create -f ../../../../connections/azure_openai.yml --name open_ai_connection --set api_key=<your_api_key> api_base=<your_api_base>
```

## Run flow

### Run with single line input

```
test with default input value in flow.dag.yaml
pf flow test --flow .

test with inputs
pf flow test --flow . --inputs text="Python Hello World!"
```

### run with multiple lines data

- create run

```
pf run create --flow . --data ./data.jsonl --column-mapping text='${data.text}' --stream
```

You can also skip providing `column-mapping` if provided data has same column name as the flow. Reference [here \(https://aka.ms/pf/column-mapping\)](https://aka.ms/pf/column-mapping) for default behavior when `column-mapping` not provided in CLI.

- list and show run meta

```
list created run
pf run list

get a sample run name
name=$(pf run list -r 10 | jq '[] | select(.name | contains("basic_with_builtin_llm")) | .name' | head -n 1 | tr -d '"')

show specific run detail
pf run show --name $name

show output
pf run show-details --name $name

visualize run in browser
pf run visualize --name $name
```

filepath: `promptflow/examples/flows/standard/basic-with-connection/README.md` content: # Basic flow with custom connection A basic standard flow that using custom python tool calls Azure OpenAI with connection info stored in custom connection.

Tools used in this flow :

- `prompt tool`
- `custom python Tool`

Connections used in this flow:

- None

## Prerequisites

Install promptflow sdk and other dependencies:

```
pip install -r requirements.txt
```

## Setup connection

Prepare your Azure Open AI resource follow this [instruction \(https://learn.microsoft.com/en-us/azure/cognitive-services/openai/how-to/create-resource?pivots=web-portal\)](https://learn.microsoft.com/en-us/azure/cognitive-services/openai/how-to/create-resource?pivots=web-portal) and get your `api_key` if you don't have one.



Create connection if you haven't done that.

```
Override keys with --set to avoid yaml file changes
pf connection create -f custom.yml --set secrets.api_key=<your_api_key> configs.api_base=<your_api_base>
```

Ensure you have created `basic_custom_connection` connection.

```
pf connection show -n basic_custom_connection
```

## Run flow

### Run with single line input

```
test with default input value in flow.dag.yaml
pf flow test --flow .

test with flow inputs
pf flow test --flow . --inputs text="Hello World!"

test node with inputs
pf flow test --flow . --node llm --inputs prompt="Write a simple Hello World! program that displays the greeting message."
```

### Run with multiple lines data

- create run

```
pf run create --flow . --data ./data.jsonl --column-mapping text='${data.text}' --stream
```

You can also skip providing `column-mapping` if provided data has same column name as the flow. Reference [here \(https://aka.ms/pf/column-mapping\)](https://aka.ms/pf/column-mapping) for default behavior when `column-mapping` not provided in CLI.

- list and show run meta

```
list created run
pf run list -r 3

get a sample run name
name=$(pf run list -r 10 | jq '[] | select(.name | contains("basic_with_connection")) | .name' | head -n 1 | tr -d ' ')

show specific run detail
pf run show --name $name

show output
pf run show-details --name $name

visualize run in browser
pf run visualize --name $name
```

### Run with connection override

Ensure you have created `open_ai_connection` connection before.

```
pf connection show -n open_ai_connection
```

Create connection if you haven't done that.

```
Override keys with --set to avoid yaml file changes
pf connection create --file ../../connections/azure_openai.yml --set api_key=<your_api_key> api_base=<your_api_base>
```

Run flow with newly created connection.

```
pf run create --flow . --data ./data.jsonl --connections llm.connection=open_ai_connection --column-mapping text='${data.text}' --st
```

## Run in cloud with connection override

Ensure you have created `open_ai_connection` connection in cloud. Reference [this notebook \(./../tutorials/get-started/quickstart-azure.ipynb\)](#) on how to create connections in cloud with UI.

Run flow with connection `open_ai_connection`.

```
set default workspace
az account set -s <your_subscription_id>
az configure --defaults group=<your_resource_group_name> workspace=<your_workspace_name>

pfazure run create --flow . --data ./data.jsonl --connections llm.connection=open_ai_connection --column-mapping text='${data.text}'
```

filepath: promptflow/examples/flows/standard/autonomous-agent/README.md content: # Autonomous Agent

This is a flow showcasing how to construct a AutoGPT agent with promptflow to autonomously figures out how to apply the given functions to solve the goal, which is film trivia that provides accurate and up-to-date information about movies, directors, actors, and more in this sample.

It involves inferring next executed function and user intent with LLM, and then use the function to generate observation. The observation above will be used as augmented prompt which is the input of next LLM inferring loop until the inferred function is the signal that you have finished all your objectives. The functions set used in the flow contains Wikipedia search function that can search the web to find answer about current events and PythonREPL python function that can run python code in a REPL.

For the sample input about movie introduction, the AutoGPT usually runs 4 rounds to finish the task. The first round is searching for the movie name, the second round is searching for the movie director, the third round is calculating director age, and the last round is outputting finishing signal. It takes 30s~40s to finish the task, but may take longer time if you use "gpt-3.5" or encounter Azure OpenAI rate limit. You could use "gpt-4" or go to <https://aka.ms/oai/quotaincrease> (<https://aka.ms/oai/quotaincrease>) if you would like to further increase the default rate limit.

Note: This is just a sample introducing how to use promptflow to build a simple AutoGPT. You can go to <https://github.com/Significant-Gravitas/Auto-GPT> (<https://github.com/Significant-Gravitas/Auto-GPT>) to get more concepts about AutoGPT.

## What you will learn

In this flow, you will learn

- how to use prompt tool.
- how to compose an AutoGPT flow using functions.

## Prerequisites

Install prompt-flow sdk and other dependencies:

```
pip install -r requirements.txt
```

## Getting Started

### 1 Create Azure OpenAI or OpenAI connection

```
Override keys with --set to avoid yaml file changes
pf connection create --file ../../connections/azure_openai.yaml --set api_key=<your_api_key> api_base=<your_api_base>
```

Note that you need to use "2023-07-01-preview" as Azure OpenAI connection API version when using function calling. See [How to use function calling with Azure OpenAI Service](#) (<https://learn.microsoft.com/en-us/azure/ai-services/openai/how-to/function-calling>) for more details.

### 2. Configure the flow with your connection

`flow.dag.yaml` is already configured with connection named `open_ai_connection`. It is recommended to use "gpt-4" model for stable performance. Using "gpt-3.5-turbo" may lead to the model getting stuck in the agent inner loop due to its suboptimal and unstable performance.

### 3. Test flow with single line data

```
test with default input value in flow.dag.yaml
pf flow test --flow .
```

## 4. Run with multi-line data

```
create run using command line args
pf run create --flow . --data ./data.jsonl --column-mapping name='${data.name}' role='${data.role}' goals='${data.goals}' --stream
```

You can also skip providing `column-mapping` if provided data has same column name as the flow. Reference [here \(https://aka.ms/pf/column-mapping\)](https://aka.ms/pf/column-mapping) for default behavior when `column-mapping` not provided in CLI.

## Disclaimer

LLM systems are susceptible to prompt injection, and you can gain a deeper understanding of this issue in the [technical blog \(https://developer.nvidia.com/blog/securing-llm-systems-against-prompt-injection/\)](https://developer.nvidia.com/blog/securing-llm-systems-against-prompt-injection/). As an illustration, the PythonREPL function might execute harmful code if provided with a malicious prompt within the provided sample. Furthermore, we cannot guarantee that implementing AST validations solely within the PythonREPL function will reliably elevate the sample's security to an enterprise level. We kindly remind you to refrain from utilizing this in a production environment.

filepath: promptflow/examples/flows/standard/flow-with-symlinks/README.md content: # Flow with symlinks

User sometimes need to reference some common files or folders, this sample demos how to solve the problem using symlinks. But it has the following limitations. It is recommended to use **additional include**. Learn more: [flow-with-additional-includes \(./flow-with-additional-includes/README.md\)](#).

1. For Windows user, by default need Administrator role to create symlinks.
2. For Windows user, directly copy the folder with symlinks, it will deep copy the contents to the location.
3. Need to update the git config to support symlinks.

### Notes:

- For Windows user, please grant user permission to [create symbolic links without administrator role \(https://learn.microsoft.com/en-us/windows/security/threat-protection/security-policy-settings/create-symbolic-links\)](https://learn.microsoft.com/en-us/windows/security/threat-protection/security-policy-settings/create-symbolic-links).
  1. Open your Local Security Policy
  2. Find Local Policies -> User Rights Assignment -> Create symbolic links
  3. Add you user name to this policy then reboot the compute.

### Attention:

- For git operations, need to set: `git config core.symlinks true`

## Tools used in this flow

- LLM Tool
- Python Tool

## What you will learn

In this flow, you will learn

- how to use symlinks in the flow

## Prerequisites

Install promptflow sdk and other dependencies:

```
pip install -r requirements.txt
```

## Getting Started

### 1. Create symbolic links in the flow

```
python ./create_symlinks.py
```

## 2. Test & run the flow with symlinks

In this sample, this flow will references some files in the [web-classification](#) ([./web-classification/README.md](#)) flow, and assume you already have required connection setup. You can execute this flow or submit it to cloud.

### Test flow with single line data

```
test flow with default input value in flow.dag.yaml
pf flow test --flow .

test flow with input
pf flow test --flow . --inputs url=https://www.youtube.com/watch?v=o5ZQyXaAv1g answer=Channel evidence=Url

test node in the flow
pf flow test --flow . --node convert_to_dict --inputs classify_with_llm.output='{ "category": "App", "evidence": "URL" }'
```

### Run with multi-line data

```
create run using command line args
pf run create --flow . --data ./data.jsonl --column-mapping url='${data.url}' --stream

create run using yaml file
pf run create --file run.yml --stream
```

You can also skip providing `column-mapping` if provided data has same column name as the flow. Reference [here](https://aka.ms/pf/column-mapping) (<https://aka.ms/pf/column-mapping>) for default behavior when `column-mapping` not provided in CLI.

### Submit run to cloud

```
create run
pfazure run create --flow . --data ./data.jsonl --column-mapping url='${data.url}' --stream --subscription <your_subscription_id> -g

set default workspace
az account set -s <your_subscription_id>
az configure --defaults group=<your_resource_group_name> workspace=<your_workspace_name>

pfazure run create --file run.yml --stream
```

filepath: promptflow/examples/flows/standard/maths-to-code/README.md content: # Math to Code Math to Code is a project that utilizes the power of the chatGPT model to generate code that models math questions and then executes the generated code to obtain the final numerical answer.

#### [!NOTE]

Building a system that generates executable code from user input with LLM is [a complex problem with potential security risks](https://developer.nvidia.com/blog/securing-llm-systems-against-prompt-injection/) (<https://developer.nvidia.com/blog/securing-llm-systems-against-prompt-injection/>), this example is more of a demonstration rather than something you can directly use in production. To build such system correctly, you should address key security considerations like input validation, additional sanitization of the code generated or better run the generated code in a sandbox environment.

Tools used in this flow :

- python tool
- built-in llm tool

Connections used in this flow:

- open\_ai connection

## Prerequisites

Install promptflow sdk and other dependencies:

```
pip install -r requirements.txt
```

## Setup connection

Prepare your Azure Open AI resource follow this [instruction \(https://learn.microsoft.com/en-us/azure/cognitive-services/openai/how-to/create-resource?pivots=web-portal\)](https://learn.microsoft.com/en-us/azure/cognitive-services/openai/how-to/create-resource?pivots=web-portal) and get your api\_key if you don't have one.

Note in this example, we are using [chat api \(https://learn.microsoft.com/en-us/azure/ai-services/openai/how-to/chatgpt?pivots=programming-language-chat-completions\)](https://learn.microsoft.com/en-us/azure/ai-services/openai/how-to/chatgpt?pivots=programming-language-chat-completions), please use gpt-35-turbo or gpt-4 model deployment.

Create connection if you haven't done that. Ensure you have put your azure open ai endpoint key in [azure\\_openai.yml \(azure\\_openai.yml\)](#) file.

```
Override keys with --set to avoid yaml file changes
pf connection create -f ../../../../connections/azure_openai.yml --set api_key=<your_api_key> api_base=<your_api_base>
```

Ensure you have created open\_ai\_connection connection.

```
pf connection show -n open_ai_connection
```

## Run flow in local

### Run locally with single line input

```
test with default input value in flow.dag.yaml
pf flow test --flow .
test with specific input
pf flow test --flow . --inputs math_question='If a rectangle has a length of 10 and width of 5, what is the area?'
```

### Run with multiple lines data

- create run

```
create a random run name
run_name="math_to_code_"$(openssl rand -hex 12)
pf run create --flow . --data ./math_data.jsonl --column-mapping math_question='${data.question}' --name $run_name --stream
```

## Get the accuracy using evaluation flow

Use [eval-accuracy-maths-to-code \(./evaluation/eval-accuracy-maths-to-code/\)](#) to evaluate accuracy and error rate metrics against the math-to-code flow.

- **accuracy**: if the generated code can be correctly executed and got final number answer, it will be compare with the groundtruth in the test data. For single instance, it's True if the final number equals to the groundtruth, False otherwise. Accuracy is to measure the correct percentage against test data.
- **error\_rate**: some case the flow cannot get number answer, for example, the generated code cannot be executed due to code parsing error of dependent package not available in conda env. Error rate is to measure the percentage of this case in test data.

```
create a random eval run name
eval_run_name="math_to_code_eval_run_"$(openssl rand -hex 12)

invoke accuracy and error rate evaluation against math-to-code batch run
pf run create --flow ../../evaluation/eval-accuracy-maths-to-code/ --data ./math_data.jsonl --column-mapping groundtruth='${data.answer}' --name $eval_run_name --stream

view the run details
pf run show-details -n $eval_run_name
pf run show-metrics -n $eval_run_name
```

filepath: [promptflow/examples/flows/standard/customer-intent-extraction/README.md](#) content: # Customer Intent Extraction

This sample is using OpenAI chat model(ChatGPT/GPT4) to identify customer intent from customer's question.

By going through this sample you will learn how to create a flow from existing working code (written in LangChain in this case).

This is the [existing code \(./intent.py\)](#).

## Prerequisites

Install promptflow sdk and other dependencies:

```
pip install -r requirements.txt
```

Ensure you have put your azure open ai endpoint key in .env file.

```
cat .env
```

## Run flow

### 1. init flow directory - create promptflow folder from existing python file

```
pf flow init --flow . --entry intent.py --function extract_intent --prompt-template chat_prompt=user_intent_zero_shot.jinja2
```

The generated files:

- `extract_intent_tool.py`: Wrap the func `extract_intent` in the `intent.py` script into a [Python Tool \(https://promptflow.azurewebsites.net/tools-reference/python-tool.html\)](https://promptflow.azurewebsites.net/tools-reference/python-tool.html).
- `flow.dag.yaml`: Describes the DAG(Directed Acyclic Graph) of this flow.
- `.gitignore`: File/folder in the flow to be ignored.

### 2. create needed custom connection

```
pf connection create -f .env --name custom_connection
```

### 3. test flow with single line input

```
pf flow test --flow . --input ./data/denormalized-flat.jsonl
```

### 4. run with multiple lines input

```
pf run create --flow . --data ./data --column-mapping history='${data.history}' customer_info='${data.customer_info}'
```

You can also skip providing `column-mapping` if provided data has same column name as the flow. Reference [here \(https://aka.ms/pf/column-mapping\)](https://aka.ms/pf/column-mapping) for default behavior when `column-mapping` not provided in CLI.

### 5. list/show

```
list created run
pf run list

get a sample completed run name
name=$(pf run list | jq '.[0] | select(.name | contains("customer_intent_extraction")) | .name' | head -n 1 | tr -d ' ')

show run
pf run show --name $name

show specific run detail, top 3 lines
pf run show-details --name $name -r 3
```

### 6. evaluation

```
create evaluation run
pf run create --flow ../../evaluation/eval-classification-accuracy --data ./data --column-mapping groundtruth='${data.intent}' prediction='${data.output}'
```

```
get the evaluation run in previous step
eval_run_name=$(pf run list | jq '.[0] | select(.name | contains("eval_classification_accuracy")) | .name' | head -n 1 | tr -d ' ')

show run
pf run show --name $eval_run_name

show run output
pf run show-details --name $eval_run_name -r 3
```

### 6. visualize

```
visualize in browser
pf run visualize --name $eval_run_name # your evaluation run name
```

# Deploy

## Serve as a local test app

```
pf flow serve --source . --port 5123 --host localhost
```

Visit <http://localhost:5123> to access the test app.

## Export

### Export as docker

```
pf flow export --source . --format docker --output ./package
```

filepath: promptflow/examples/flows/standard/named-entity-recognition/README.md content: # Named Entity Recognition

A flow that perform named entity recognition task.

Named Entity Recognition (NER) ([https://en.wikipedia.org/wiki/Named-entity\\_recognition](https://en.wikipedia.org/wiki/Named-entity_recognition)) is a Natural Language Processing (NLP) task. It involves identifying and classifying named entities (such as people, organizations, locations, date expressions, percentages, etc.) in a given text. This is a crucial aspect of NLP as it helps to understand the context and extract key information from the text.

This sample flow performs named entity recognition task using ChatGPT/GPT4 and prompts.

Tools used in this flow :

- python tool
- built-in llm tool

Connections used in this flow:

- azure\_open\_ai connection

## Prerequisites

Install promptflow sdk and other dependencies:

```
pip install -r requirements.txt
```

## Setup connection

Prepare your Azure Open AI resource follow this [instruction \(https://learn.microsoft.com/en-us/azure/cognitive-services/openai/how-to/create-resource?pivots=web-portal\)](https://learn.microsoft.com/en-us/azure/cognitive-services/openai/how-to/create-resource?pivots=web-portal) and get your `api_key` if you don't have one.

Note in this example, we are using [chat api \(https://learn.microsoft.com/en-us/azure/ai-services/openai/how-to/chatgpt?pivots=programming-language-chat-completions\)](https://learn.microsoft.com/en-us/azure/ai-services/openai/how-to/chatgpt?pivots=programming-language-chat-completions), please use `gpt-35-turbo` or `gpt-4` model deployment.

Create connection if you haven't done that. Ensure you have put your azure open ai endpoint key in `azure_openai.yml` (`../connections/azure_openai.yml`) file.

```
Override keys with --set to avoid yaml file changes
pf connection create -f ../../connections/azure_openai.yml --set api_key=<your_api_key> api_base=<your_api_base>
```

Ensure you have created `open_ai_connection` connection.

```
pf connection show -n open_ai_connection
```

## Run flow

### Run with single line input

```
test with default input value in flow.dag.yaml
pf flow test --flow .

test with specific input
pf flow test --flow . --inputs text='The phone number (321) 654-0987 is no longer in service' entity_type='phone number'
```

run with multiple lines data

- create run

```
pf run create --flow . --data ./data.jsonl --column-mapping entity_type='${data.entity_type}' text='${data.text}' --stream
```

You can also skip providing `column-mapping` if provided data has same column name as the flow. Reference [here \(https://aka.ms/pf/column-mapping\)](https://aka.ms/pf/column-mapping) for default behavior when `column-mapping` not provided in CLI.

filepath: `promptflow/examples/flows/standard/describe-image/README.md` content: `# Describe image flow A flow that take image input, flip it horizontally and uses OpenAI GPT-4V tool to describe it.`

Tools used in this flow :

- OpenAI GPT-4V tool
- custom python Tool

Connections used in this flow:

- OpenAI Connection

## Prerequisites

Install promptflow sdk and other dependencies, create connection for OpenAI GPT-4V tool to use:

```
pip install -r requirements.txt
pf connection create --file ../../connections/azure_openai.yml --set api_key=<your_api_key> api_base=<your_api_base> name=aoai_gp
```

## Run flow

- Prepare OpenAI connection Go to "Prompt flow" "Connections" tab. Click on "Create" button, and create an "OpenAI" connection. If you do not have an OpenAI account, please refer to [OpenAI \(https://platform.openai.com/\)](https://platform.openai.com/) for more details.
- Test flow/node

```
test with default input value in flow.dag.yaml
pf flow test --flow .

test with flow inputs
pf flow test --flow . --inputs question="How many colors can you see?" input_image="https://developer.microsoft.com/_devcom/images/1"
```

- Create run with multiple lines data

```
using environment from .env file (loaded in user code: hello.py)
pf run create --flow . --data ./data.jsonl --column-mapping question='${data.question}' --stream
```

You can also skip providing `column-mapping` if provided data has same column name as the flow. Reference [here \(https://aka.ms/pf/column-mapping\)](https://aka.ms/pf/column-mapping) for default behavior when `column-mapping` not provided in CLI.

- List and show run meta



```
list created run
pf run list

get a sample run name
name=$(pf run list -r 10 | jq '[] | select(.name | contains("describe_image_variant_0")) | .name' | head -n 1 | tr -d '"')

show specific run detail
pf run show --name $name

show output
pf run show-details --name $name

visualize run in browser
pf run visualize --name $name
```

filepath: `promptflow/examples/flows/standard/conditional-flow-for-switch/README.md` content: `# Conditional flow for switch scenario`

This example is a conditional flow for switch scenario.

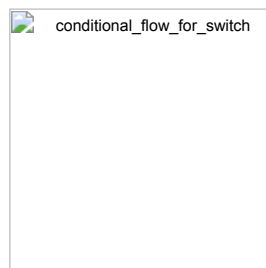
By following this example, you will learn how to create a conditional flow using the `activate` config.

## Flow description

In this flow, we set the background to the search function of a certain mall, use `activate` config to implement switch logic and determine user intent based on the input queries to achieve dynamic processing and generate user-oriented output.

- The `classify_with_llm` node analyzes user intent based on input query and provides one of the following results: "product\_recommendation," "order\_search," or "product\_info".
- The `class_check` node generates the correctly formatted user intent.
- The `product_recommendation`, `order_search`, and `product_info` nodes are configured with `activate` config and are only executed when the output from `class_check` meets the specified conditions.
- The `generate_response` node generates user-facing output.

For example, as the shown below, the input query is "When will my order be shipped" and the LLM node classifies the user intent as "order\_search", resulting in both the `product_info` and `product_recommendation` nodes being bypassed and only the `order_search` node being executed, and then generating the outputs.



## Prerequisites

Install promptflow sdk and other dependencies:

```
pip install -r requirements.txt
```

## Setup connection

Prepare your Azure Open AI resource follow this [instruction](https://learn.microsoft.com/en-us/azure/cognitive-services/openai/how-to/create-resource?pivots=web-portal) (<https://learn.microsoft.com/en-us/azure/cognitive-services/openai/how-to/create-resource?pivots=web-portal>) and get your `api_key` if you don't have one.

Note in this example, we are using `chat api` (<https://learn.microsoft.com/en-us/azure/ai-services/openai/how-to/chatgpt?pivots=programming-language-chat-completions>), please use `gpt-35-turbo` or `gpt-4` model deployment.

Create connection if you haven't done that. Ensure you have put your azure open ai endpoint key in `azure_openai.yml` (`./../connections/azure_openai.yml`) file.

```
Override keys with --set to avoid yaml file changes
pf connection create -f ./../connections/azure_openai.yml --name open_ai_connection --set api_key=<your_api_key> api_base=<your_endpoint>
```

Note in [flow.dag.yaml \(flow.dag.yaml\)](#) we are using connection named `open_ai_connection`.

```
show registered connection
pf connection show --name open_ai_connection
```

## Run flow

- Test flow

```
test with default input value in flow.dag.yaml
pf flow test --flow .

test with flow inputs
pf flow test --flow . --inputs query="When will my order be shipped?"
```

- Create run with multiple lines of data

```
create a random run name
run_name="conditional_flow_for_switch_"$(openssl rand -hex 12)

create run
pf run create --flow . --data ./data.jsonl --column-mapping query='${data.query}' --stream --name $run_name
```

- List and show run metadata

```
list created run
pf run list

show specific run detail
pf run show --name $run_name

show output
pf run show-details --name $run_name

visualize run in browser
pf run visualize --name $run_name
```

filepath: `promptflow/examples/flows/standard/basic/README.md` content: # Basic standard flow A basic standard flow using custom python tool that calls Azure OpenAI with connection info stored in environment variables.

Tools used in this flow :

- `prompt tool`
- `custom python Tool`

Connections used in this flow:

- None

## Prerequisites

Install promptflow sdk and other dependencies:

```
pip install -r requirements.txt
```

## Run flow

- Prepare your Azure Open AI resource follow this [instruction \(https://learn.microsoft.com/en-us/azure/cognitive-services/openai/how-to/create-resource?pivots=web-portal\)](https://learn.microsoft.com/en-us/azure/cognitive-services/openai/how-to/create-resource?pivots=web-portal) and get your `api_key` if you don't have one.
- Setup environment variables

Ensure you have put your azure open ai endpoint key in `.env (.env)` file. You can create one refer to this [example file \(.env.example\)](#).

```
cat .env
```

- Test flow/node

```
test with default input value in flow.dag.yaml
pf flow test --flow .

test with flow inputs
pf flow test --flow . --inputs text="Java Hello World!"

test node with inputs
pf flow test --flow . --node llm --inputs prompt="Write a simple Hello World program that displays the greeting message."
```

- Create run with multiple lines data

```
using environment from .env file (loaded in user code: hello.py)
pf run create --flow . --data ./data.jsonl --column-mapping text='${data.text}' --stream
```

You can also skip providing `column-mapping` if provided data has same column name as the flow. Reference [here \(https://aka.ms/pf/column-mapping\)](https://aka.ms/pf/column-mapping) for default behavior when `column-mapping` not provided in CLI.

- List and show run meta

```
list created run
pf run list

get a sample run name
name=$(pf run list -r 10 | jq '.[0] | select(.name | contains("basic_variant_0")) | .name' | head -n 1 | tr -d '"')

show specific run detail
pf run show --name $name

show output
pf run show-details --name $name

visualize run in browser
pf run visualize --name $name
```

## Run flow with connection

Storing connection info in `.env` with plaintext is not safe. We recommend to use `pf connection` to guard secrets like `api_key` from leak.

- Show or create `open_ai_connection`

```
create connection from `azure_openai.yaml` file
Override keys with --set to avoid yaml file changes
pf connection create --file ../../connections/azure_openai.yaml --set api_key=<your_api_key> api_base=<your_api_base>

check if connection exists
pf connection show -n open_ai_connection
```

- Test using connection secret specified in environment variables **Note:** we used `'` to wrap value since it supports raw value without escape in powershell & bash. For windows command prompt, you may remove the `'` to avoid it become part of the value.

```
test with default input value in flow.dag.yaml
pf flow test --flow . --environment-variables AZURE_OPENAI_API_KEY='${open_ai_connection.api_key}' AZURE_OPENAI_API_BASE='${open_ai_connection.api_base}'
```

- Create run using connection secret binding specified in environment variables, see [run.yml \(run.yml\)](#).

```
create run
pf run create --flow . --data ./data.jsonl --stream --environment-variables AZURE_OPENAI_API_KEY='${open_ai_connection.api_key}' AZURE_OPENAI_ENDPOINT='${open_ai_connection.endpoint}'

create run using yaml file
pf run create --file run.yaml --stream

show outputs
name=$(pf run list -r 10 | jq '.[0] | select(.name | contains("basic_variant_0")) | .name' | head -n 1 | tr -d '"')
pf run show-details --name $name
```

## Run flow in cloud with connection

- Assume we already have a connection named `open_ai_connection` in workspace.

```
set default workspace
az account set -s <your_subscription_id>
az configure --defaults group=<your_resource_group_name> workspace=<your_workspace_name>
```

- Create run

```
run with environment variable reference connection in azureml workspace
pfazure run create --flow . --data ./data.jsonl --environment-variables AZURE_OPENAI_API_KEY='${open_ai_connection.api_key}' AZURE_OPENAI_ENDPOINT='${open_ai_connection.endpoint}'

run using yaml file
pfazure run create --file run.yaml --stream
```

- List and show run meta

```
list created run
pfazure run list -r 3

get a sample run name
name=$(pfazure run list -r 100 | jq '.[0] | select(.name | contains("basic_variant_0")) | .name' | head -n 1 | tr -d '"')

show specific run detail
pfazure run show --name $name

show output
pfazure run show-details --name $name

visualize run in browser
pfazure run visualize --name $name
```

filepath: `promptflow/examples/flows/standard/conditional-flow-for-if-else/README.md` content: # Conditional flow for if-else scenario

This example is a conditional flow for if-else scenario.

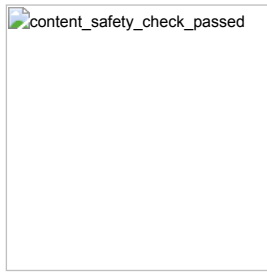
By following this example, you will learn how to create a conditional flow using the `activate` config.

## Flow description

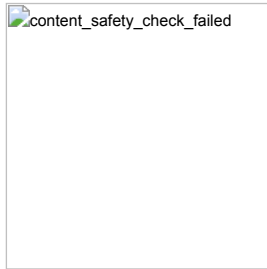
In this flow, it checks if an input query passes content safety check. If it's denied, we'll return a default response; otherwise, we'll call LLM to get a response and then summarize the final results.

The following are two execution situations of this flow:

- if input query passes content safety check:



- else:



**Notice:** The `content_safety_check` and `llm_result` node in this flow are dummy nodes that do not actually use the content safety tool and LLM tool. You can replace them with the real ones. Learn more: [LLM Tool \(https://microsoft.github.io/promptflow/reference/tools-reference/llm-tool.html\)](https://microsoft.github.io/promptflow/reference/tools-reference/llm-tool.html).

## Prerequisites

Install promptflow sdk and other dependencies:

```
pip install -r requirements.txt
```

## Run flow

- Test flow

```
test with default input value in flow.dag.yaml
pf flow test --flow .

test with flow inputs
pf flow test --flow . --inputs question="What is Prompt flow?"
```

- Create run with multiple lines of data

```
create a random run name
run_name="conditional_flow_for_if_else_"$(openssl rand -hex 12)

create run
pf run create --flow . --data ./data.jsonl --column-mapping question='${data.question}' --stream --name $run_name
```

- List and show run metadata

```
list created run
pf run list

show specific run detail
pf run show --name $run_name

show output
pf run show-details --name $run_name

visualize run in browser
pf run visualize --name $run_name
```

filepath: promptflow/examples/flows/evaluation/eval-groundedness/gpt-groundedness.md content: user:

# Instructions

- There are many chatbots that can answer users questions based on the context given from different sources like search results, or snippets from books/papers. They try to understand users's question and then get context by either performing search from search engines, databases or books/papers for relevant content. Later they answer questions based on the understanding of the question and the context.
- Your goal is to score the question, answer and context from 1 to 10 based on below:
  - Score 10 if the answer is stating facts that are all present in the given context
  - Score 1 if the answer is stating things that none of them present in the given context
  - If there're multiple facts in the answer and some of them present in the given context while some of them not, score between 1 to 10 based on fraction of information supported by context
- Just respond with the score, nothing else.

## Real work

### Question

{{question}}

### Answer

{{answer}}

### Context

{{context}}

### Score

filepath: promptflow/examples/flows/evaluation/eval-groundedness/README.md content: # Groundedness Evaluation

This is a flow leverage llm to eval groundedness: whether answer is stating facts that are all present in the given context.

Tools used in this flow :

- python tool
- built-in llm tool

## 0. Setup connection

Prepare your Azure Open AI resource follow this [instruction \(https://learn.microsoft.com/en-us/azure/cognitive-services/openai/how-to/create-resource?pivots=web-portal\)](https://learn.microsoft.com/en-us/azure/cognitive-services/openai/how-to/create-resource?pivots=web-portal) and get your api\_key if you don't have one.

```
Override keys with --set to avoid yaml file changes
pf connection create --file ../../connections/azure_openai.yaml --set api_key=<your_api_key> api_base=<your_api_base>
```

## 1. Test flow/node

```
test with default input value in flow.dag.yaml
pf flow test --flow .
```

## 2. create flow run with multi line data

```
pf run create --flow . --data ./data.jsonl --column-mapping question='${data.question}' answer='${data.answer}' context='${data.context}'
```

You can also skip providing column-mapping if provided data has same column name as the flow. Reference [here \(https://aka.ms/pf/column-mapping\)](https://aka.ms/pf/column-mapping) for default behavior when column-mapping not provided in CLI.

filepath: promptflow/examples/flows/evaluation/eval-classification-accuracy/README.md content: # Classification Accuracy Evaluation

This is a flow illustrating how to evaluate the performance of a classification system. It involves comparing each prediction to the groundtruth and assigns a "Correct" or "Incorrect" grade, and aggregating the results to produce metrics such as accuracy, which reflects how good the system is at classifying the data.

Tools used in this flow :

- python tool

## What you will learn

In this flow, you will learn

- how to compose a point based evaluation flow, where you can calculate point-wise metrics.
- the way to log metrics. use `from promptflow import log_metric`
  - see file [calculate\\_accuracy.py](#) ([calculate\\_accuracy.py](#)).

## 0. Setup connection

Prepare your Azure Open AI resource follow this [instruction](https://learn.microsoft.com/en-us/azure/cognitive-services/openai/how-to/create-resource?pivots=web-portal) (<https://learn.microsoft.com/en-us/azure/cognitive-services/openai/how-to/create-resource?pivots=web-portal>) and get your `api_key` if you don't have one.

```
Override keys with --set to avoid yaml file changes
pf connection create --file ../../../../connections/azure_openai.yaml --set api_key=<your_api_key> api_base=<your_api_base>
```

## 1. Test flow/node

```
test with default input value in flow.dag.yaml
pf flow test --flow .

test with flow inputs
pf flow test --flow . --inputs groundtruth=APP prediction=APP

test node with inputs
pf flow test --flow . --node grade --inputs groundtruth=groundtruth prediction=prediction
```

## 2. create flow run with multi line data

There are two ways to evaluate an classification flow.

```
pf run create --flow . --data ./data.jsonl --column-mapping groundtruth='${data.groundtruth}' prediction='${data.prediction}' --stream
```

You can also skip providing `column-mapping` if provided data has same column name as the flow. Reference [here](https://aka.ms/pf/column-mapping) (<https://aka.ms/pf/column-mapping>) for default behavior when `column-mapping` not provided in CLI.

## 3. create run against other flow run

Learn more in [web-classification](#) ([./../standard/web-classification/README.md](#)).

filepath: `promptflow/examples/flows/evaluation/eval-qna-rag-metrics/README.md` content: # Q&A Evaluation:

This is a flow evaluating the Q&A RAG (Retrieval Augmented Generation) systems by leveraging the state-of-the-art Large Language Models (LLM) to measure the quality and safety of responses. Utilizing GPT model to assist with measurements aims to achieve a high agreement with human evaluations compared to traditional mathematical measurements.

## What you will learn

The Q&A RAG evaluation flow allows you to assess and evaluate your model with the LLM-assisted metrics:

**gpt\_retrieval\_score**: Measures the relevance between the retrieved documents and the potential answer to the given question in the range of 1 to 5:

- 1 means that none of the document is relevant to the question at all
- 5 means that either one of the documents or combination of a few documents is ideal for answering the given question.

**gpt\_groundedness** : Measures how grounded the factual information in the answers is against the fact from the retrieved documents. Even if answers is true, if not verifiable against context, then such answers are considered ungrounded.

Grounding score is scored on a scale of 1 to 5, with 1 being the worst and 5 being the best.

**gpt\_relevance**: Measures the answer quality against the preference answer generated by LLM with the retrieved documents in the range of 1 to 5:

- 1 means the provided answer is completely irrelevant to the reference answer.
- 5 means the provided answer includes all information necessary to answer the question based on the reference answer. If the reference answer is can not be generated since no relevant document were retrieved, the answer would be rated as 5.

## Prerequisites

- Connection: Azure OpenAI or OpenAI connection.
- Data input: Evaluating the Coherence metric requires you to provide data inputs including a question, an answer, and documents in json format.

## Tools used in this flow

- Python tool
- LLM tool

## 0. Setup connection

Prepare your Azure Open AI resource follow this [instruction \(https://learn.microsoft.com/en-us/azure/cognitive-services/openai/how-to/create-resource?pivots=web-portal\)](https://learn.microsoft.com/en-us/azure/cognitive-services/openai/how-to/create-resource?pivots=web-portal) and get your `api_key` if you don't have one.

```
Override keys with --set to avoid yaml file changes
pf connection create --file ../../connections/azure_openai.yaml --set api_key=<your_api_key> api_base=<your_api_base>
```

## 1. Test flow/node

```
test with default input value in flow.dag.yaml
pf flow test --flow .
```

## 2. Create flow run with multi line data and selected metrics

```
pf run create --flow . --data ./data.jsonl --column-mapping question='${data.question}' answer='${data.answer}' documents='${data.doc
```

You can also skip providing `column-mapping` if provided data has same column name as the flow. Reference [here \(https://aka.ms/pf/column-mapping\)](https://aka.ms/pf/column-mapping) for default behavior when `column-mapping` not provided in CLI.

## 3. Run and Evaluate your flow with this Q&A RAG evaluation flow

After you develop your flow, you may want to run and evaluate it with this evaluation flow.

Here we use the flow [basic\\_chat \(./chat/basic-chat\)](#) as the main flow to evaluate. It is a flow demonstrating how to create a chatbot with LLM. The chatbot can remember previous interactions and use the conversation history to generate next message, given a question.

### 3.1 Create a batch run of your flow

```
pf run create --flow ../../chat/basic-chat --data data.jsonl --column-mapping question='${data.question}' --name basic_chat_run --st
```

Please note that `column-mapping` is a mapping from flow input name to specified values. Please refer to [Use column mapping \(https://aka.ms/pf/column-mapping\)](https://aka.ms/pf/column-mapping) for more details.

The flow run is named by specifying `--name basic_chat_run` in the above command. You can view the run details with its run name using the command:

```
pf run show-details -n basic_chat_run
```

### 3.2 Evaluate your flow

You can use this evaluation flow to measure the quality and safety of your flow responses.



After the chat flow run is finished, you can this evaluation flow to the run:

```
pf run create --flow . --data data.jsonl --column-mapping answer='${run.outputs.answer}' documents='${data.documents}' question='!;
```

Please note the flow run to be evaluated is specified with `--run basic_chat_run`. Also same as previous run, the evaluation run is named with `--name evaluation_qa_rag`.

You can view the evaluation run details with:

```
pf run show-details -n evaluation_qa_rag
pf run show-metrics -n evaluation_qa_rag
```

filepath: promptflow/examples/flows/evaluation/eval-basic/README.md content: # Basic Eval This example shows how to create a basic evaluation flow.

Tools used in this flow :

- python tool

## Prerequisites

Install promptflow sdk and other dependencies in this folder:

```
pip install -r requirements.txt
```

## What you will learn

In this flow, you will learn

- how to compose a point based evaluation flow, where you can calculate point-wise metrics.
- the way to log metrics. use `from promptflow import log_metric`
  - see file `aggregate (aggregate.py)`.

### 1. Test flow with single line data

Testing flow/node:

```
test with default input value in flow.dag.yaml
pf flow test --flow .

test with flow inputs
pf flow test --flow . --inputs groundtruth=ABC prediction=ABC

test node with inputs
pf flow test --flow . --node line_process --inputs groundtruth=ABC prediction=ABC
```

### 2. create flow run with multi line data

There are two ways to evaluate an classification flow.

```
pf run create --flow . --data ./data.jsonl --column-mapping groundtruth='${data.groundtruth}' prediction='${data.prediction}' --stre;
```

You can also skip providing `column-mapping` if provided data has same column name as the flow. Reference [here \(https://aka.ms/pf/column-mapping\)](https://aka.ms/pf/column-mapping) for default behavior when `column-mapping` not provided in CLI.

filepath: promptflow/examples/flows/evaluation/eval-entity-match-rate/README.md content: # Entity match rate evaluation

This is a flow evaluates: entity match rate.

Tools used in this flow :

- python tool

## Prerequisites

Install promptflow sdk and other dependencies:

```
pip install -r requirements.txt
```

## 1. Test flow/node

```
test with default input value in flow.dag.yaml
pf flow test --flow .
```

## 2. create flow run with multi line data

```
pf run create --flow . --data ./data.jsonl --column-mapping ground_truth='${data.ground_truth}' entities='${data.entities}' --stream
```

You can also skip providing `column-mapping` if provided data has same column name as the flow. Reference [here \(https://aka.ms/pf/column-mapping\)](https://aka.ms/pf/column-mapping) for default behavior when `column-mapping` not provided in CLI.

filepath: promptflow/examples/flows/evaluation/eval-qna-non-rag/README.md content: # Q&A Evaluation:

This is a flow evaluating the Q&A systems by leveraging Large Language Models (LLM) to measure the quality and safety of responses. Utilizing GPT and GPT embedding model to assist with measurements aims to achieve a high agreement with human evaluations compared to traditional mathematical measurements.

# Evaluation Metrics

The Q&A evaluation flow allows you to assess and evaluate your model with the LLM-assisted metrics and f1\_score:

- **gpt\_coherence**: Measures the quality of all sentences in a model's predicted answer and how they fit together naturally.

Coherence is scored on a scale of 1 to 5, with 1 being the worst and 5 being the best.

- **gpt\_relevance**: Measures how relevant the model's predicted answers are to the questions asked.

Relevance metric is scored on a scale of 1 to 5, with 1 being the worst and 5 being the best.

- **gpt\_fluency**: Measures how grammatically and linguistically correct the model's predicted answer is.

Fluency is scored on a scale of 1 to 5, with 1 being the worst and 5 being the best

- **gpt\_similarity**: Measures similarity between user-provided ground truth answers and the model predicted answer.

Similarity is scored on a scale of 1 to 5, with 1 being the worst and 5 being the best.

- **gpt\_groundedness** (against context): Measures how grounded the model's predicted answers are against the context. Even if LLM's responses are true, if not verifiable against context, then such responses are considered ungrounded.

Groundedness metric is scored on a scale of 1 to 5, with 1 being the worst and 5 being the best.

- **ada\_similarity**: Measures the cosine similarity of ada embeddings of the model prediction and the ground truth.

ada\_similarity is a value in the range [0, 1].

- **F1-score**: Compute the f1-Score based on the tokens in the predicted answer and the ground truth.

The f1-score evaluation flow allows you to determine the f1-score metric using number of common tokens between the normalized version of the ground truth and the predicted answer.

F1-score is a value in the range [0, 1].

## Tools used in this flow

- Python tool
- LLM tool
- Embedding tool

## 0. Setup connection

Prepare your Azure Open AI resource follow this [instruction \(https://learn.microsoft.com/en-us/azure/cognitive-services/openai/how-to/create-resource?pivots=web-portal\)](https://learn.microsoft.com/en-us/azure/cognitive-services/openai/how-to/create-resource?pivots=web-portal) and get your `api_key` if you don't have one.

```
Override keys with --set to avoid yaml file changes
pf connection create --file ../../connections/azure_openai.yml --set api_key=<your_api_key> api_base=<your_api_base>
```

## 1. Test flow/node

```
test with default input value in flow.dag.yaml
pf flow test --flow .

test with flow inputs
pf flow test --flow . --inputs metrics="ada_similarity,gpt_fluency,f1_score" question="what programming language is good for learning"
```

## 2. Create flow run with multi line data and selected metrics

```
pf run create --flow . --data ./data.jsonl --column-mapping question='${data.question}' answer='${data.answer}' context='${data.context}'
```

You can also skip providing `column-mapping` if provided data has same column name as the flow. Reference [here \(https://aka.ms/pf/column-mapping\)](https://aka.ms/pf/column-mapping) for default behavior when `column-mapping` not provided in CLI.

## 3. Run and Evaluate your flow with this Q&A evaluation flow

After you develop your flow, you may want to run and evaluate it with this evaluation flow.

Here we use the flow [basic\\_chat \(./chat/basic-chat\)](#) as the flow to evaluate. It is a flow demonstrating how to create a chatbot with LLM. The chatbot can remember previous interactions and use the conversation history to generate next message, given a question.

### 3.1 Create a batch run of your flow

```
pf run create --flow ../chat/basic-chat --data data.jsonl --column-mapping question='${data.question}' --name basic_chat_run --st
```

Please note that `column-mapping` is a mapping from flow input name to specified values. Please refer to [Use column mapping \(https://aka.ms/pf/column-mapping\)](https://aka.ms/pf/column-mapping) for more details.

The flow run is named by specifying `--name basic_chat_run` in the above command. You can view the run details with its run name using the command:

```
pf run show-details -n basic_chat_run
```

### 3.2 Evaluate your flow

You can use this evaluation flow to measure the quality and safety of your flow responses.

After the chat flow run is finished, you can this evaluation flow to the run:

```
pf run create --flow . --data data.jsonl --column-mapping groundtruth='${data.ground_truth}' answer='${run.outputs.answer}' context=
```

Please note the flow run to be evaluated is specified with `--run basic_chat_run`. Also same as previous run, the evaluation run is named with `--name evaluation_qa`. You can view the evaluation run details with:

```
pf run show-details -n evaluation_qa
pf run show-metrics -n evaluation_qa
```

filepath: `promptflow/examples/flows/evaluation/eval-chat-math/README.md` content: `# Eval chat math`

This example shows how to evaluate the answer of math questions, which can compare the output results with the standard answers numerically.

Learn more on corresponding [tutorials \(./tutorials/flow-fine-tuning-evaluation/promptflow-quality-improvement.md\)](#).

Tools used in this flow :

- python tool

## Prerequisites

Install promptflow sdk and other dependencies in this folder:

```
pip install -r requirements.txt
```

## 1. Test flow with single line data

Testing flow/node:

```
test with default input value in flow.dag.yaml
pf flow test --flow .

test with flow inputs
pf flow test --flow . --inputs groundtruth=123 prediction=123

test node with inputs
pf flow test --flow . --node line_process --inputs groundtruth=123 prediction=123
```

## 2. create flow run with multi line data

There are two ways to evaluate an classification flow.

```
pf run create --flow . --data ./data.jsonl --stream
```

filepath: promptflow/examples/flows/evaluation/eval-perceived-intelligence/gpt\_perceived\_intelligence.md content: user:

# Instructions

- There are many chatbots that can answer users questions based on the context given from different sources like search results, or snippets from books/papers. They try to understand users's question and then get context by either performing search from search engines, databases or books/papers for relevant content. Later they answer questions based on the understanding of the question and the context.
- Perceived intelligence is the degree to which a bot can impress the user with its responses, by showing originality, insight, creativity, knowledge, and adaptability. Perceived intelligence can be influenced by various factors, such as the content, tone, style, and structure of the bot's responses, the relevance, coherence, and accuracy of the information the bot provides, the creativity, originality, and wit of the bot's expressions, the depth, breadth, and insight of the bot's knowledge, and the ability of the bot to adapt, learn, and use feedback.
- Your goal is to score the answer for given question and context from 1 to 10 based on perceived intelligence described above:
  - Score 10 means the answer is excellent for perceived intelligence
  - Score 1 means the answer is poor for perceived intelligence
  - Score 5 means the answer is normal for perceived intelligence
- Just respond with the score, nothing else.

# Real work

## Question

{{question}}

## Answer

{{answer}}

## Context

{{context}}

## Score

filepath: promptflow/examples/flows/evaluation/eval-perceived-intelligence/README.md content: # Perceived Intelligence Evaluation

This is a flow leverage llm to eval perceived intelligence. Perceived intelligence is the degree to which a bot can impress the user with its responses, by showing originality, insight, creativity, knowledge, and adaptability.

Tools used in this flow :

- python tool
- built-in llm tool

## 0. Setup connection

Prepare your Azure Open AI resource follow this [instruction \(https://learn.microsoft.com/en-us/azure/cognitive-services/openai/how-to/create-resource?pivots=web-portal\)](https://learn.microsoft.com/en-us/azure/cognitive-services/openai/how-to/create-resource?pivots=web-portal) and get your api\_key if you don't have one.

```
Override keys with --set to avoid yaml file changes
pf connection create --file ../../../../connections/azure_openai.yaml --set api_key=<your_api_key> api_base=<your_api_base>
```

## 1. Test flow/node

```
test with default input value in flow.dag.yaml
pf flow test --flow .
```

## 2. create flow run with multi line data

```
pf run create --flow . --data ./data.jsonl --column-mapping question='${data.question}' answer='${data.answer}' context='${data.context}'
```

You can also skip providing column-mapping if provided data has same column name as the flow. Reference [here \(https://aka.ms/pf/column-mapping\)](https://aka.ms/pf/column-mapping) for default behavior when column-mapping not provided in CLI.

filepath: promptflow/examples/tools/use-cases/dynamic-list-input-tool-showcase/README.md content: # Basic flow with tool using a dynamic list input This is a flow demonstrating how to use a tool with a dynamic list input.

Tools used in this flow:

- python Tool

Connections used in this flow:

- None

## Prerequisites

Install promptflow sdk and other dependencies:

```
pip install -r requirements.txt
```

## Run flow

- Test flow

```
pf flow test --flow .
```

filepath: promptflow/examples/tools/use-cases/custom-strong-type-connection-script-tool-showcase/README.md content: # Basic flow with script tool using custom strong type connection This is a flow demonstrating the use of a script tool with custom string type connection which provides a secure way to manage credentials for external APIs and data sources, and it offers an improved user-friendly and intellisense experience compared to custom connections.

Tools used in this flow :

- custom python tool

Connections used in this flow:

- custom strong type connection

## Prerequisites

Install promptflow sdk and other dependencies:

```
pip install -r requirements.txt
```

## Setup connection

Create connection if you haven't done that.

```
Override keys with --set to avoid yaml file changes
pf connection create -f custom.yml --set secrets.api_key='<your_api_key>' configs.api_base='<your_api_base>'
```

Ensure you have created `normal_custom_connection` connection.

```
pf connection show -n normal_custom_connection
```

## Run flow

### Run with single line input

```
test with default input value in flow.dag.yaml
pf flow test --flow .

test with flow inputs
pf flow test --flow . --inputs text="Promptflow"
```

### Run with multiple lines data

- create run

```
pf run create --flow . --data ./data.jsonl --stream
```

- list and show run meta

```
list created run
pf run list -r 3

get a sample run name
name=$(pf run list -r 10 | jq '.[0] | select(.name | contains("custom_strong_type")) | .name' | head -n 1 | tr -d '"')

show specific run detail
pf run show --name $name

show output
pf run show-details --name $name

visualize run in browser
pf run visualize --name $name
```

### Run with connection override

Run flow with newly created connection.

```
pf run create --flow . --data ./data.jsonl --connections my_script_tool.connection=normal_custom_connection --stream
```

filepath: `promptflow/examples/tools/use-cases/cascading-inputs-tool-showcase/README.md` content: # Basic flow with package tool using cascading inputs This is a flow demonstrating the use of a tool with cascading inputs which frequently used in situations where the selection in one input field determines what subsequent inputs should be shown, and it helps in creating a more efficient, user-friendly, and error-free input process.

Tools used in this flow :

- python Tool

Connections used in this flow:

- None

## Prerequisites

Install promptflow sdk and other dependencies:

```
pip install -r requirements.txt
```

## Run flow

- Test flow

```
pf flow test --flow .
```

filepath: promptflow/examples/tools/use-cases/custom\_llm\_tool\_showcase/README.md content: # Flow with custom\_llm tool This is a flow demonstrating how to use a custom\_llm tool, which enables users to seamlessly connect to a large language model with prompt tuning experience using a PromptTemplate.

Tools used in this flow:

- custom\_llm Tool

Connections used in this flow:

- custom connection

## Prerequisites

Install promptflow sdk and other dependencies:

```
pip install -r requirements.txt
```

## Setup connection

Create connection if you haven't done that.

```
Override keys with --set to avoid yaml file changes
pf connection create -f custom_connection.yml --set secrets.api_key=<your_api_key> configs.api_base=<your_api_base>
```

Ensure you have created basic\_custom\_connection connection.

```
pf connection show -n basic_custom_connection
```

## Run flow

- Test flow

```
pf flow test --flow .
```

filepath: promptflow/examples/tools/use-cases/custom-strong-type-connection-package-tool-showcase/README.md content: # Basic flow with package tool using custom strong type connection This is a flow demonstrating the use of a package tool with custom string type connection which provides a secure way to manage credentials for external APIs and data sources, and it offers an improved user-friendly and intellisense experience compared to custom connections.

Tools used in this flow :

- custom package tool

Connections used in this flow:

- custom strong type connection

# Prerequisites

Install promptflow sdk and other dependencies:

```
pip install -r requirements.txt
```

## Setup connection

Create connection if you haven't done that.

```
Override keys with --set to avoid yaml file changes
pf connection create -f my_custom_connection.yml --set secrets.api_key='<your_api_key>' configs.api_base='<your_api_base>'
```

Ensure you have created `my_custom_connection` connection.

```
pf connection show -n my_custom_connection
```

## Run flow

### Run with single line input

```
test with default input value in flow.dag.yaml
pf flow test --flow .

test with flow inputs
pf flow test --flow . --inputs text="Promptflow"
```

### Run with multiple lines data

- create run

```
pf run create --flow . --data ./data.jsonl --stream
```

- list and show run meta

```
list created run
pf run list -r 3

get a sample run name
name=$(pf run list -r 10 | jq '.[0] | select(.name | contains("custom_strong_type")) | .name' | head -n 1 | tr -d ' ')

show specific run detail
pf run show --name $name

show output
pf run show-details --name $name

visualize run in browser
pf run visualize --name $name
```

### Run with connection override

Run flow with newly created connection.

```
pf run create --flow . --data ./data.jsonl --connections my_package_tool.connection=my_custom_connection --stream
```

filepath: promptflow/examples/connections/README.md content: # Working with Connection This folder contains example YAML files for creating connection using pf cli. Learn more on all the [connections types](https://microsoft.github.io/promptflow/concepts/concept-connections.html) (<https://microsoft.github.io/promptflow/concepts/concept-connections.html>).



# Prerequisites

- Install promptflow sdk and other dependencies:

```
pip install -r requirements.txt
```

## Get started

- To create a connection using any of the sample `YAML` files provided in this directory, execute following command:

```
Override keys with --set to avoid yaml file changes
pf connection create -f custom.yml --set configs.key1='<your_api_key>'
pf connection create -f azure_openai.yml --set api_key='<your_api_key>'
```

- To create a custom connection using an `.env` file, execute following command:

```
pf connection create -f .env --name custom_connection
```

- To list the created connection, execute following command:

```
pf connection list
```

- To show one connection details, execute following command:

```
pf connection show --name custom_connection
```

- To update a connection that in workspace, execute following command. Currently only a few fields(description, display\_name) support update:

```
Update an existing connection with --set to override values
Update an azure open ai connection with a new api base
pf connection update -n open_ai_connection --set api_base='<your_api_base>'
Update a custom connection
pf connection update -n custom_connection --set configs.key1='<your_new_key>' secrets.key2='<your_another_key>'
```

- To delete a connection:

```
pf connection delete -n custom_connection
```

filepath: promptflow/examples/tutorials/flow-fine-tuning-evaluation/promptflow-quality-improvement.md content: ---  
resources: examples/connections/azure\_openai.yml, examples/flows/chat/basic-chat, examples/flows/chat/chat-math-variant, examples/flows/evaluation/eval-chat-math

# Tutorial: How prompt flow helps on quality improvement

This tutorial is designed to enhance your understanding of improving flow quality through prompt tuning and evaluation.

Embark on a journey to overcome the inherent randomness of Language Models (LLMs) and enhance output reliability through **prompt fine-tuning** with this comprehensive tutorial.

Explore how prompt flow can simplify this process, enabling you to swiftly build high-quality, LLM-native apps.

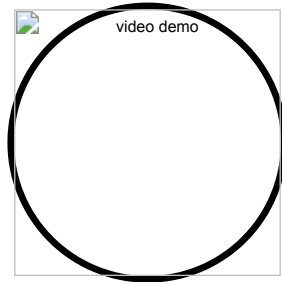
Prompt fine-tuning involves optimizing the input prompts given to an LLM. This strategic adjustment helps the model to focus on specific information needed for a task, thereby improving the accuracy and reliability of the LLM's responses.

When we talk about "high quality", it's not just about accuracy. It's equally important to strike a balance between the accuracy and the token cost of the LLM. Spend just 15 minutes with us to discover how prompt flow expedites the process of prompt tuning, testing, and evaluation, guiding you towards finding the ideal prompt (**accuracy ↑, token ↓**)



## Video tutorial

Before practicing, you can watch the video for a quick understand. This video shows how to use the **prompt flow VS code extension** to develop your chat flow, fine tune the prompt, batch test the flow, and evaluate the quality.



([http://www.youtube.com/watch?feature=player\\_embedded&v=gcle6nk2gA4](http://www.youtube.com/watch?feature=player_embedded&v=gcle6nk2gA4)).

## Hands-on practice

- Option 1 - VS Code Extension: [Install the prompt flow extension \(https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow\)](https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow) in VS Code and follow the [video tutorial \(https://youtu.be/gcle6nk2gA4\)](https://youtu.be/gcle6nk2gA4) above for a guided practice.
- Option 2 - CLI : Follow the steps below to gain hands-on experience with the prompt flow CLI.

It's time to put theory into practice! Execute our sample and witness the effects.

## Prerequisite

Before moving ahead, ensure you've completed the [Quick Start \(.\\.\\.\\README.md#get-started-with-prompt-flow-%E2%9A%A1\)](#) guidance. Ensure you have the following setup:

- [Install prompt flow \(.\\.\\.\\README.md#installation\)](#)
- [Setup a connection for your API key \(.\\.\\.\\README.md#quick-start-%E2%9A%A1\)](#)

*For testing quickly, this tutorial uses CLI command.*

Clone the promptflow repository to your local machine:

```
git clone https://github.com/microsoft/promptflow.git
```

Setup sample open\_ai\_connection connection

```
Override keys with --set to avoid yaml file changes
pf connection create --file ../../connections/azure_openai.yml --set api_key=<your_api_key> api_base=<your_api_base> --name open_ai_<
```

Next, let's get started with customizing the flow for a specific task.

## Customize the flow for a specific task

In the `promptflow/examples/flows/chat` folder, you can see a `basic-chat` folder, which represents a chat template flow as same as the one you created in the [Quick Start \(.\\.\\.\\README.md#get-started-with-prompt-flow-%E2%9A%A1\)](#) guidance. We'll use this flow as a starting point to build a math problem solver.

```
cd ../../flows/chat/basic-chat/
```

To enable your chatbot flow to solve math problems, you need to instruct the LLM about the task and target in the prompt. Open `chat.jinja2`, update the prompt as below:

```
system:
You are an assistant to calculate the answer to the provided math problems.
Please return the final numerical answer only, without any accompanying reasoning or explanation.

{% for item in chat_history %}
user:
{{item.inputs.question}}
assistant:
{{item.outputs.answer}}
{% endfor %}

user:
{{question}}
```

Before run, check your connection settings in `flow.dag.yaml` file. The default connection name is `open_ai_connection`, and the default model is `gpt-3.5-turbo`. If you have a different connection name or model, please modify the `flow.dag.yaml` file accordingly.

► (click to toggle details) For example, if you use Azure Open AI, please modify the `'flow.dag.yaml'` file to specify your connection and deployment

Go back to the `promptflow/examples/flows/chat` path, run the following command to test the flow with a simple math problem:

```
cd ..
pf flow test --flow ./basic-chat --inputs question="1+1=?"
```

This will yield the following output:

```
{
 "answer": "2"
}
```

Sometime, the question may be challenging. Now, let's test it with a complex math problem, such as:

```
pf flow test --flow ./basic-chat --inputs question="We are allowed to remove exactly one integer from the list $-1, 0, 1, 2, 3, 4, 5$,
```

The output is:

```
{
 "answer": "-1"
}
```

However, the correct answer is 5, so the output answer is incorrect! (Don't be surprised if you got the correct answer, as the randomness of LLM. You can try multiple times for different answers.) It indicates that we need to further evaluate the performance. Therefore, in the next step, we will test the flow with more math problems to better evaluate the quality.

## Evaluate the quality of your prompt

With prompt flow, you can quickly trigger a batch-run to test your prompt with a larger dataset, and evaluate the quality of the answers.

There is a `data.jsonl` file in the `promptflow/examples/flows/chat/chat-math-variant` folder, which is a dataset containing 20 test data entries (a subset of [the Math Dataset \(https://github.com/hendrycks/math/\)](https://github.com/hendrycks/math/)). It includes the input question, the ground truth for numerical answer, and the reasoning (`raw_answer`). Here's one example:

```
{
 "question": "Determine the number of ways to arrange the letters of the word PROOF.",
 "answer": "60",
 "raw_answer": "There are two O's and five total letters, so the answer is $\frac{5!}{2!} = \boxed{60}$."
}
```

Run the following command to test your prompt with this dataset:

First, set the environment variable `base_run_name` to specify the run name.

```
base_run_name="base_run"
```

***I The default model is gpt-turbo-3.5, let's try gpt-4 to see if it's smarter to get better results. Use --connections <node\_name>.connection=<connection\_name>...to specify.***

```
pf run create --flow ./basic-chat --data ./chat-math-variant/data.jsonl --column-mapping question='${data.question}' chat_history=[]
```

***i** For Azure Open AI, run the following command instead:*

```
pf run create --flow ./chat_math_variant --data test_data.jsonl --column-mapping question='${data.question}' chat_history=[] --co
```

► For Windows CMD users, run `cmdn` in toggle

---

*i* The run name must be unique. Please specify a new name in `--name`. If you see "Run 'base\_run' already exists.", you can specify another name. But please remember the name you specified, because you'll need it in the next step.

When it completes, you can run the following command to see the details of results:

*Specify the run name of your completed run in `--name` argument:*

```
pf run show-details --name $base_run_name
```

This can show the line by line input and output of the run:

	inputs.chat	inputs.question	inputs.line	outputs.ans
	_history		_number	wer
0	[]	Compute $\$ \backslash dbi$	0	4368
		nom{16}{5}\$.		
1	[]	Determine the	1	60
		number of		
		ways to		
		arrange the		
		letters of		
		the word		
		PROOF.		
..	...	...	...	...

Run the following command to create an evaluation run:

```
eval_run_name="eval_run"
pf run create --flow ./eval-chat-math --data ../chat/chat-math-variant/data.jsonl --column-mapping groundtruth='${data.answer}' pred:
```

► For Windows CMD users, run commnad in toggle

*If needed, specify the run name which you want to evaluate in `--run` argument, and specify this evaluation run name in `--name` argument.*

Then get metrics of the `eval_run`:

```
pf run show-metrics --name $eval_run_name
```

► For Windows CMD users, run commnad in toggle

You can visualize and compare the output line by line of `base_run` and `eval_run` in a web browser:

```
pf run visualize --name "$base_run_name,$eval_run_name"
```

► For Windows CMD users, run commnad in toggle

Because of the randomness of the LLM, the accuracy may vary. For example, in my run, the metrics are as follows:

```
{
 "accuracy": 0.35,
 "error_rate": 0.65
}
```

Oops! The accuracy isn't satisfactory. It's time to fine-tune your prompt for higher quality!

## Fine-tuning your prompt and evaluate the improvement

In the `/chat` folder, you can see a `chat-math-variant` folder, which represents a flow with two additional prompt variants compared to the original one you customized based on the `basic-chat`.

In this sample flow, you'll find three Jinja files:

- `chat.jinja2` is the original prompt as same as the one you customized in `basic-chat`.
- `chat_variant_1.jinja2` and `chat_variant_2.jinja2` are the 2 additional prompt variants.

We leverage the Chain of Thought (CoT) prompt engineering method to adjust the prompt. The goal is to activate the Language Model's reasoning capability of the questions, by providing a few CoT examples.

- Variant\_1: 2 CoT examples
- Variant\_2: 6 CoT examples.

These two jinja files are specified in the `flow.dag.yaml` file, which defines the flow structure. You can see that the `chat` node has 3 variants, which point to these 3 Jinja files.

## Test and evaluate your prompt variants

First, you need to modify your flow to add two more prompt variants into the `chat` node, in addition to the existed default one. In the `flow.dag.yaml` file, you can see 3 variants definition of the `chat` node, which point to these 3 Jinja files.

Run the CLI command below to start the experiment: test all variants, evaluate them, get the visualized comparison results of the experiment.

***By default, the connection is set to `open_ai_connection` and the model is set to `gpt-4` for each variant, as specified in the `flow.dag.yaml` file. However, you have the flexibility to specify a different connection and model by adding `--connections chat.connection=<your_connection_name> chat.deployment_name=<model_name>` in the test run command.***

Navigate to the `promptflow/examples/flows` folder

```
cd ..
```

Set the environment variable `base_run_name` and `eval_run_name` to specify the run name.

```
base_run_name="base_run_variant_"
eval_run_name="eval_run_variant_"
```

► For Windows CMD users, run command in toggle

Run the following command to test and evaluate the variants:

```
Test and evaluate variant_0:
Test-run
pf run create --flow ./chat/chat-math-variant --data ./chat/chat-math-variant/data.jsonl --column-mapping question='${data.question}'
Evaluate-run
pf run create --flow ./evaluation/eval-chat-math --data ./chat/chat-math-variant/data.jsonl --column-mapping groundtruth='${data.answer}'

Test and evaluate variant_1:
Test-run
pf run create --flow ./chat/chat-math-variant --data ./chat/chat-math-variant/data.jsonl --column-mapping question='${data.question}'
Evaluate-run
pf run create --flow ./evaluation/eval-chat-math --data ./chat/chat-math-variant/data.jsonl --column-mapping groundtruth='${data.answer}'

Test and evaluate variant_2:
Test-run
pf run create --flow ./chat/chat-math-variant --data ./chat/chat-math-variant/data.jsonl --column-mapping question='${data.question}'
Evaluate-run
pf run create --flow ./evaluation/eval-chat-math --data ./chat/chat-math-variant/data.jsonl --column-mapping groundtruth='${data.answer}'
```

► For Windows CMD users, run command in toggle

Get metrics of the all evaluations:

```
pf run show-metrics --name "${eval_run_name}0"
pf run show-metrics --name "${eval_run_name}1"
pf run show-metrics --name "${eval_run_name}2"
```

You may get the familiar output like this:

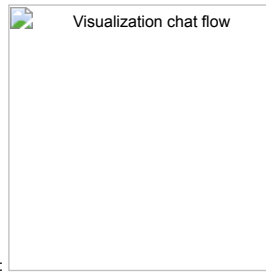
```
eval_variant_0_run
{
 "accuracy": 0.3,
 "error_rate": 0.7
}
eval_variant_1_run
{
 "accuracy": 0.9,
 "error_rate": 0.1
}
eval_variant_2_run
{
 "accuracy": 0.9,
 "error_rate": 0.1
}
```

Visualize the results:

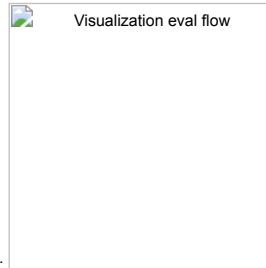
```
pf run visualize --name "${base_run_name}0,${eval_run_name}0,${base_run_name}1,${eval_run_name}1,${base_run_name}2,${eval_run_name}2"
```

► For Windows CMD users, run command in toggle

Click the HTML link, to get the experiment results. Click on column in the **Output** table will allow you to view the snapshot of each line.



The snapshot of chat flow:



The snapshot of evaluation flow:

Excellent! Now you can compare their performances and token costs, and choose the prompt that best suits your needs. We can see that variant\_1 and variant\_2 have the same accuracy, but variant\_1 has a lower token cost (only 2 few shots rather than the 6 in variant\_2). So variant\_1 is the best choice for the quality and cost balance.

## Conclusion

Great! Now you can compare their performances and token costs to choose the prompt that best suits your needs. Upon comparison, we can observe that variant\_1 and variant\_2 have the similar accuracy. However, variant\_1 stands out as the better choice due to its lower token cost (2 few-shots vs. 6 few-shots).



It is evident that adding more CoT examples in the prompt does not necessarily improve the accuracy further. Instead, we should identify the optimal point where the number of shots maximizes accuracy while minimizing cost.

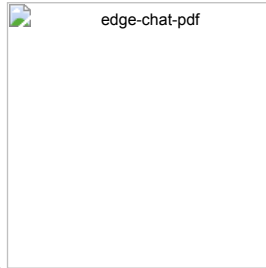
Just in a few steps, we identified that variant\_1 strikes the ideal balance between quality and cost! This is where the value of prompt tuning and evaluation using prompt flow becomes apparent. With prompt flow, you can easily test and evaluate different prompt variants, enabling you to facilitate high quality LLM-native apps to production.

filepath: `promptflow/examples/tutorials/e2e-development/chat-with-pdf.md` content: --- resources:  
examples/connections/azure\_openai.yml,  
examples/flows/chat/chat-with-pdf

# Tutorial: Chat with PDF

## Overview

Retrieval Augmented Generation (or RAG) has become a prevalent pattern to build intelligent application with Large Language Models (or LLMs) since it can infuse external knowledge into the model, which is not trained with those up-to-date or proprietary information. The screenshot below shows how new Bing in Edge sidebar can answer questions



based on the page content on the left - in this case, a PDF file.

Note that new Bing will also search web for more information to generate the answer, let's ignore that part for now.

In this tutorial we will try to mimic the functionality of retrieval of relevant information from the PDF to generate an answer with GPT.

#### We will guide you through the following steps:

Creating a console chatbot "chat\_with\_pdf" that takes a URL to a PDF file as an argument and answers questions based on the PDF's content. Constructing a prompt flow for the chatbot, primarily reusing the code from the first step. Creating a dataset with multiple questions to swiftly test the flow. Evaluating the quality of the answers generated by the chat\_with\_pdf flow. Incorporating these tests and evaluations into your development cycle, including unit tests and CI/CD. Deploying the flow to Azure App Service and Streamlit to handle real user traffic.

## Prerequisite

To go through this tutorial you should:

### 1. Install dependencies

```
cd ../../flows/chat/chat-with-pdf/
pip install -r requirements.txt
```

1. Install and configure [Prompt flow for VS Code extension](https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow) (https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow) follow [Quick Start Guide](https://microsoft.github.io/promptflow/how-to-guides/quick-start.html) (https://microsoft.github.io/promptflow/how-to-guides/quick-start.html). (This extension is optional but highly recommended for flow development and debugging.)
2. Deploy an OpenAI or Azure OpenAI chat model (e.g. gpt4 or gpt-35-turbo-16k), and an Embedding model (text-embedding-ada-002). Follow the [how-to](https://learn.microsoft.com/en-us/azure/ai-services/openai/how-to/create-resource?pivots=web-portal) (https://learn.microsoft.com/en-us/azure/ai-services/openai/how-to/create-resource?pivots=web-portal) for an Azure OpenAI example.

## Console chatbot chat\_with\_pdf

A typical RAG process consists of two steps:

- **Retrieval:** Retrieve contextual information from external systems (database, search engine, files, etc.)
- **Generation:** Construct the prompt with the retrieved context and get response from LLMs.

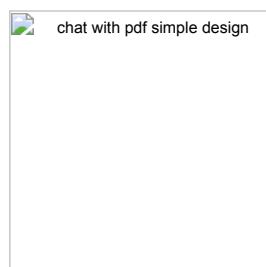
The retrieval step, being more of a search problem, can be quite complex. A widely used, simple yet effective approach is vector search, which requires an index building process.

Suppose you have one or more documents containing the contextual information, the index building process would look something like this:

1. **Chunk:** Break down the documents into multiple chunks of text.
2. **Embedding:** Each text chunk is then processed by an embedding model to convert it into an array of floating-point numbers, also known as embedding or vector.
3. **Indexing:** These vectors are then stored in an index or a database that supports vector search. This allows for the retrieval of the top K relevant or similar vectors from the index or database.

Once the index is built, the **Retrieval** step simply involves converting the question into an embedding/vector and performing a vector search on the index to obtain the most relevant context for the question.

OK now back to the chatbot we want to build, a simplified design could be:





A more robust or practical application might consider using an external vector database to store the vectors. For this simple example we're using a [FAISS](https://github.com/facebookresearch/faiss) (<https://github.com/facebookresearch/faiss>) index, which can be saved as a file. However, a more robust or practical application should consider using an external vector database with advanced management capabilities to store the vectors. With this sample's FAISS index, to prevent repetitive downloading and index building for same PDF file, we will add a check that if the PDF file already exists then we won't download, same for index building.

This design is quite effective for question and answering, but it may fall short when it comes to multi-turn conversations with the chatbot. Consider a scenario like this:

*\$User: what is BERT?*

*\$Bot: BERT stands for Bidirectional Encoder Representations from Transformers.*

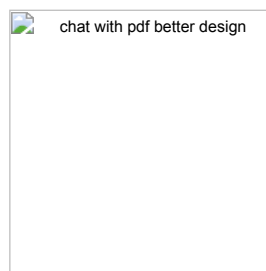
*\$User: is it better than GPT?*

*\$Bot: ...*

You would typically expect the chatbot to be intelligent enough to decipher that the "it" in your second question refers to BERT, and your actual question is "is BERT better than GPT".

However, if you present the question "is it better than GPT" to the embedding model and then to the vector index/database, they won't recognize that "it" represents BERT.

Consequently, you won't receive the most relevant context from the index. To address this issue, we will enlist the assistance of a Large Language Model (LLM), such as GPT, to "rewrite" the question based on the previous question. The updated design is as follows:



A "rewrite\_question" step is performed before feeding the question to "find\_context" step.

## Configurations

Despite being a minimalistic LLM application, there are several aspects we may want to adjust or experiment with in the future. We'll store these in environment variables for ease of access and modification. In the subsequent sections, we'll guide you on how to experiment with these configurations to enhance your chat application's quality.

Create a .env file in the second chat\_with\_pdf directory (same directory with the main.py) and populate it with the following content. We can use the load\_dotenv() function (from the python-dotenv package) to import these into our environment variables later on. We'll delve into what these variables represent when discussing how each step of the process is implemented.

Rename the .env.example file in chat\_with\_pdf directory and modify per your need.

*If you're using Open AI, your .env should look like:*

```
OPENAI_API_KEY=<open_ai_key>
EMBEDDING_MODEL_DEPLOYMENT_NAME=<text-embedding-ada-002>
CHAT_MODEL_DEPLOYMENT_NAME=<gpt-4>
PROMPT_TOKEN_LIMIT=3000
MAX_COMPLETION_TOKENS=1024
CHUNK_SIZE=256
CHUNK_OVERLAP=64
VERBOSE=False
```

Note: if you have an org id, it can be set via OPENAI\_ORG\_ID=<your\_org\_id>

*If you're using Azure Open AI, you .env should look like:*

```
OPENAI_API_TYPE=azure
OPENAI_API_BASE=<AOAI_endpoint>
OPENAI_API_KEY=<AOAI_key>
OPENAI_API_VERSION=2023-05-15
EMBEDDING_MODEL_DEPLOYMENT_NAME=<text-embedding-ada-002>
CHAT_MODEL_DEPLOYMENT_NAME=<gpt-4>
PROMPT_TOKEN_LIMIT=3000
MAX_COMPLETION_TOKENS=1024
CHUNK_SIZE=256
CHUNK_OVERLAP=64
VERBOSE=False
```

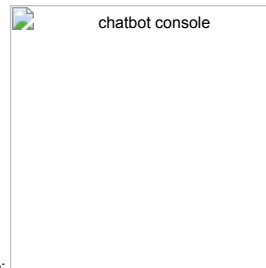
Note: CHAT\_MODEL\_DEPLOYMENT\_NAME should point to a chat model like gpt-3.5-turbo or gpt-4, OPENAI\_API\_KEY should use the deployment key, and EMBEDDING\_MODEL\_DEPLOYMENT\_NAME should point to a text embedding model like text-embedding-ada-002.

## Take a look at the chatbot in action!

You should be able to run the console app by:

```
python chat_with_pdf/main.py https://arxiv.org/pdf/1810.04805.pdf
```

Note: <https://arxiv.org/pdf/1810.04805.pdf> (<https://arxiv.org/pdf/1810.04805.pdf>) is the paper about one of the most famous earlier LLMs: BERT.



It looks like below if everything goes fine:

Now, let's delve into the actual code that implements the chatbot.

## Implementation of each steps

Download pdf: [download.py \(./flows/chat/chat-with-pdf/chat\\_with\\_pdf/download.py\)](#).

The downloaded PDF file will be stored into a temp folder.

Build index: [build\\_index.py \(./flows/chat/chat-with-pdf/chat\\_with\\_pdf/build\\_index.py\)](#).

Several libraries are used in this step to build index:

1. PyPDF2 for extraction of text from the PDF file.
  2. OpenAI python library for generating embeddings.
  3. The FAISS library is utilized to build a vector index and save it to a file. It's important to note that an additional dictionary is used to maintain the mapping from the vector index to the actual text snippet. This is because when we later attempt to query for the most relevant context, we need to locate the text snippets, not just the embedding or vector.
- The environment variables used in this step:

- OPENAI\_API\_\* and EMBEDDING\_MODEL\_DEPLOYMENT\_NAME: to access the Azure OpenAI embedding model
- CHUNK\_SIZE and CHUNK\_OVERLAP: controls how to split the PDF file into chunks for embedding

Rewrite question: [rewrite\\_question.py \(./flows/chat/chat-with-pdf/chat\\_with\\_pdf/rewrite\\_question.py\)](#).

This step is to use ChatGPT/GPT4 to rewrite the question to be better fit for finding relevant context from the vector index. The prompt file [rewrite\\_question.md \(./flows/chat/chat-with-pdf/chat\\_with\\_pdf/rewrite\\_question\\_prompt.md\)](#) should give you a better idea how it works.

Find context: [find\\_context.py \(./flows/chat/chat-with-pdf/chat\\_with\\_pdf/find\\_context.py\)](#).

In this step we load the FAISS index and the dict that were built in the "build index" step. We then turn the question into a vector using the same embedding function in the build index step. There is a small trick in this step to make sure the context will not exceed the token limit of model input prompt ([aoai model max request tokens \(https://learn.microsoft.com/en-us/azure/ai-services/openai/concepts/models\)](https://learn.microsoft.com/en-us/azure/ai-services/openai/concepts/models), OpenAI has similar limit). The output of this step is the final prompt that QnA step will send to the chat model. The

PROMPT\_TOKEN\_LIMIT environment variable decides how big the context is.

QnA: [qna.py \(../flows/chat/chat-with-pdf/chat\\_with\\_pdf/qna.py\)](#)

Use OpenAI's ChatGPT or GPT4 model and ChatCompletion API to get an answer with the previous conversation history and context from PDF.

The main loop: [main.py \(../flows/chat/chat-with-pdf/chat\\_with\\_pdf/main.py\)](#)

This is the main entry of the chatbot, which includes a loop that reads questions from user input and subsequently calls the steps mentioned above to provide an answer.

To simplify this example, we store the downloaded file and the constructed index as local files. Although there is a mechanism in place to utilize cached files/indices, loading the index still takes a certain amount of time and contributes to a latency that users may notice. Moreover, if the chatbot is hosted on a server, it requires requests for the same PDF file to hit the same server node in order to effectively use the cache. In a real-world scenario, it's likely preferable to store the index in a centralized service or database. There're many such database available, such as [Azure Cognitive Search \(https://learn.microsoft.com/en-us/azure/search/vector-search-overview\)](#), [Pinecone \(https://www.pinecone.io/\)](#), [Qdrant \(https://qdrant.tech/\)](#), ...

## Prompt flow: when you start considering the quality of your LLM app

Having a functioning chatbot is a great start, but it's only the beginning of the journey. Much like any application based on machine learning, the development of a high-quality LLM app usually involves a substantial amount of tuning. This could include experimenting with different prompts such as rewriting questions or QnAs, adjusting various parameters like chunk size, overlap size, or context limit, or even redesigning the workflow (for instance, deciding whether to include the `rewrite_question` step in our example).

Appropriate tooling is essential for facilitating this experimentation and fine-tuning process with LLM apps. This is where the concept of prompt flow comes into play. It enables you to test your LLM apps by:

- Running a few examples and manually verifying the results.
- Running larger scale tests with a formal approach (using metrics) to assess your app's quality.

You may have already learned how to create a prompt flow from scratch. Building a prompt flow from existing code is also straightforward. You can construct a chat flow either by composing the YAML file or using the visual editor of [Visual Studio Code extension \(https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow\)](#) and create a few wrappers for existing code.

Check out below:

- [flow.dag.yaml \(../flows/chat/chat-with-pdf/flow.dag.yaml\)](#)
- [setup\\_env.py \(../flows/chat/chat-with-pdf/setup\\_env.py\)](#)
- [download\\_tool.py \(../flows/chat/chat-with-pdf/download\\_tool.py\)](#)
- [build\\_index\\_tool.py \(../flows/chat/chat-with-pdf/build\\_index\\_tool.py\)](#)
- [rewrite\\_question\\_tool.py \(../flows/chat/chat-with-pdf/rewrite\\_question\\_tool.py\)](#)
- [find\\_context\\_tool.py \(../flows/chat/chat-with-pdf/find\\_context\\_tool.py\)](#)
- [qna\\_tool.py \(../flows/chat/chat-with-pdf/qna\\_tool.py\)](#)

E.g. `build_index_tool` wrapper:

```
from promptflow import tool
from chat_with_pdf.build_index import create_faiss_index

@tool
def build_index_tool(pdf_path: str) -> str:
 return create_faiss_index(pdf_path)
```

The `setup_env` node requires some explanation: you might recall that we use environment variables to manage different configurations, including OpenAI API key in the console chatbot, in prompt flow we use [Connection \(https://microsoft.github.io/promptflow/concepts/concept-connections.html\)](#) to manage access to external services like OpenAI and support passing configuration object into flow so that you can do experimentation easier. The `setup_env` node is to write the properties from connection and configuration object into environment variables. This allows the core code of the chatbot remain unchanged.

We're using Azure OpenAI in this example, below is the shell command to do so:

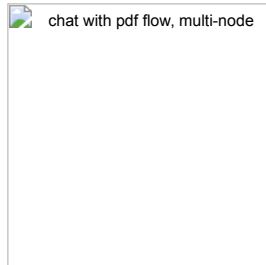
CLI

```
create connection needed by flow
if pf connection list | grep open_ai_connection; then
 echo "open_ai_connection already exists"
else
 pf connection create --file ../../connections/azure_openai.yaml --name open_ai_connection --set api_key=<your_api_key> api_base=
fi
```

If you plan to use OpenAI instead you can use below instead:

```
create connection needed by flow
if pf connection list | grep open_ai_connection; then
 echo "open_ai_connection already exists"
else
 pf connection create --file ../../connections/openai.yml --name open_ai_connection --set api_key=<your_api_key>
fi
```

The flow looks like:



## Prompt flow evaluations

Now the prompt flow for chat\_with\_pdf is created, you might have already run/debug flow through the Visual Studio Code extension. It's time to do some testing and evaluation, which starts with:

1. Create a test dataset which contains a few question and pdf\_url pairs.
2. Use existing [evaluation flows \(https://github.com/microsoft/promptflow/tree/main/examples/flows/evaluation\)](https://github.com/microsoft/promptflow/tree/main/examples/flows/evaluation) or develop new evaluation flows to generate metrics.

A small dataset can be found here: [bert-paper-qna.jsonl \(./flows/chat/chat-with-pdf/data/bert-paper-qna.jsonl\)](#) which contains around 10 questions for the BERT paper.

Evaluations are executed through 'batch runs'. Conceptually, they are a batch run of an evaluation flow which uses the previous run as input.

Here is an example of how to create a batch run for the chat\_with\_pdf flow using the test dataset and manually reviewing the output. This can be done through the Visual Studio Code extension, or CLI or Python SDK.

### batch\_run.yaml

```
name: chat_with_pdf_default_20230820_162219_559000
flow: .
data: ./data/bert-paper-qna.jsonl
#run: <Uncomment to select a run input>
column_mapping:
 chat_history: ${data.chat_history}
 pdf_url: ${data.pdf_url}
 question: ${data.question}
config:
 EMBEDDING_MODEL_DEPLOYMENT_NAME: text-embedding-ada-002
 CHAT_MODEL_DEPLOYMENT_NAME: gpt-35-turbo
 PROMPT_TOKEN_LIMIT: 3000
 MAX_COMPLETION_TOKENS: 1024
 VERBOSE: true
 CHUNK_SIZE: 256
 CHUNK_OVERLAP: 64
```

### CLI

```
run_name="chat_with_pdf_"$(openssl rand -hex 12)
pf run create --file batch_run.yaml --stream --name $run_name
```

The output will include something like below:

```
{
 "name": "chat_with_pdf_default_20230820_162219_559000",
 "created_on": "2023-08-20T16:23:39.608101",
 "status": "Completed",
 "display_name": "chat_with_pdf_default_20230820_162219_559000",
 "description": null,
 "tags": null,
 "properties": {
 "flow_path": "/Users/<user>/Work/azure-promptflow/scratchpad/chat_with_pdf",
 "output_path": "/Users/<user>/promptflow/.runs/chat_with_pdf_default_20230820_162219_559000"
 },
 "flow_name": "chat_with_pdf",
 "data": "/Users/<user>/Work/azure-promptflow/scratchpad/chat_with_pdf/data/bert-paper-qna.jsonl",
 "output": "/Users/<user>/promptflow/.runs/chat_with_pdf_default_20230820_162219_559000/ flow_outputs/output.jsonl"
}
```

Reference [here \(https://aka.ms/pf/column-mapping\)](https://aka.ms/pf/column-mapping) for default behavior when column-mapping not provided in CLI. And we developed two evaluation flows one for "[groundedness \(./flows/evaluation/eval-groundedness/\)](https://aka.ms/pf/eval-groundedness)" and one for "[perceived intelligence \(./flows/evaluation/eval-perceived-intelligence/\)](https://aka.ms/pf/eval-perceived-intelligence)". These two flows are using GPT models (ChatGPT or GPT4) to "grade" the answers. Reading the prompts will give you better idea what are these two metrics:

- [groundedness prompt \(./flows/evaluation/eval-groundedness/gpt\\_groundness.md\)](https://aka.ms/pf/eval-groundedness-gpt)
- [perceived intelligence prompt \(./flows/evaluation/eval-perceived-intelligence/gpt\\_perceived\\_intelligence.md\)](https://aka.ms/pf/eval-perceived-intelligence-gpt)

The following example creates an evaluation flow.

**eval\_run.yaml:**

```
flow: ../../evaluation/eval-groundedness
run: chat_with_pdf_default_20230820_162219_559000
column_mapping:
 question: ${run.inputs.question}
 answer: ${run.outputs.answer}
 context: ${run.outputs.context}
```

**NOTE:** the run property in eval\_run.yaml is the run name of batch\_run.yaml

**CLI:**

```
eval_run_name="eval_groundness_$(openssl rand -hex 12)
pf run create --file eval_run.yaml --run $run_name --name $eval_run_name
```

**Note:** this assumes that you have followed previous steps to create OpenAI/Azure OpenAI connection with name "open\_ai\_connection".

After the run completes you can use below commands to get detail of the runs:

```
pf run show-details --name $eval_run_name
pf run show-metrics --name $eval_run_name
pf run visualize --name $eval_run_name
```

## Experimentation!!

We have now explored how to conduct tests and evaluations for prompt flow. Additionally, we have defined two metrics to gauge the performance of our chat\_with\_pdf flow. By trying out various settings and configurations, running evaluations, and then comparing the metrics, we can determine the optimal configuration for production deployment.

There are several aspects we can experiment with, including but not limited to:

- Varying prompts for the rewrite\_question and/or QnA steps.
- Adjusting the chunk size or chunk overlap during index building.
- Modifying the context limit.

These elements can be managed through the "config" object in the flow inputs. If you wish to experiment with the first point (varying prompts), you can add properties to the config object to control this behavior - simply by directing it to different prompt files.

Take a look at how we experiment with #3 in below test: [test\\_eval in tests/chat\\_with\\_pdf\\_test.py \(../flows/chat/chat-with-pdf/tests/azure\\_chat\\_with\\_pdf\\_test.py\)](#). This test will create 6 runs in total:

1. chat\_with\_pdf\_2k\_context
2. chat\_with\_pdf\_3k\_context
3. eval\_groundedness\_chat\_with\_pdf\_2k\_context
4. eval\_perceived\_intelligence\_chat\_with\_pdf\_2k\_context
5. eval\_groundedness\_chat\_with\_pdf\_3k\_context
6. eval\_perceived\_intelligence\_chat\_with\_pdf\_3k\_context

As you can probably tell through the names: run #3 and #4 generate metrics for run #1, run #5 and #6 generate metrics for run #2. You can compare these metrics to decide which performs better - 2K context or 3K context.

NOTE: [azure\\_chat\\_with\\_pdf\\_test \(../flows/chat/chat-with-pdf/tests/azure\\_chat\\_with\\_pdf\\_test.py\)](#) does the same tests but using Azure AI as backend, so you can see all the runs in a nice web portal with all the logs and metrics comparison etc.

Further reading:

- Learn [how to experiment with the chat-with-pdf flow \(../flows/chat/chat-with-pdf/chat-with-pdf.ipynb\)](#)
- Learn [how to experiment with the chat-with-pdf flow on Azure \(../flows/chat/chat-with-pdf/chat-with-pdf-azure.ipynb\)](#) so that you can collaborate with your team.

## Integrate prompt flow into your CI/CD workflow

It's also straightforward to integrate these into your CI/CD workflow using either CLI or SDK. In this example we have various unit tests to run tests/evaluations for chat\_with\_pdf flow.

Check the [test \(../flows/chat/chat-with-pdf/tests/\)](#) folder.

```
run all the tests
python -m unittest discover -s tests -p '*_test.py'
```

## Deployment

The flow can be deployed across multiple platforms, such as a local development service, within a Docker container, onto a Kubernetes cluster, etc.

The following sections will guide you through the process of deploying the flow to a Docker container, for more details about the other choices, please refer to [flow deploy docs \(https://microsoft.github.io/promptflow/how-to-guides/deploy-a-flow/index.html\)](#).

### Build a flow as docker format app

Use the command below to build a flow as docker format app:

```
pf flow build --source . --output dist --format docker
```

### Deploy with Docker

#### Build Docker image

Like other Dockerfile, you need to build the image first. You can tag the image with any name you want. In this example, we use `promptflow-serve`.

Run the command below to build image:

```
docker build dist -t chat-with-pdf-serve
```

#### Run Docker image

Run the docker image will start a service to serve the flow inside the container.

### Connections

If the service involves connections, all related connections will be exported as yaml files and recreated in containers. Secrets in connections won't be exported directly. Instead, we will export them as a reference to environment variables:

```
$schema: https://azuremlschemas.azureedge.net/promptflow/latest/OpenAIConnection.schema.json
type: open_ai
name: open_ai_connection
module: promptflow.connections
api_key: ${env:OPEN_AI_CONNECTION_API_KEY} # env reference
```

You'll need to set up the environment variables in the container to make the connections work.

Run with `docker run`

You can run the docker image directly set via below commands:

```
The started service will listen on port 8080. You can map the port to any port on the host machine as you want.
docker run -p 8080:8080 -e OPEN_AI_CONNECTION_API_KEY=<secret-value> chat-with-pdf-serve
```

### Test the endpoint

After start the service, you can open the test page at <http://localhost:8080/> and test it:



or use curl to test it from cli:

```
curl http://localhost:8080/score --data '{"question":"what is BERT?", "chat_history": [], "pdf_url": "https://arxiv.org/pdf/1810.04801.pdf"}'
```



filepath: `promptflow/examples/tutorials/flow-deploy/README.md` content: # Deploy flow as applications

This folder contains examples of how to build & deploy flow as applications like Web Application packaged in Docker format.

filepath: `promptflow/examples/tutorials/flow-deploy/distribute-flow-as-executable-app/README.md` content: --- resources:  
examples/connections/azure\_openai.yml,  
examples/flows/standard/web-classification

## Distribute flow as executable app

This example demos how to package flow as a executable app. We will use [web-classification](#) ([./../flows/standard/web-classification/README.md](#)) as example in this tutorial.

Please ensure that you have installed all the required dependencies. You can refer to the "Prerequisites" section in the README of the [web-classification](#) (<https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification/>) for a comprehensive list of prerequisites and installation instructions. And we recommend you to add a `requirements.txt` to indicate all the required dependencies for each flow.

[Pyinstaller](https://pyinstaller.org/en/stable/installation.html) (<https://pyinstaller.org/en/stable/installation.html>) is a popular tool used for converting Python applications into standalone executables. It allows you to package your Python scripts into a single executable file, which can be run on a target machine without requiring the Python interpreter to be installed. [Streamlit](https://docs.streamlit.io/library/get-started) (<https://docs.streamlit.io/library/get-started>) is an open-source Python library used for creating web applications quickly and easily. It's designed for data scientists and engineers who want to turn data scripts into shareable web apps with minimal effort. We use Pyinstaller to package the flow and Streamlit to create custom web apps. Prior to distributing the workflow, kindly ensure that you have installed them. In this example, we use PyInstaller version 5.13.2 and Streamlit version 1.26.0 within a Python 3.10.8 environment.

## Build a flow as executable format

Note that all dependent connections must be created before building as executable.

```
create connection if not created before
pf connection create --file ../../../../connections/azure_openai.yml --set api_key=<your_api_key> api_base=<your_api_base> --name open_
```

Use the command below to build a flow as executable format app:

```
pf flow build --source ../../../../flows/standard/web-classification --output target --format executable
```

## Executable format folder structure

Exported files & its dependencies are located in the same folder. The structure is as below:

- flow: the folder contains all the flow files.
- connections: the folder contains yaml files to create all related connections.
- app.py: the entry file is included as the entry point for the bundled application.
- app.spec: the spec file tells PyInstaller how to process your script.
- main.py: it will start Streamlit service and be called by the entry file.
- settings.json: a json file to store the settings of the executable application.
- build: a folder contains various log and working files.
- dist: a folder contains the executable application.
- README.md: Simple introduction of the files.

## A template script of the entry file

PyInstaller reads a spec file or Python script written by you. It analyzes your code to discover every other module and library your script needs in order to execute. Then it collects copies of all those files, including the active Python interpreter, and puts them with your script in a single folder, or optionally in a single executable file.

We provide a Python entry script named `app.py` as the entry point for the bundled app, which enables you to serve a flow folder as an endpoint.



```

import os
import sys

from promptflow_cli_pf_connection import create_connection
from streamlit.web import cli as st_cli
from streamlit.runtime import exists

from main import start

def is_yaml_file(file_path):
 _, file_extension = os.path.splitext(file_path)
 return file_extension.lower() in ('.yaml', '.yml')

def create_connections(directory_path) -> None:
 for root, dirs, files in os.walk(directory_path):
 for file in files:
 file_path = os.path.join(root, file)
 if is_yaml_file(file_path):
 create_connection(file_path)

if __name__ == "__main__":
 create_connections(os.path.join(os.path.dirname(__file__), "connections"))
 if exists():
 start()
 else:
 main_script = os.path.join(os.path.dirname(__file__), "main.py")
 sys.argv = ["streamlit", "run", main_script, "--global.developmentMode=false"]
 st_cli.main(prog_name="streamlit")

```

## A template script of the spec file

The spec file tells PyInstaller how to process your script. It encodes the script names and most of the options you give to the pyinstaller command. The spec file is actually executable Python code. PyInstaller builds the app by executing the contents of the spec file.

To streamline this process, we offer a `app.spec` spec file that bundles the application into a single file. For additional information on spec files, you can refer to the [Using Spec Files \(https://pyinstaller.org/en/stable/spec-files.html\)](https://pyinstaller.org/en/stable/spec-files.html). Please replace `{{streamlit_runtime_interpreter_path}}` with the path of streamlit runtime interpreter in your environment.

```

-*- mode: python ; coding: utf-8 -*-
from PyInstaller.utils.hooks import collect_data_files
from PyInstaller.utils.hooks import copy_metadata

datas = [('connections', 'connections'), ('flow', 'flow'), ('settings.json', '.'), ('main.py', '.'), ('{{streamlit_runtime_interpreter}}', 'streamlit_runtime_interpreter')]
datas += collect_data_files('streamlit')
datas += copy_metadata('streamlit')
datas += collect_data_files('keyrings.alt', include_py_files=True)
datas += copy_metadata('keyrings.alt')

block_cipher = None

a = Analysis(
 ['app.py', 'main.py'],
 pathex=[],
 binaries=[],
 datas=datas,
 hiddenimports=['bs4'],
 hookspath=[],
 hooksconfig={},
 runtime_hooks=[],
 excludes=[],
 win_no_prefer_redirects=False,
 win_private_assemblies=False,
 cipher=block_cipher,
 noarchive=False,
)
pyz = PYZ(a.pure, a.zipped_data, cipher=block_cipher)

exe = EXE(
 pyz,
 a.scripts,
 a.binaries,
 a.zipfiles,
 a.datas,
 [],
 name='app',
 debug=False,
 bootloader_ignore_signals=False,
 strip=False,
 upx=True,
 upx_exclude=[],
 runtime_tmpdir=None,
 console=True,
 disable_windowed_traceback=False,
 argv_emulation=False,
 target_arch=None,
 codesign_identity=None,
 entitlements_file=None,
)

```

## The bundled application using Pyinstaller

Once you've build a flow as executable format following [Build a flow as executable format](#). It will create two folders named `build` and `dist` within your specified output directory, denoted as `.`. The `build` folder houses various log and working files, while the `dist` folder contains the `app` executable application.

### Connections

If the service involves connections, all related connections will be exported as yaml files and recreated in the executable package. Secrets in connections won't be exported directly. Instead, we will export them as a reference to environment variables:

```
$schema: https://azuremlschemas.azureedge.net/promptflow/latest/OpenAIConnection.schema.json
type: open_ai
name: open_ai_connection
module: promptflow.connections
api_key: ${env:OPEN_AI_CONNECTION_API_KEY} # env reference
```

## Test the endpoint

Finally, You can distribute the bundled application `app` to other people. They can execute your program by double clicking the executable file, e.g. `app.exe` in Windows system or running the binary file, e.g. `app` in Linux system.

The development server has a built-in web page they can use to test the flow by opening 'http://localhost:8501' in the browser. The expected result is as follows: if the flow served successfully, the process will keep alive until it is killed manually.

To your users, the app is self-contained. They do not need to install any particular version of Python or any modules. They do not need to have Python installed at all.

**Note:** The executable generated is not cross-platform. One platform (e.g. Windows) packaged executable can't run on others (Mac, Linux).

## Known issues

1. Note that Python 3.10.0 contains a bug making it unsupportable by PyInstaller. PyInstaller will also not work with beta releases of Python 3.13.

filepath: promptflow/examples/tutorials/flow-deploy/create-service-with-flow/README.md content: --- resources: examples/tutorials/flow-deploy/create-service-with-flow

## Create service with flow

This example shows how to create a simple service with flow.

You can create your own service by utilize `flow-as-function`.

This folder contains a example on how to build a service with a flow. Reference [here](#) (`./simple_score.py`) for a minimal service example. The output of `score.py` will be a json serialized dictionary. You can use json parser to parse the output.

### 1. Start the service and put in background

```
nohup python simple_score.py &
Note: added this to run in our CI pipeline, not needed for user.
sleep 10
```

### 2. Test the service with request

Executing the following command to send a request to execute a flow.

```
curl -X POST http://127.0.0.1:5000/score --header "Content-Type: application/json" --data '{"flow_input": "some_flow_input", "node_id": "some_node_id"}
```

Sample output of the request:

```
{
 "output": {
 "value": "some_flow_input"
 }
}
```

Reference [here](#) (`./simple_score.py`) for more.

filepath: promptflow/examples/tutorials/flow-deploy/docker/README.md content: --- resources: examples/connections/azure\_openai.yml, examples/flows/standard/web-classification

# Deploy a flow using Docker

This example demos how to deploy flow as a docker app. We will use [web-classification \(./../flows/standard/web-classification/README.md\)](#) as example in this tutorial.

## Build a flow as docker format app

Note that all dependent connections must be created before building as docker.

```
create connection if not created before
pf connection create --file ../../connections/azure_openai.yml --set api_key=<your_api_key> api_base=<your_api_base> --name open_
```

Use the command below to build a flow as docker format app:

```
pf flow build --source ../../flows/standard/web-classification --output dist --format docker
```

## Deploy with Docker

### Build Docker image

Like other Dockerfile, you need to build the image first. You can tag the image with any name you want. In this example, we use `promptflow-serve`.

Run the command below to build image:

```
docker build dist -t web-classification-serve
```

### Run Docker image

Run the docker image will start a service to serve the flow inside the container.

### Connections

If the service involves connections, all related connections will be exported as yaml files and recreated in containers. Secrets in connections won't be exported directly. Instead, we will export them as a reference to environment variables:

```
$schema: https://azuremlschemas.azureedge.net/promptflow/latest/OpenAIConnection.schema.json
type: open_ai
name: open_ai_connection
module: promptflow.connections
api_key: ${env:OPEN_AI_CONNECTION_API_KEY} # env reference
```

You'll need to set up the environment variables in the container to make the connections work.

### Run with `docker run`

You can run the docker image directly set via below commands:

```
The started service will listen on port 8080.You can map the port to any port on the host machine as you want.
docker run -p 8080:8080 -e OPEN_AI_CONNECTION_API_KEY=<secret-value> web-classification-serve
```

### Test the endpoint

After start the service, you can use curl to test it:

```
curl http://localhost:8080/score --data '{"url":"https://play.google.com/store/apps/details?id=com.twitter.android"}' -X POST -H "C
```

filepath: promptflow/examples/tutorials/flow-deploy/kubernetes/README.md content: --- resources: examples/connections/azure\_openai.yml, examples/flows/standard/web-classification

## Deploy flow using Kubernetes

This example demos how to deploy flow as a Kubernetes app. We will use [web-classification](#) ([../flows/standard/web-classification/README.md](#)) as example in this tutorial.

Please ensure that you have installed all the required dependencies. You can refer to the "Prerequisites" section in the README of the [web-classification](#) ([../flows/standard/web-classification/README.md#Prerequisites](#)) for a comprehensive list of prerequisites and installation instructions.

### Build a flow as docker format

Note that all dependent connections must be created before building as docker.

```
create connection if not created before
pf connection create --file ../../connections/azure_openai.yml --set api_key=<your_api_key> api_base=<your_api_base> --name open_
```

Use the command below to build a flow as docker format app:

```
pf flow build --source ../../flows/standard/web-classification --output dist --format docker
```

## Deploy with Kubernetes

### Build Docker image

Like other Dockerfile, you need to build the image first. You can tag the image with any name you want. In this example, we use `web-classification-serve`.

Then run the command below:

```
cd dist
docker build . -t web-classification-serve
```

### Create Kubernetes deployment yaml.

The Kubernetes deployment yaml file acts as a guide for managing your docker container in a Kubernetes pod. It clearly specifies important information like the container image, port configurations, environment variables, and various settings. Below, you'll find a simple deployment template that you can easily customize to meet your needs.

**Note:** You need encode the secret using base64 firstly and input the `<encoded_secret>` as 'open-ai-connection-api-key' in the deployment configuration. For example, you can run below commands in linux:

```
encoded_secret=$(echo -n <your_api_key> | base64)
```

```

kind: Namespace
apiVersion: v1
metadata:
 name: web-classification

apiVersion: v1
kind: Secret
metadata:
 name: open-ai-connection-api-key
 namespace: web-classification
type: Opaque
data:
 open-ai-connection-api-key: <encoded_secret>

apiVersion: v1
kind: Service
metadata:
 name: web-classification-service
 namespace: web-classification
spec:
 type: NodePort
 ports:
 - name: http
 port: 8080
 targetPort: 8080
 nodePort: 30123
 selector:
 app: web-classification-serve-app

apiVersion: apps/v1
kind: Deployment
metadata:
 name: web-classification-serve-app
 namespace: web-classification
spec:
 selector:
 matchLabels:
 app: web-classification-serve-app
 template:
 metadata:
 labels:
 app: web-classification-serve-app
 spec:
 containers:
 - name: web-classification-serve-container
 image: web-classification-serve
 imagePullPolicy: Never
 ports:
 - containerPort: 8080
 env:
 - name: OPEN_AI_CONNECTION_API_KEY
 valueFrom:
 secretKeyRef:
 name: open-ai-connection-api-key
 key: open-ai-connection-api-key

```

## Apply the deployment.

Before you can deploy your application, ensure that you have set up a Kubernetes cluster and installed [kubectl](https://kubernetes.io/docs/reference/kubectl/) (<https://kubernetes.io/docs/reference/kubectl/>) if it's not already installed. In this documentation, we will use [Minikube](https://minikube.sigs.k8s.io/docs/) (<https://minikube.sigs.k8s.io/docs/>) as an example. To start the cluster, execute the following command:

```
minikube start
```

Once your Kubernetes cluster is up and running, you can proceed to deploy your application by using the following command:

```
kubectl apply -f deployment.yaml
```

This command will create the necessary pods to run your application within the cluster.

**Note:** You need replace <pod\_name> below with your specific pod\_name. You can retrieve it by running `kubectl get pods -n web-classification`.

## Retrieve flow service logs of the container

The `kubectl logs` command is used to retrieve the logs of a container running within a pod, which can be useful for debugging, monitoring, and troubleshooting applications deployed in a Kubernetes cluster.

```
kubectl -n web-classification logs <pod-name>
```

## Connections

If the service involves connections, all related connections will be exported as yaml files and recreated in containers. Secrets in connections won't be exported directly. Instead, we will export them as a reference to environment variables:

```
$schema: https://azuremlschemas.azureedge.net/promptflow/latest/OpenAIConnection.schema.json
type: open_ai
name: open_ai_connection
module: promptflow.connections
api_key: ${env:OPEN_AI_CONNECTION_API_KEY} # env reference
```

You'll need to set up the environment variables in the container to make the connections work.

## Test the endpoint

- Option1:

Once you've started the service, you can establish a connection between a local port and a port on the pod. This allows you to conveniently test the endpoint from your local terminal. To achieve this, execute the following command:

```
kubectl port-forward <pod_name> 8080:8080 -n web-classification
```

With the port forwarding in place, you can use the `curl` command to initiate the endpoint test:

```
curl http://localhost:8080/score --data '{"url":"https://play.google.com/store/apps/details?id=com.twitter.android"}' -X POST
```

- Option2:

`minikube service web-classification-service --url -n web-classification` runs as a process, creating a tunnel to the cluster. The command exposes the service directly to any program running on the host operating system.

The command above will retrieve the URL of a service running within a Minikube Kubernetes cluster (e.g. `http://<assigned_port>`), which you can click to interact with the flow service in your web browser. Alternatively, you can use the following command to test the endpoint:

**Note:** Minikube will use its own external port instead of nodePort to listen to the service. So please substitute <assigned\_port> with the port obtained above.

```
curl http://localhost:<assigned_port>/score --data '{"url":"https://play.google.com/store/apps/details?id=com.twitter.android"}'
```

filepath: promptflow/examples/tutorials/flow-deploy/azure-app-service/README.md content: --- resources: examples/connections/azure\_openai.yml, examples/flows/standard/web-classification

# Deploy flow using Azure App Service

This example demos how to deploy a flow using Azure App Service.

[Azure App Service \(https://learn.microsoft.com/azure/app-service/\)](https://learn.microsoft.com/azure/app-service/) is an HTTP-based service for hosting web applications, REST APIs, and mobile back ends. The scripts (deploy.sh for bash and deploy.ps1 for powershell) under this folder are here to help deploy the docker image to Azure App Service.

We will use [web-classification \(./.flows/standard/web-classification/README.md\)](#) as example in this tutorial.

## Build a flow as docker format app

Note that all dependent connections must be created before building as docker.

```
create connection if not created before
pf connection create --file ../../connections/azure_openai.yml --set api_key=<your_api_key> api_base=<your_api_base> --name open_
```

Use the command below to build a flow as docker format app:

```
pf flow build --source ../../flows/standard/web-classification --output dist --format docker
```

## Deploy with Azure App Service

The two scripts will do the following things:

1. Create a resource group if not exists.
2. Build and push the image to docker registry.
3. Create an app service plan with the give sku.
4. Create an app with specified name, set the deployment container image to the pushed docker image.
5. Set up the environment variables for the app.

Example command to use bash script:

```
bash deploy.sh --path dist -i <image_tag> --name my-app-23d8m -r <docker_registry> -g <resource_group>
```

Example command to use powershell script:

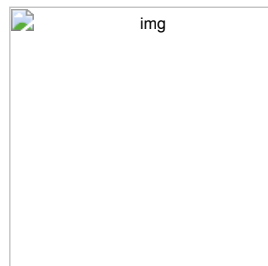
```
.\deploy.ps1 -Path dist -i <image_tag> -n my-app-23d8m -r <docker_registry> -g <resource_group>
```

Note that the name will produce a unique FQDN as AppName.azurewebsites.net.

See the full parameters by `bash deploy.sh -h` or `.\deploy.ps1 -h`.

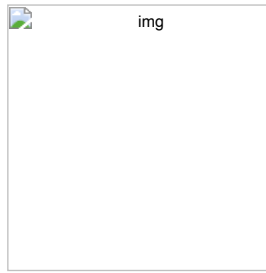
## View and test the web app

The web app can be found via [azure portal \(https://portal.azure.com/\)](https://portal.azure.com/).

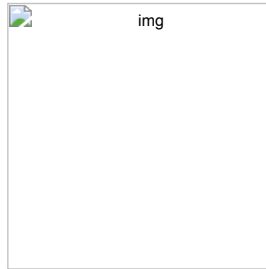


After the app created, you will need to go to [https://portal.azure.com/ \(https://portal.azure.com/\)](https://portal.azure.com/) find the app and set up the environment variables at (Settings>Configuration) or (Settings>Environment variables), then restart the app.





Browse the app at Overview and see the test page:



You can also test the app by sending a POST request to the app like:

```
curl http://<Default-domain-of-app-service>/score --data '{"url":"https://play.google.com/store/apps/details?id=com.twitter.android"'
```

Tips:

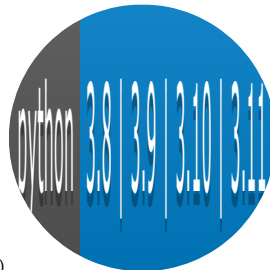
- Reach deployment logs at (Deployment>Deployment Central) and app logs at (Monitoring>Log stream).
- Reach advanced deployment tools at (Development Tools>Advanced Tools).
- Reach more details about app service at [Azure App Service](https://learn.microsoft.com/azure/app-service/) (<https://learn.microsoft.com/azure/app-service/>).

filepath: promptflow/src/promptflow/README.md content:

# Prompt flow



[.\(https://pypi.org/project/promptflow/\)](https://pypi.org/project/promptflow/)



[.\(https://pypi.python.org/pypi/promptflow/\)](https://pypi.python.org/pypi/promptflow/)



[.\(https://pypi.org/project/promptflow/\)](https://pypi.org/project/promptflow/)



[.\(https://microsoft.github.io/promptflow/reference/pf-command-](https://microsoft.github.io/promptflow/reference/pf-command-reference.html)



[reference.html\)](https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow)

[.\(https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow\)](https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow)



<https://microsoft.github.io/promptflow/index.html>



<https://github.com/microsoft/promptflow/issues/new/choose>

<https://github.com/microsoft/promptflow/issues/new/choose>



<https://github.com/microsoft/promptflow/blob/main/CONTRIBUTING.md>

<https://github.com/microsoft/promptflow/blob/main/LICENSE>



Welcome to join us to make prompt flow better by participating [discussions](https://github.com/microsoft/promptflow/discussions) (<https://github.com/microsoft/promptflow/discussions>), opening [issues](https://github.com/microsoft/promptflow/issues/new/choose) (<https://github.com/microsoft/promptflow/issues/new/choose>), submitting [PRs](https://github.com/microsoft/promptflow/pulls) (<https://github.com/microsoft/promptflow/pulls>).

**Prompt flow** is a suite of development tools designed to streamline the end-to-end development cycle of LLM-based AI applications, from ideation, prototyping, testing, evaluation to production deployment and monitoring. It makes prompt engineering much easier and enables you to build LLM apps with production quality.

With prompt flow, you will be able to:

- **Create and iteratively develop flow**
  - Create executable [flows](https://microsoft.github.io/promptflow/concepts/concept-flows.html) (<https://microsoft.github.io/promptflow/concepts/concept-flows.html>) that link LLMs, prompts, Python code and other [tools](https://microsoft.github.io/promptflow/concepts/concept-tools.html) (<https://microsoft.github.io/promptflow/concepts/concept-tools.html>) together.
  - Debug and iterate your flows, especially the [interaction with LLMs](https://microsoft.github.io/promptflow/concepts/concept-connections.html) (<https://microsoft.github.io/promptflow/concepts/concept-connections.html>) with ease.
- **Evaluate flow quality and performance**
  - Evaluate your flow's quality and performance with larger datasets.
  - Integrate the testing and evaluation into your CI/CD system to ensure quality of your flow.
- **Streamlined development cycle for production**
  - Deploy your flow to the serving platform you choose or integrate into your app's code base easily.
  - (Optional but highly recommended) Collaborate with your team by leveraging the cloud version of [prompt flow in Azure AI](https://learn.microsoft.com/en-us/azure/machine-learning/prompt-flow/overview-what-is-prompt-flow?view=azureml-api-2) (<https://learn.microsoft.com/en-us/azure/machine-learning/prompt-flow/overview-what-is-prompt-flow?view=azureml-api-2>).

## Installation

Ensure you have a python environment, `python=3.9` is recommended.

```
pip install promptflow promptflow-tools
```

## Quick Start ↵

Create a chatbot with prompt flow

Run the command to initiate a prompt flow from a chat template, it creates folder named `my_chatbot` and generates required files within it:

```
pf flow init --flow ./my_chatbot --type chat
```

#### Setup a connection for your API key

For OpenAI key, establish a connection by running the command, using the `openai.yaml` file in the `my_chatbot` folder, which stores your OpenAI key:

```
Override keys with --set to avoid yaml file changes
pf connection create --file ./my_chatbot/openai.yaml --set api_key=<your_api_key> --name open_ai_connection
```

For Azure OpenAI key, establish the connection by running the command, using the `azure_openai.yaml` file:

```
pf connection create --file ./my_chatbot/azure_openai.yaml --set api_key=<your_api_key> api_base=<your_api_base> --name open_ai_connection
```

#### Chat with your flow

In the `my_chatbot` folder, there's a `flow.dag.yaml` file that outlines the flow, including inputs/outputs, nodes, connection, and the LLM model, etc

*Note that in the `chat` node, we're using a connection named `open_ai_connection` (specified in `connection` field) and the `gpt-35-turbo` model (specified in `deployment_name` field). The `deployment_name` field is to specify the OpenAI model, or the Azure OpenAI deployment resource.*

Interact with your chatbot by running: (press `Ctrl + C` to end the session)

```
pf flow test --flow ./my_chatbot --interactive
```

Continue to delve deeper into [prompt flow](https://github.com/microsoft/promptflow) (<https://github.com/microsoft/promptflow>).

filepath: `promptflow/src/promptflow/CHANGELOG.md` content: `# Release History`

## 1.6.0 (TBD)

### Features Added

- [SDK/CLI] Support configuring environment variable to directly use `AzureCliCredential` for `pfazure` commands.

```
PF_USE_AZURE_CLI_CREDENTIAL=true
```

## 1.5.0 (2024.02.06)

### Features Added

- [SDK/CLI][azure] Support specify compute instance as session compute in `run.yaml`
- [SDK/CLI][azure] Stop support specifying `idle_time_before_shutdown_minutes` for automatic runtime since each session will be auto deleted after execution.

### Bugs Fixed

- [SDK/CLI] The inputs of node `test` allows the value of reference node output be passed directly in.
- [SDK/CLI][azure] Fixed bug for cloud batch run referencing registry flow with automatic runtime.
- [SDK/CLI] Fix "Without Import Data" in run visualize page when invalid JSON value exists in metrics.
- [SDK/CLI][azure] Fix azureml serving get UAI(user assigned identity) token failure bug.
- [SDK/CLI] Fix flow as function connection override when node has default variant.

### Improvements

- [SDK/CLI] For `pf run delete`, `pf connection delete`, introducing an option to skip confirmation prompts.
- [SDK/CLI] Move `pfs` extra dependency to required dependency.

## 1.4.0 (2024.01.22)

### Features Added

- [Executor] Calculate `system_metrics` recursively in `api_calls`.
- [Executor] Add flow root level `api_calls`, so that user can overview the aggregated metrics of a flow.
- [Executor] Add `@trace` decorator to make it possible to log traces for functions that are called by tools.
- [Tool] `InputSetting` of tool supports passing undefined configuration.
- [SDK/CLI][azure] Switch automatic runtime's session provision to system wait.
- [SDK/CLI] Add `--skip-open-browser` option to `pf flow serve` to skip opening browser.
- [SDK/CLI][azure] Support submit flow to sovereign cloud.
- [SDK/CLI] Support `pf run delete` to delete a run irreversibly.
- [SDK/CLI][azure] Automatically put `requirements.txt` to `flow.dag.yaml` if exists in flow snapshot.
- [SDK/CLI] Support `pf upgrade` to upgrade prompt flow to the latest version.
- [SDK/CLI] Support env variables in yaml file.

## Bugs Fixed

- Fix unaligned inputs & outputs or pandas exception during get details against run in Azure.
- Fix loose flow path validation for run schema.
- Fix "Without Import Data" in run visualize page results from invalid JSON value (`-Infinity`, `Infinity` and `NaN`).
- Fix "ValueError: invalid width -1" when show-details against long column(s) in narrow terminal window.
- Fix invalid tool code generated when initializing the script tool with icon.

## Improvements

- [SDK/CLI] For `pfazure flow create`:
  - If used by non-msft tenant user, use user name instead of user object id in the remote flow folder path. (e.g. `Users/<user-name>/promptflow`).
  - When flow has unknown attributes, log warning instead of raising error.
  - Use local flow folder name and timestamp as the azure flow file share folder name.
- [SDK/CLI] For `pf/pfazure run create`, when run has unknown attribute, log warning instead of raising error.
- Replace `pyyaml` with `ruamel.yaml` to adopt YAML 1.2 specification.

# 1.3.0 (2023.12.27)

## Features Added

- [SDK/CLI] Support `pfazure run cancel` to cancel a run on Azure AI.
- Add support to configure prompt flow home directory via environment variable `PF_HOME_DIRECTORY`.
  - Please set before importing `promptflow`, otherwise it won't take effect.
- [Executor] Handle `KeyboardInterrupt` in flow test so that the final state is Canceled.

## Bugs Fixed

- [SDK/CLI] Fix single node run doesn't work when consuming sub item of upstream node

## Improvements

- Change `ruamel.yaml` lower bound to 0.17.10.
- [SDK/CLI] Improve `pfazure run download` to handle large run data files.
- [Executor] Exit the process when all async tools are done or exceeded timeout after cancellation.

# 1.2.0 (2023.12.14)

## Features Added

- [SDK/CLI] Support `pfazure run download` to download run data from Azure AI.
- [SDK/CLI] Support `pf run create` to create a local run record from downloaded run data.

## Bugs Fixed

- [SDK/CLI] Removing telemetry warning when running commands.
- Empty node stdout & stderr to avoid large visualize HTML.
- Hide unnecessary fields in run list for better readability.
- Fix bug that ignores timeout lines in batch run status summary.

# 1.1.1 (2023.12.1)

## Bugs Fixed

- [SDK/CLI] Fix compatibility issue with `semantic-kernel==0.4.0.dev0` and `azure-ai-ml==1.12.0`.
- [SDK/CLI] Add back workspace information in CLI telemetry.
- [SDK/CLI] Disable the feature to customize user agent in CLI to avoid changes on operation context.
- Fix openai metrics calculator to adapt openai v1.

## 1.1.0 (2023.11.30)

### Features Added

- Add `pfazure flow show/list` to show or list flows from Azure AI.
- Display node status in run visualize page graph view.
- Add support for image input and output in prompt flow.
- [SDK/CLI] SDK/CLI will collect telemetry by default, user can use `pf config set telemetry.enabled=false` to opt out.
- Add `raise_on_error` for stream run API, by default we raise for failed run.
- Flow as function: consume a flow like a function with parameters mapped to flow inputs.
- Enable specifying the default output path for run.
  - Use `pf config set run.output_path=<output-path>` to specify, and the run output path will be `<output-path>/<run-name>`.
  - Introduce macro `${flow_directory}` for `run.output_path` in config, which will be replaced with corresponding flow directory.
  - The flow directory cannot be set as run output path, which means `pf config set run.output_path='${flow_directory}'` is invalid; but you can use child folder, e.g. `pf config set run.output_path='${flow_directory}/.runs'`.
- Support `pfazure run create` with remote flow.
  - For remote workspace flow: `pfazure run create --flow azureml:<flow-name>`
  - For remote registry flow: `pfazure run create --flow azureml://registries/<registry-name>/models/<flow-name>/versions/<flow-version>`
- Support set logging level via environment variable `PF_LOGGING_LEVEL`, valid values includes `CRITICAL`, `ERROR`, `WARNING`, `INFO`, `DEBUG`, default to `INFO`.
- Remove openai version restrictions

## Bugs Fixed

- [SDK/CLI] Fix node test with dict node input will raise "Required input(s) missing".
- [SDK/CLI] Will use run name as display name when display name not specified (used flow folder name before).
- [SDK/CLI] Fix pf flow build created unexpected layer of dist folder
- [SDK/CLI] Fix deploy prompt flow: connections value may be none

## Improvements

- Force 'az login' if using azureml connection provider in cli command.
- Add env variable 'PF\_NO\_INTERACTIVE\_LOGIN' to disable interactive login if using azureml connection provider in promptflow sdk.
- Improved CLI invoke time.
- Bump `pydash` upper bound to 8.0.0.
- Bump `SQLAlchemy` upper bound to 3.0.0.
- Bump `flask` upper bound to 4.0.0, `flask-restx` upper bound to 2.0.0.
- Bump `ruamel.yaml` upper bound to 1.0.0.

## 1.0.0 (2023.11.09)

### Features Added

- [Executor] Add `enable_kwargs` tag in tools.json for customer python tool.
- [SDK/CLI] Support `pfazure flow create`. Create a flow on Azure AI from local flow folder.
- [SDK/CLI] Changed column mapping `${run.inputs.xx}`'s behavior, it will refer to run's data columns instead of run's inputs columns.

## Bugs Fixed

- [SDK/CLI] Keep original format in run output.jsonl.
- [Executor] Fix the bug that raise an error when an aggregation node references a bypassed node

## Improvements

- [Executor] Set the outputs of the bypassed nodes as None

# 0.1.0b8 (2023.10.26)

## Features Added

- [Executor] Add average execution time and estimated execution time to batch run logs
- [SDK/CLI] Support `pfazure run archive/restore/update`.
- [SDK/CLI] Support custom strong type connection.
- [SDK/CLI] Enable telemetry and won't collect by default, use `pf config set cli.telemetry_enabled=true` to opt in.
- [SDK/CLI] Exposed function from `promptflow import load_run` to load run object from local YAML file.
- [Executor] Support `ToolProvider` for script tools.

## Bugs Fixed

- **pf config set:**
  - Fix bug for workspace `connection.provider=azureml` doesn't work as expected.
- [SDK/CLI] Fix the bug that using `sdk/cli` to submit batch run did not display the log correctly.
- [SDK/CLI] Fix encoding issues when input is non-English with `pf flow test`.
- [Executor] Fix the bug can't read file containing "Private Use" unicode character.
- [SDK/CLI] Fix string type data will be converted to integer/float.
- [SDK/CLI] Remove the max rows limitation of loading data.
- [SDK/CLI] Fix the bug `--set` not taking effect when creating run from file.

## Improvements

- [SDK/CLI] Experience improvements in `pf run visualize` page:
  - Add column status.
  - Support opening flow file by clicking run id.

# 0.1.0b7.post1 (2023.09.28)

## Bug Fixed

- Fix extra dependency bug when importing `promptflow` without `azure-ai-ml` installed.

# 0.1.0b7 (2023.09.27)

## Features Added

- **pf flow validate:** support validate flow
- **pf config set:** support set user-level promptflow config.
  - Support workspace connection provider, usage: `pf config set connection.provider=azureml://subscriptions/<subscription_id>/resourceGroups/<resource_group>/providers/Microsoft.MachineLe`
- Support override openai connection's model when submitting a flow. For example: `pf run create --flow ./ --data ./data.jsonl --connection llm.model=xxx --column-mapping url='${data.url}'`

## Bugs Fixed

- [Flow build] Fix flow build file name and environment variable name when connection name contains space.
- Reserve `.promptflow` folder when dump run snapshot.
- Read/write log file with encoding specified.
- Avoid inconsistent error message when executor exits abnormally.
- Align inputs & outputs row number in case partial completed run will break `pfazure run show-details`.
- Fix bug that failed to parse portal url for run data when the form is an asset id.
- Fix the issue of process hanging for a long time when running the batch run.

## Improvements

- [Executor][Internal] Improve error message with more details and actionable information.
- [SDK/CLI] `pf/pfazure run show-details:`
  - Add `--max-results` option to control the number of results to display.
  - Add `--all-results` option to display all results.
- Add validation for azure `PFClient` constructor in case wrong parameter is passed.

## 0.1.0b6 (2023.09.15)

### Features Added

- [promptflow][Feature] Store token metrics in run properties

### Bugs Fixed

- Refine error message body for flow\_validator.py
- Refine error message body for run\_tracker.py
- [Executor][Internal] Add some unit test to improve code coverage of log/metric
- [SDK/CLI] Update portal link to remove flight.
- [Executor][Internal] Improve inputs mapping's error message.
- [API] Resolve warnings/errors of sphinx build

## 0.1.0b5 (2023.09.08)

### Features Added

- **pf run visualize**: support lineage graph & display name in visualize page

### Bugs Fixed

- Add missing requirement `psutil` in `setup.py`

## 0.1.0b4 (2023.09.04)

### Features added

- Support `pf flow build` commands

## 0.1.0b3 (2023.08.30)

- Minor bug fixes.

## 0.1.0b2 (2023.08.29)

- First preview version with major CLI & SDK features.

### Features added

- **pf flow**: init/test/serve/export
- **pf run**: create/update/stream/list/show/show-details/show-metrics/visualize/archive/restore/export
- **pf connection**: create/update/show/list/delete
- Azure AI support:
  - **pfazure run**: create/list/stream/show/show-details/show-metrics/visualize

## 0.1.0b1 (2023.07.20)

- Stub version in Pypi.

filepath: promptflow/src/promptflow/promptflow/\_cli/data/chat\_flow/flow\_files/README.md content: # Chat flow Chat flow is designed for conversational application development, building upon the capabilities of standard flow and providing enhanced support for chat inputs/outputs and chat history management. With chat flow, you can easily create a chatbot that handles chat input and output.

## Create connection for LLM tool to use

You can follow these steps to create a connection required by a LLM tool.

Currently, there are two connection types supported by LLM tool: "AzureOpenAI" and "OpenAI". If you want to use "AzureOpenAI" connection type, you need to create an Azure OpenAI service first. Please refer to [Azure OpenAI Service \(https://azure.microsoft.com/en-us/products/cognitive-services/openai-service/\)](https://azure.microsoft.com/en-us/products/cognitive-services/openai-service/) for more details. If you want to use "OpenAI" connection type, you need to create an OpenAI account first. Please refer to [OpenAI \(https://platform.openai.com/\)](https://platform.openai.com/) for more details.

```
Override keys with --set to avoid yaml file changes
Create open ai connection
pf connection create --file openai.yaml --set api_key=<your_api_key> --name open_ai_connection

Create azure open ai connection
pf connection create --file azure_openai.yaml --set api_key=<your_api_key> api_base=<your_api_base> --name open_ai_connection
```

Note in [flow.dag.yaml \(flow.dag.yaml\)](#), we are using connection named open\_ai\_connection.

```
show registered connection
pf connection show --name open_ai_connection
```

Please refer to connections [document \(https://promptflow.azurewebsites.net/community/local/manage-connections.html\)](https://promptflow.azurewebsites.net/community/local/manage-connections.html) and [example \(https://github.com/microsoft/promptflow/tree/main/examples/connections\)](https://github.com/microsoft/promptflow/tree/main/examples/connections) for more details.

## Develop a chat flow

The most important elements that differentiate a chat flow from a standard flow are **Chat Input**, **Chat History**, and **Chat Output**.

- **Chat Input:** Chat input refers to the messages or queries submitted by users to the chatbot. Effectively handling chat input is crucial for a successful conversation, as it involves understanding user intentions, extracting relevant information, and triggering appropriate responses.
- **Chat History:** Chat history is the record of all interactions between the user and the chatbot, including both user inputs and AI-generated outputs. Maintaining chat history is essential for keeping track of the conversation context and ensuring the AI can generate contextually relevant responses. Chat History is a special type of chat flow input, that stores chat messages in a structured format.
- **Chat Output:** Chat output refers to the AI-generated messages that are sent to the user in response to their inputs. Generating contextually appropriate and engaging chat outputs is vital for a positive user experience.

A chat flow can have multiple inputs, but Chat History and Chat Input are required inputs in chat flow.

## Interact with chat flow

Promptflow CLI provides a way to start an interactive chat session for chat flow. Customer can use below command to start an interactive chat session:

```
pf flow test --flow <flow_folder> --interactive
```

After executing this command, customer can interact with the chat flow in the terminal. Customer can press **Enter** to send the message to chat flow. And customer can quit with **ctrl+C**. Promptflow CLI will distinguish the output of different roles by color, User input, Bot output, Flow script output, Node output.

```
=====
Welcome to chat flow, .
Press Enter to send your message.
You can quit with ctrl+C.
=====
User: What types of container software there are
Bot: There are several types of container software available, including:

1. Docker: This is one of the most popular containerization software that allows developers to package their applications into containers and deploy them across different environments.
2. Kubernetes: This is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications.

User: What's the different between them
Bot: The main difference between the various container software systems is their functionality and purpose. Here are some key differences between them:

1. Docker is more focused on container packaging and deployment, while Kubernetes is more focused on container orchestration and management.
2. Kubernetes: Kubernetes is a container orchestration tool that helps manage and deploy containers at scale. It automates the deployment, scaling, and management of containerized applications across multiple hosts.
```

If customer adds "--verbose" in the pf command, the output of each step will be displayed.



=====

Welcome to chat flow, Template Chat Flow.

Press Enter to send your message.

You can quit with ctrl+C.

=====

User: What types of container software there are

chat: There are several types of container software available, including:

1. Docker: A popular container platform that is widely used in the industry.
2. Kubernetes: A container orchestration tool that helps manage and deploy containers at scale.

Bot: There are several types of container software available, including:

1. Docker: A popular container platform that is widely used in the industry.
2. Kubernetes: A container orchestration tool that helps manage and deploy containers at scale.

User: What's the different between them

chat: The main differences between container software are in their architecture, feature sets, and use cases. Here are some brief explanations of the differences between the examples I listed:

1. Docker: Docker is a container platform that is widely used for building, packaging, and deploying containerized applications. It is known for its ease of use, portability, and large ecosystem of tools and services.
2. Kubernetes: Kubernetes is a container orchestration tool that helps manage and deploy containers at scale. It automates the deployment, scaling, and management of containerized applications across multiple hosts.

Bot: The main differences between container software are in their architecture, feature sets, and use cases. Here are some brief explanations of the differences between the examples I listed:

1. Docker: Docker is a container platform that is widely used for building, packaging, and deploying containerized applications. It is known for its ease of use, portability, and large ecosystem of tools and services.
2. Kubernetes: Kubernetes is a container orchestration tool that helps manage and deploy containers at scale. It automates the deployment, scaling, and management of containerized applications across multiple hosts.

filepath: promptflow/src/promptflow-tools/README.dev.md content: # Development Guide

## Prerequisites

```
pip install -r requirements.txt
pip install pytest pytest-mock
```

## Run tests

- Create connection config file by `cp connections.json.example connections.json`.
- Fill in fields manually in `connections.json`.
- `cd tests` and run `pytest -s -v` to run all tests.

## Run tests in CI

Use this [workflow](https://github.com/microsoft/promptflow/actions/workflows/tools_secret_upload.yml) ([https://github.com/microsoft/promptflow/actions/workflows/tools\\_secret\\_upload.yml](https://github.com/microsoft/promptflow/actions/workflows/tools_secret_upload.yml)) to upload secrets in key vault. The secrets you uploaded would be used in [tools tests](https://github.com/microsoft/promptflow/actions/workflows/tools_tests.yml) ([https://github.com/microsoft/promptflow/actions/workflows/tools\\_tests.yml](https://github.com/microsoft/promptflow/actions/workflows/tools_tests.yml)). Note that you only need to upload the SECRETS.

[!NOTE] After triggering the workflow, kindly request approval from Promptflow Support before proceeding further.

## PR check-in criteria

Here's a friendly heads-up! We've got some criteria for you to self-review your code changes. It's a great way to double-check your work and make sure everything is in order before you share it. Happy coding!

## Maintain code quality

The code you submit in your pull request should adhere to the following guidelines:

- **Maintain clean code:** The code should be clean, easy to understand, and well-structured to promote readability and maintainability.
- **Comment on your code:** Use comments to explain the purpose of certain code segments, particularly complex or non-obvious ones. This assists other developers in understanding your work.
- **Correct typos and grammatical errors:** Ensure that the code and file names are free from spelling mistakes and grammatical errors. This enhances the overall presentation and clarity of your code.
- **Avoid hard-coded values:** It is best to avoid hard-coding values unless absolutely necessary. Instead, use variables, constants, or configuration files, which can be easily modified without changing the source code.
- **Prevent code duplication:** Modify the original code to be more general instead of duplicating it. Code duplication can lead to longer, more complex code that is harder to maintain.
- **Implement effective error handling:** Good error handling is critical for troubleshooting customer issues and analyzing key metrics. Follow the guidelines provided in the [Error Handling Guideline \(https://msdata.visualstudio.com/Vienna/\\_git/PromptFlow?path=/docs/error\\_handling\\_guidance.md&a=preview\)](https://msdata.visualstudio.com/Vienna/_git/PromptFlow?path=/docs/error_handling_guidance.md&a=preview) and reference the [exception.py \(https://github.com/microsoft/promptflow/blob/main/src/promptflow-tools/promptflow/tools/exception.py\)](https://github.com/microsoft/promptflow/blob/main/src/promptflow-tools/promptflow/tools/exception.py) file for examples.

## Ensure high test coverage

Test coverage is crucial for maintaining code quality. Please adhere to the following guidelines:

- **Comprehensive Testing:** Include unit tests and e2e tests for any new functionality introduced.
- **Exception Testing:** Make sure to incorporate unit tests for all exceptions. These tests should verify error codes, error messages, and other important values. For reference, you can check out [TestHandleOpenAIErrors \(https://github.com/microsoft/promptflow/blob/main/src/promptflow-tools/tests/test\\_handle\\_openai\\_error.py\)](https://github.com/microsoft/promptflow/blob/main/src/promptflow-tools/tests/test_handle_openai_error.py).
- **VSCode Testing:** If you're adding a new built-in tool, make sure to test your tool within the VSCode environment prior to submitting your PR. For more guidance on this, refer to [Use your tool from VSCode Extension \(https://github.com/microsoft/promptflow/blob/main/docs/how-to-guides/develop-a-tool/create-and-use-tool-package.md#use-your-tool-from-vscode-extension\)](https://github.com/microsoft/promptflow/blob/main/docs/how-to-guides/develop-a-tool/create-and-use-tool-package.md#use-your-tool-from-vscode-extension).

## Add documents

Ensure to include documentation for your new built-in tool, following the guidelines below:

- **Error-Free Content:** Rectify all typographical and grammatical errors in the documentation. This will ensure clarity and readability.
- **Code Alignment:** The documentation should accurately reflect the current state of your code. Ensure that all described functionalities and behaviors match with your implemented code.
- **Functional Links:** Verify that all embedded links within the documentation are functioning properly, leading to the correct resources or references.

filepath: promptflow/src/promptflow-tools/README.md content: # Prompt flow tools



[\(https://pypi.org/project/promptflow-tools/\)](https://pypi.org/project/promptflow-tools/)



[\\_ \(https://github.com/microsoft/promptflow/blob/main/LICENSE\)](https://github.com/microsoft/promptflow/blob/main/LICENSE)

## Introduction

Tools are the fundamental building blocks of a flow in Azure Machine Learning prompt flow. Each tool is a simple, executable unit with a specific function, allowing users to perform various tasks. By combining different tools, users can create a flow that accomplishes a wide range of goals. One of the key benefit of prompt flow tools is their seamless integration with third-party APIs and python open source packages. This not only improves the functionality of large language models but also makes the development process more efficient.

In this package, we provide a set of builtin tools of prompt flow, which are the most commonly used tools in the development of AI applications. We also provide a flexible way for users to create their own tools and share them with others. See [Create and Use Tool Package \(https://github.com/microsoft/promptflow/blob/main/docs/how-to-guides/develop-a-tool/create-and-use-tool-package.md\)](https://github.com/microsoft/promptflow/blob/main/docs/how-to-guides/develop-a-tool/create-and-use-tool-package.md) for more details.

filepath: promptflow/src/promptflow-tools/CHANGELOG.md content: # Release History

# 1.0.0 (2023.11.30)

## Features Added

- Support openai 1.x in promptflow-tools
- Add new tool "OpenAI GPT-4V"

filepath: promptflow/docs/index.md content: --- myst: html\_meta:  
"description lang=en": "Prompt flow Doc" "google-site-  
verification": "rEZN-  
2h5TVqEco07aaMpqNcDx4bjr2czx1Hwfoxydrg"  
html\_theme.sidebar\_secondary.remove: true

# Prompt flow

**Prompt flow** (<https://github.com/microsoft/promptflow>) is a suite of development tools designed to streamline the end-to-end development cycle of LLM-based AI applications, from ideation, prototyping, testing, evaluation to production deployment and monitoring. It makes prompt engineering much easier and enables you to build LLM apps with production quality.

With prompt flow, you will be able to:

- **Create flows** ([./concepts/concept-flows.md](#)) that link [LLMs](#) ([./reference/tools-reference/llm-tool.md](#)), [prompts](#) ([./reference/tools-reference/prompt-tool.md](#)), [Python](#) ([./reference/tools-reference/python-tool.md](#)) code and other [tools](#) ([./concepts/concept-tools.md](#)) together in a executable workflow.
- **Debug and iterate your flows**, especially the interaction with LLMs with ease.
- **Evaluate your flows**, calculate quality and performance metrics with larger datasets.
- **Integrate the testing and evaluation into your CI/CD system** to ensure quality of your flow.
- **Deploy your flows** to the serving platform you choose or integrate into your app's code base easily.
- (Optional but highly recommended) **Collaborate with your team** by leveraging the cloud version of [Prompt flow in Azure AI](#) (<https://learn.microsoft.com/en-us/azure/machine-learning/prompt-flow/overview-what-is-prompt-flow?view=azureml-api-2>).

Welcome to join us to make prompt flow better by participating [discussions](#) (<https://github.com/microsoft/promptflow/discussions>), opening [issues](#) (<https://github.com/microsoft/promptflow/issues/new/choose>), submitting [PRs](#) (<https://github.com/microsoft/promptflow/pulls>).

This documentation site contains guides for prompt flow [sdk\\_cli](#) (<https://pypi.org/project/promptflow/>) and [vscode extension](#) (<https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow>) users.

```
:grid-columns: 1 2 2 2
- header: "📄 Quick Start"
 content: "
 Quick start and end-to-end tutorials.

 - [Getting started with prompt flow](how-to-guides/quick-start.md)

 - [E2E development tutorial: chat with PDF] (https://github.com/microsoft/promptflow/blob/main/examples/tutorials/e2e-development)

 - Find more: [tutorials & samples](tutorials/index.md)

 "

- header: "📄 How-to Guides"
 content: "
 Articles guide user to complete a specific task in prompt flow.

 - [Develop a flow](how-to-guides/develop-a-flow/index.md)

 - [Initialize and test a flow](how-to-guides/init-and-test-a-flow.md)

 - [Run and evaluate a flow](how-to-guides/run-and-evaluate-a-flow/index.md)

 - [Tune prompts using variants](how-to-guides/tune-prompts-with-variants.md)

 - [Develop custom tool](how-to-guides/develop-a-tool/create-and-use-tool-package.md)

 - [Deploy a flow](how-to-guides/deploy-a-flow/index.md)

 - [Process image in flow](how-to-guides/process-image-in-flow.md)
 "
```

```

:grid-columns: 1 2 2 2
- header: "□ Concepts"
 content: "
 Introduction of key concepts of prompt flow.

 - [Flows] (concepts/concept-flows.md)

 - [Tools] (concepts/concept-tools.md)

 - [Connections] (concepts/concept-connections.md)

 - [Design principles] (concepts/design-principles.md)

 "

- header: "□ Reference"
 content: "
 Reference provides technical information about prompt flow API.

 - Command line Interface reference: [pf] (reference/pf-command-reference.md)

 - Python library reference: [promptflow] (reference/python-library-reference/promptflow.md)

 - Tool reference: [LLM Tool] (reference/tools-reference/llm-tool.md), [Python Tool] (reference/tools-reference/python-tool.md), [P
 "

```

```

:hidden:
:maxdepth: 1
how-to-guides/quick-start

```

```

:hidden:
:maxdepth: 1
how-to-guides/index

```

```

:hidden:
:maxdepth: 1
tutorials/index

```

```

:hidden:
:maxdepth: 2
concepts/index

```

```

:hidden:
:maxdepth: 1
reference/index

```

```

:hidden:
:maxdepth: 1
cloud/index

```

```

:hidden:
:maxdepth: 1
integrations/index

```

filepath: promptflow/docs/README.md content: # Promptflow documentation contribute guidelines

This folder contains the source code for [prompt flow documentation site \(https://microsoft.github.io/promptflow/\)](https://microsoft.github.io/promptflow/).

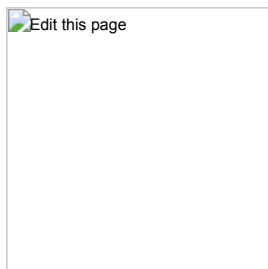
This readme file will not be included in above doc site. It keeps a guide for promptflow documentation contributors.

## Content

Below is a table of important doc pages. | Category | Article | |-----|-----| |Quick start|[Getting started with prompt flow \(./how-to-guides/quick-start.md\)](#) |[Concepts](#)|[Flows](#)  
[\(./concepts/concept-flows.md\)](#)  
[Tools \(./concepts/concept-tools.md\)](#)  
[Connections \(./concepts/concept-connections.md\)](#)  
[Variants \(./concepts/concept-variants.md\)](#)

| [How-to guides](#) | [How to initialize and test a flow \(./how-to-guides/init-and-test-a-flow.md\)](#)  
[How to run and evaluate a flow \(./how-to-guides/run-and-evaluate-a-flow/index.md\)](#)  
[How to tune prompts using variants \(./how-to-guides/tune-prompts-with-variants.md\)](#)  
[How to deploy a flow \(./how-to-guides/deploy-a-flow/index.md\)](#)  
[How to create and use your own tool package \(./how-to-guides/develop-a-tool/create-and-use-tool-package.md\)](#) | [Tools reference](#) | [LLM tool \(./reference/tools-reference/llm-tool.md\)](#)  
[Prompt tool \(./reference/tools-reference/prompt-tool.md\)](#)  
[Python tool \(./reference/tools-reference/python-tool.md\)](#)  
[Embedding tool \(./reference/tools-reference/embedding\\_tool.md\)](#)  
[SERP API tool \(./reference/tools-reference/serp-api-tool.md\)](#) ||

## Writing tips



0. Reach the doc source repository by clicking `Edit this page` on any page.
1. Please use `:::` for experimental feature or notes, and admonition with dropdown for the Limitation Part.
2. Please use `:::` to group your sdk/cli example, and put the cli at first. Use `:sync:` to sync multiple tables .
3. If you are unclear with the above lines, refer to [get started \(./how-to-guides/quick-start.md\)](#) to see the usage.
4. Add gif: Use [ScreenToGif \(https://www.screentogif.com/\)](https://www.screentogif.com/) to record your screen, edit and save as a gif.
5. Reach more element style at [Sphinx Design Components \(https://pydata-sphinx-theme.readthedocs.io/en/latest/user\\_guide/web-components.html\)](https://pydata-sphinx-theme.readthedocs.io/en/latest/user_guide/web-components.html).

## Preview your changes

**Local build:** We suggest using local build at the beginning, as it's fast and efficiency.

Please refer to [How to build doc site locally \(./dev/documentation\\_guidelines.md#how-to-build-doc-site-locally\)](#).

## FAQ

### Adding image in doc

Please use markdown syntax `![img desc](img link)` to reference image, because the relative path of image will be changed after sphinx build, and image placed in html tags can not be referenced when build.

### Draw flow chart in doc

We recommend using the mermaid, learn more from the [mermaid syntax doc \(https://mermaid-js.github.io/mermaid/#/flowchart?id=flowcharts-basic-syntax\)](https://mermaid-js.github.io/mermaid/#/flowchart?id=flowcharts-basic-syntax)

- Recommend to install [vscode extension \(https://marketplace.visualstudio.com/items?itemName=bierner.markdown-mermaid\)](https://marketplace.visualstudio.com/items?itemName=bierner.markdown-mermaid) to preview graph in vscode.

## Reference

- [md-and-rst \(https://coderefinery.github.io/sphinx-lesson/md-and-rst/\)](https://coderefinery.github.io/sphinx-lesson/md-and-rst/)
- [sphinx-quickstart \(https://www.sphinx-doc.org/en/master/usage/quickstart.html\)](https://www.sphinx-doc.org/en/master/usage/quickstart.html)

filepath: `promptflow/docs/dev/replay-e2e-test.md` content: `# Replay end-to-end tests`

- This document introduces replay tests for those located in `sdk cli azure test (./src/promptflow/tests/sdk cli azure test/e2etests/)` and `sdk cli test (./src/promptflow/tests/sdk cli test/e2etests/)`.
- The primary purpose of replay tests is to avoid the need for credentials, Azure workspaces, OpenAI tokens, and to directly test prompt flow behavior.
- Although there are different techniques behind recording/replaying, there are some common steps to run the tests in replay mode.
- The key handle of replay tests is the environment variable `PROMPT_FLOW_TEST_MODE`.

## How to run tests in replay mode

After cloning the full repo and setting up the proper test environment following [dev\\_setup.md \(./dev\\_setup.md\)](#), run the following command in the root directory of the repo:

1. If you have changed/affected tests in **sdk\_cli\_test** : Copy or rename the file [dev-connections.json.example \(./../src/promptflow/dev-connections.json.example\)](#) to `connections.json` in the same folder.
2. In your Python environment, set the environment variable `PROMPT_FLOW_TEST_MODE` to 'replay' and run the test(s).

These tests should work properly without any real connection settings.

## Test modes

There are 3 representative values of the environment variable `PROMPT_FLOW_TEST_MODE`

- **live**: Tests run against the real backend, which is the way traditional end-to-end tests do.
- **record**: Tests run against the real backend, and network traffic will be sanitized (filter sensitive and unnecessary requests/responses) and recorded to local files (recordings).
- **replay**: There is no real network traffic between SDK/CLI and the backend, tests run against local recordings.

## Update test recordings

To record a test, don't forget to clone the full repo and set up the proper test environment following [dev\\_setup.md \(./dev\\_setup.md\)](#):

1. Prepare some data.
  - If you have changed/affected tests in **sdk\_cli\_test**: Copy or rename the file [dev-connections.json.example \(./../src/promptflow/dev-connections.json.example\)](#) to `connections.json` in the same folder.
  - If you have changed/affected tests in **sdk\_cli\_azure\_test**: prepare your Azure ML workspace, make sure your Azure CLI logged in, and set the environment variable `PROMPT_FLOW_SUBSCRIPTION_ID`, `PROMPT_FLOW_RESOURCE_GROUP_NAME`, `PROMPT_FLOW_WORKSPACE_NAME` and `PROMPT_FLOW_RUNTIME_NAME` (if needed) pointing to your workspace.
2. Record the test.
  - Specify the environment variable `PROMPT_FLOW_TEST_MODE` to 'record'. If you have a `.env` file, we recommend specifying it there. Here is an example [.env file \(./../src/promptflow/.env.example\)](#). Then, just run the test that you want to record.
3. Once the test completed.
  - If you have changed/affected tests in **sdk\_cli\_azure\_test**: There should be one new YAML file located in `src/promptflow/tests/test_configs/recordings/`, containing the network traffic of the test.
  - If you have changed/affected tests in **sdk\_cli\_test**: There may be changes in the folder `src/promptflow/tests/test_configs/node_recordings/`. Don't worry if there are no changes, because similar LLM calls may have been recorded before.

## Techniques behind replay test

### Sdk\_cli\_azure\_test

End-to-end tests for pfazure aim to test the behavior of the PromptFlow SDK/CLI as it interacts with the service. This process can be time-consuming, error-prone, and require credentials (which are unavailable to pull requests from forked repositories); all of these go against our intention for a smooth development experience.

Therefore, we introduce replay tests, which leverage [VCR.py \(https://pypi.org/project/vcrpy/\)](#) to record all required network traffic to local files and replay during tests. In this way, we avoid the need for credentials, speed up, and stabilize the test process.

### Sdk\_cli\_test

`sdk_cli_test` often doesn't use a real backend. It will directly invokes LLM calls from localhost. Thus the key target of replay tests is to avoid the need for OpenAI tokens. If you have OpenAI / Azure OpenAI tokens yourself, you can try recording the tests. Record Storage will not record your own LLM connection, but only the inputs and outputs of the LLM calls.

There are also limitations. Currently, recorded calls are:

- AzureOpenAI calls
- OpenAI calls
- tool name "fetch\_text\_content\_from\_url" and tool name "my\_python\_tool"

filepath: `promptflow/docs/dev/dev_setup.md` content: `# Dev Setup`

## Set up process

- First create a new [conda \(https://conda.io/projects/conda/en/latest/user-guide/getting-started.html\)](#) environment. Please specify python version as 3.9. `conda create -n <env_name> python=3.9`.
- Activate the env you created.
- Set environment variable `PYTHONPATH` in your new conda environment. `conda env config vars set PYTHONPATH=<path-to-src>\promptflow`. Once you have set the environment variable, you have to reactivate your environment. `conda activate <env_name>`.
- In root folder, run `python scripts/building/dev_setup.py --promptflow-extra-deps azure` to install the package and dependencies.

# How to run tests

## Set up your secrets

`dev-connections.json.example` is a template about connections provided in `src/promptflow`. You can follow these steps to refer to this template to configure your connection for the test cases:

1. `cd ./src/promptflow`
2. Run the command `cp dev-connections.json.example connections.json`;
3. Replace the values in the json file with your connection info;
4. Set the environment `PROMPTFLOW_CONNECTIONS='connections.json'`;

After above setup process is finished. You can use `pytest` command to run test, for example in root folder you can:

## Run tests via command

- Run all tests under a folder: `pytest src/promptflow/tests -v`
- Run a single test: `pytest src/promptflow/tests/promptflow_test/e2etests/test_executor.py::TestExecutor::test_executor_basic_flow -v`

## Run tests in VSCode

1. Set up your python interpreter

- Open the Command Palette (Ctrl+Shift+P) and select `Python: Select Interpreter`.



- Select existing conda env which you created previously.



2. Set up your test framework and directory

- Open the Command Palette (Ctrl+Shift+P) and select `Python: Configure Tests`.



- Select `pytest` as test framework.



- Select `Root` directory as test directory.



3. Exclude specific test folders.

You can exclude specific test folders if you don't have some extra dependency to avoid VS Code's test discovery fail. For example, if you don't have azure dependency, you can `exclude sdk_cli_azure_test`. Open `.vscode/settings.json`, write `--ignore=src/promptflow/tests/sdk_cli_azure_test` to `"python.testing.pytestArgs"`.



4. Click the `Run Test` button on the left



## Run tests in pycharm

1. Set up your pycharm python interpreter



2. Select existing conda env which you created previously





3. Run test, right-click the test name to run, or click the green arrow button on the left.



## Record and replay tests

Please refer to [Replay End-to-End Tests \(/replay-e2e-test.md\)](#) to learn how to record and replay tests.

## How to write docstring.

A clear and consistent API documentation is crucial for the usability and maintainability of our codebase. Please refer to [API Documentation Guidelines \(/documentation\\_guidelines.md\)](#) to learn how to write docstring when developing the project.

## How to write tests

- Put all test data/configs under `src/promptflow/tests/test_configs`.
- Write unit tests:
  - Flow run: `src/promptflow/tests/sdk_cli_test/unittest/`
  - Flow run in azure: `src/promptflow/tests/sdk_cli_azure_test/unittest/`
- Write e2e tests:
  - Flow run: `src/promptflow/tests/sdk_cli_test/e2etests/`
  - Flow run in azure: `src/promptflow/tests/sdk_cli_azure_test/e2etests/`
- Test file name and the test case name all start with `test_`.
- A basic test example, see [test\\_connection.py \(/src/promptflow/tests/sdk\\_cli\\_test/e2etests/test\\_connection.py\)](#).

## Test structure

Currently all tests are under `src/promptflow/tests/` folder:

- tests/
  - promptflow/
    - sdk\_cli\_test/
      - e2etests/
      - unittests/
    - sdk\_cli\_azure\_test/
      - e2etests/
      - unittests/
  - test\_configs/
    - connections/
    - datas/
    - flows/
    - runs/
    - wrong\_flows/
    - wrong\_tools/

When you want to add tests for a new feature, you can add new test file let's say a e2e test file `test_construction.py` under `tests/promptflow/**/e2etests/`.

Once the project gets more complicated or anytime you find it necessary to add new test folder and test configs for a specific feature, feel free to split the `promptflow` to more folders, for example:

- tests/
  - (Test folder name)/
    - e2etests/
      - test\_xxx.py
    - unittests/
      - test\_xxx.py
  - test\_configs/
    - (Data or config folder name)/

filepath: `promptflow/docs/dev/documentation_guidelines.md` content: `# Promptflow Reference Documentation Guide`

## Overview

This guide describes how to author Python docstrings for promptflow public interfaces. See our doc site at [Promptflow API reference documentation](https://microsoft.github.io/promptflow/reference/python-library-reference/promptflow.html) (<https://microsoft.github.io/promptflow/reference/python-library-reference/promptflow.html>).

## Principles

- **Coverage:** Every public object must have a docstring. For private objects, docstrings are encouraged but not required.
- **Style:** All docstrings should be written in [Sphinx style](https://sphinx-rtd-tutorial.readthedocs.io/en/latest/docstrings.html#the-sphinx-docstring-format) (<https://sphinx-rtd-tutorial.readthedocs.io/en/latest/docstrings.html#the-sphinx-docstring-format>) noting all types and if any exceptions are raised.
- **Relevance:** The documentation is up-to-date and relevant to the current version of the product.
- **Clarity:** The documentation is written in clear, concise language that is easy to understand.
- **Consistency:** The documentation has a consistent format and structure, making it easy to navigate and follow.

## How to write the docstring

First please read through [Sphinx style](https://sphinx-rtd-tutorial.readthedocs.io/en/latest/docstrings.html#the-sphinx-docstring-format) (<https://sphinx-rtd-tutorial.readthedocs.io/en/latest/docstrings.html#the-sphinx-docstring-format>) to have a basic understanding of sphinx style docstring.

### Write class docstring

Let's start with a class example:

```

from typing import Dict, Optional, Union
from promptflow import PFClient

class MyClass:
 """One-line summary of the class.

 More detailed explanation of the class. May include below notes, admonitions, code blocks.

 .. note::

 Here are some notes to show, with a nested python code block:

 .. code-block:: python

 from promptflow import MyClass, PFClient
 obj = MyClass(PFClient())

 .. admonition:: [Title of the admonition]

 Here are some admonitions to show.

 :param client: Description of the client.
 :type client: ~promptflow.PFClient
 :param param_int: Description of the parameter.
 :type param_int: Optional[int]
 :param param_str: Description of the parameter.
 :type param_str: Optional[str]
 :param param_dict: Description of the parameter.
 :type param_dict: Optional[Dict[str, str]]
 """
 def __init__(
 client: PFClient,
 param_int: Optional[int] = None,
 param_str: Optional[str] = None,
 param_dict: Optional[Dict[str, str]] = None,
) -> None:
 """No docstring for __init__, it should be written in class definition above."""
 ...

```

#### Notes:

- One-line summary is required. It should be clear and concise.
- Detailed explanation is encouraged but not required. This part may or may not include notes, admonitions and code blocks.
  - The format like `.. note::` is called `directive`. Directives are a mechanism to extend the content of [reStructuredText \(https://docutils.sourceforge.io/rst.html\)](https://docutils.sourceforge.io/rst.html). Every directive declares a block of content with specific role. Start a new line with `.. directive_name::` to use the directive.
  - The directives used in the sample(`note/admonition/code-block`) should be enough for basic usage of docstring in our project. But you are welcomed to explore more [Directives \(https://www.sphinx-doc.org/en/master/usage/restructuredtext/directives.html#specific-admonitions\)](https://www.sphinx-doc.org/en/master/usage/restructuredtext/directives.html#specific-admonitions).
- Parameter description and type is required.
  - A pair of `:param [ParamName]:` and `:type [ParamName]:` is required.
  - If the type is a promptflow public class, use the full path to the class and prepend it with a `~`. This will create a link when the documentation is rendered on the doc site that will take the user to the class reference documentation for more information.

```

:param client: Description of the client.
:type client: ~promptflow.PFClient

```

- Use Union/Optional when appropriate in function declaration. And use the same annotation after `:type [ParamName]:`

```

:type param_int: Optional[int]

```

4. For classes, include docstring in definition only. If you include a docstring in both the class definition and the constructor (init method) docstrings, it will show up twice in the reference docs.
5. Constructors (def `__init__`) should return `None`, per [PEP 484 standards \(https://peps.python.org/pep-0484/#the-meaning-of-annotations\)](https://peps.python.org/pep-0484/#the-meaning-of-annotations).
6. To create a link for promptflow class on our doc site. `~promptflow.xxx.MyClass` alone only works after `:type [ParamName]` and `:rtype:`. If you want to achieve the same effect in docstring summary, you should use it with `:class::`

```
"""
An example to achieve link effect in summary for :class:`~promptflow.xxx.MyClass`
For function, use :meth:`~promptflow.xxx.my_func`
"""
```

7. There are some tricks to highlight the content in your docstring:
  - Single backticks (```): Single backticks are used to represent inline code elements within the text. It is typically used to highlight function names, variable names, or any other code elements within the documentation.
  - Double backticks (````): Double backticks are typically used to highlight a literal value.
8. If there are any class level constants you don't want to expose to doc site, make sure to add `_` in front of the constant to hide it.

## Write function docstring

```
from typing import Optional

def my_method(param_int: Optional[int] = None) -> int:
 """One-line summary

 Detailed explanations.

 :param param_int: Description of the parameter.
 :type param_int: int
 :raises [ErrorType1]: [ErrorDescription1]
 :raises [ErrorType2]: [ErrorDescription2]
 :return: Description of the return value.
 :rtype: int
 """
 ...
```

In addition to class docstring notes:

1. Function docstring should include return values.
  - If return type is promptflow class, we should also use `~promptflow.xxx.[ClassName]`.
2. Function docstring should include exceptions that may be raised in this function.
  - If exception type is `PromptflowException`, use `~promptflow.xxx.[ExceptionName]`
  - If multiple exceptions are raised, just add new lines of `:raises`, see the example above.

## How to build doc site locally

You can build the documentation site locally to preview the final effect of your docstring on the rendered site. This will provide you with a clear understanding of how your docstring will appear on our site once your changes are merged into the main branch.

1. Setup your dev environment, see [dev\\_setup \(/dev\\_setup.md\)](#) for details. Sphinx will load all source code to process docstring.
  - Skip this step if you just want to build the doc site without reference doc, but do remove `-WithReferenceDoc` from the command in step 3.
2. Install `langchain` package since it is used in our code but not covered in `dev_setup`.
3. Open a powershell, activate the conda env and navigate to `<repo-root>/scripts/docs`, run `doc_generation.ps1`:

```
cd scripts/docs
.\doc_generation.ps1 -WithReferenceDoc -WarningAsError
```

- For the first time you execute this command, it will take some time to install sphinx dependencies. After the initial installation, next time you can add param `-SkipInstall` to above command to save some time for dependency check.
4. Check warnings/errors in the build log, fix them if any, then build again.
  5. Open `scripts/docs/_build/index.html` to preview the local doc site.

# Additional comments

- **Utilities:** The [autoDocstring](https://marketplace.visualstudio.com/items?itemName=njpwerner.autodocstring) (<https://marketplace.visualstudio.com/items?itemName=njpwerner.autodocstring>) VSCode extension or GitHub Copilot can help autocomplete in this style for you.
- **Advanced principles**
  - Accuracy: The documentation accurately reflects the features and functionality of the product.
  - Completeness: The documentation covers all relevant features and functionality of the product.
  - Demonstration: Every docstring should include an up-to-date code snippet that demonstrates how to use the product effectively.

## References

- [AzureML v2 Reference Documentation Guide](https://github.com/Azure/azure-sdk-for-python/blob/main/sdk/ml/azure-ai/ml/documentation_guidelines.md) ([https://github.com/Azure/azure-sdk-for-python/blob/main/sdk/ml/azure-ai/ml/documentation\\_guidelines.md](https://github.com/Azure/azure-sdk-for-python/blob/main/sdk/ml/azure-ai/ml/documentation_guidelines.md))
- [Azure SDK for Python documentation guidelines](https://azure.github.io/azure-sdk/python_documentation.html#docstrings) ([https://azure.github.io/azure-sdk/python\\_documentation.html#docstrings](https://azure.github.io/azure-sdk/python_documentation.html#docstrings))
- [How to document a Python API](https://review.learn.microsoft.com/en-us/help/onboard/admin/reference/python/documenting-api?branch=main) (<https://review.learn.microsoft.com/en-us/help/onboard/admin/reference/python/documenting-api?branch=main>)

filepath: promptflow/docs/integrations/index.md content: # Integrations

The Integrations section contains documentation on custom extensions created by the community that expand prompt flow's capabilities. These include tools that enrich flows, as well as tutorials on innovative ways to use prompt flow.

```
:maxdepth: 1

tools/index
llms/index
```

filepath: promptflow/docs/integrations/llms/index.md content: # Alternative LLMs

This section provides tutorials on incorporating alternative large language models into prompt flow.

```
:maxdepth: 1
:hidden:
```

filepath: promptflow/docs/integrations/tools/index.md content: # Custom Tools

This section contains documentation for custom tools created by the community to extend Prompt flow's capabilities for specific use cases. These tools are developed following the guide on [Creating and Using Tool Packages](https://github.com/Azure/azure-sdk-for-python/blob/main/sdk/ml/azure-ai/ml/documentation_guidelines.md) ([./../how-to-guides/develop-a-tool/create-and-use-tool-package.md](https://github.com/Azure/azure-sdk-for-python/blob/main/sdk/ml/azure-ai/ml/documentation_guidelines.md)). They are not officially maintained or endorsed by the Prompt flow team. For questions or issues when using a tool, please use the support contact link in the table below.

## Tool Package Index

The table below provides an index of custom tool packages. The columns contain:

- **Package Name:** The name of the tool package. Links to the package documentation.
- **Description:** A short summary of what the tool package does.
- **Owner:** The creator/maintainer of the tool package.
- **Support Contact:** Link to contact for support and reporting new issues.

Package Name	Description	Owner	Support Contact
promptflow-azure-ai-language	Collection of Azure AI Language Prompt flow tools.	Sean Murray	taincidents@microsoft.com

```
:maxdepth: 1
:hidden:

azure-ai-language-tool
```

filepath: promptflow/docs/integrations/tools/azure-ai-language-tool.md content: # Azure AI Language Azure AI Language enables users with task-oriented and optimized pre-trained language models to effectively understand documents and conversations. This Prompt flow tool is a wrapper for various Azure AI Language APIs. The current list of supported capabilities is as follows:

Name	Description
Abstractive Summarization	Generate abstractive summaries from documents.
Extractive Summarization	Extract summaries from documents.
Conversation Summarization	Summarize conversations.

Name	Description
Entity Recognition	Recognize and categorize entities in documents.
Key Phrase Extraction	Extract key phrases from documents.
Language Detection	Detect the language of documents.
PII Entity Recognition	Recognize and redact PII entities in documents.
Sentiment Analysis	Analyze the sentiment of documents.
Conversational Language Understanding	Predict intents and entities from user's utterances.
Translator	Translate documents.

## Requirements

- For AzureML users: follow this [wiki \(https://learn.microsoft.com/en-us/azure/machine-learning/prompt-flow/how-to-custom-tool-package-creation-and-usage?view=azureml-api-2#prepare-runtime\)](https://learn.microsoft.com/en-us/azure/machine-learning/prompt-flow/how-to-custom-tool-package-creation-and-usage?view=azureml-api-2#prepare-runtime), starting from `Prepare runtime`. Note that the PyPi package name is `promptflow-azure-ai-language`.
- For local users:

```
pip install promptflow-azure-ai-language
```

## Prerequisites

The tool calls APIs from Azure AI Language. To use it, you must create a connection to an [Azure AI Language resource \(https://learn.microsoft.com/en-us/azure/ai-services/language-service/\)](https://learn.microsoft.com/en-us/azure/ai-services/language-service/). Create a Language resource first, if necessary.

- In Prompt flow, add a new `CustomConnection`.
  - Under the `secrets` field, specify the resource's API key: `api_key: <Azure AI Language Resource api key>`
  - Under the `configs` field, specify the resource's endpoint: `endpoint: <Azure AI Language Resource endpoint>`

To use the `Translator` tool, you must set up an additional connection to an [Azure AI Translator resource \(https://azure.microsoft.com/en-us/products/ai-services/ai-translator\)](https://azure.microsoft.com/en-us/products/ai-services/ai-translator). [Create a Translator resource \(https://learn.microsoft.com/en-us/azure/ai-services/translator/create-translator-resource\)](https://learn.microsoft.com/en-us/azure/ai-services/translator/create-translator-resource) first, if necessary.

- In Prompt flow, add a new `CustomConnection`.
  - Under the `secrets` field, specify the resource's API key: `api_key: <Azure AI Translator Resource api key>`
  - Under the `configs` field, specify the resource's endpoint: `endpoint: <Azure AI Translator Resource endpoint>`
  - If your Translator Resource is regional and non-global, specify its region under `configs` as well: `region: <Azure AI Translator Resource region>`

## Inputs

The tool accepts the following inputs:

- Abstractive Summarization:** | Name | Type | Description | Required | |-----|-----|-----|-----| | connection | CustomConnection | The created connection to an Azure AI Language resource. | Yes | | language | string | The ISO 639-1 code for the language of the input. | Yes | | text | string | The input text. | Yes | | query | string | The query used to structure summarization. | Yes | | summary\_length | string (enum) | The desired summary length. Enum values are `short`, `medium`, and `long`. | No | | parse\_response | bool | Should the raw API json output be parsed. Default value is `False`. | No |
- Extractive Summarization:** | Name | Type | Description | Required | |-----|-----|-----|-----| | connection | CustomConnection | The created connection to an Azure AI Language resource. | Yes | | language | string | The ISO 639-1 code for the language of the input. | Yes | | text | string | The input text. | Yes | | query | string | The query used to structure summarization. | Yes | | sentence\_count | int | The desired number of output summary sentences. Default value is 3. | No | | sort\_by | string (enum) | The sorting criteria for extractive summarization results. Enum values are `Offset` to sort results in order of appearance in the text and `Rank` to sort results in order of importance (i.e. rank score) according to model. Default value is `Offset`. | No | | parse\_response | bool | Should the raw API json output be parsed. Default value is `False`. | No |
- Conversation Summarization:** | Name | Type | Description | Required | |-----|-----|-----|-----| | connection | CustomConnection | The created connection to an Azure AI Language resource. | Yes | | language | string | The ISO 639-1 code for the language of the input. | Yes | | text | string | The input text. Text should be of the following form: `<speaker id>: <speaker text> \n <speaker id>: <speaker text> \n ...` | Yes | | modality | string (enum) | The modality of the input text. Enum values are `text` for input from a text source, and `transcript` for input from a transcript source. | Yes | | summary\_aspect | string (enum) | The desired summary "aspect" to obtain. Enum values are `chapterTitle` to obtain the chapter title of any conversation, `issue` to obtain the summary of issues in transcripts of web chats and service calls between customer-service agents and customers, `narrative` to obtain the generic summary of any conversation, `resolution` to obtain the summary of resolutions in transcripts of web chats and service calls between customer-service agents and customers, `recap` to obtain a general summary, and `follow-up tasks` to obtain a summary of follow-up or action items. | Yes | | parse\_response | bool | Should the raw API json output be parsed. Default value is `False`. | No |
- Entity Recognition:** | Name | Type | Description | Required | |-----|-----|-----|-----| | connection | CustomConnection | The created connection to an Azure AI Language resource. | Yes | | language | string | The ISO 639-1 code for the language of the input. | Yes | | text | string | The input text. | Yes | | parse\_response | bool | Should the raw API json output be parsed. Default value is `False`. | No |

- **Key Phrase Extraction:** | Name | Type | Description | Required | |-----|-----|-----|-----| | connection | CustomConnection | The created connection to an Azure AI Language resource. | Yes | | language | string | The ISO 639-1 code for the language of the input. | Yes | | text | string | The input text. | Yes | | parse\_response | bool | Should the raw API json output be parsed. Default value is `False`. | No |
- **Language Detection:** | Name | Type | Description | Required | |-----|-----|-----|-----| | connection | CustomConnection | The created connection to an Azure AI Language resource. | Yes | | text | string | The input text. | Yes | | parse\_response | bool | Should the raw API json output be parsed. Default value is `False`. | No |
- **PII Entity Recognition:** | Name | Type | Description | Required | |-----|-----|-----|-----| | connection | CustomConnection | The created connection to an Azure AI Language resource. | Yes | | language | string | The ISO 639-1 code for the language of the input. | Yes | | text | string | The input text. | Yes | | domain | string (enum) | The PII domain used for PII Entity Recognition. Enum values are `none` for no domain, or `phi` to indicate that entities in the Personal Health domain should be redacted. Default value is `none`. | No | | categories | list[string] | Describes the PII categories to return. Default value is `[]`. | No | | parse\_response | bool | Should the raw API json output be parsed. Default value is `False`. | No |
- **Sentiment Analysis:** | Name | Type | Description | Required | |-----|-----|-----|-----| | connection | CustomConnection | The created connection to an Azure AI Language resource. | Yes | | language | string | The ISO 639-1 code for the language of the input. | Yes | | text | string | The input text. | Yes | | opinion\_mining | bool | Should opinion mining be enabled. Default value is `False`. | No | | parse\_response | bool | Should the raw API json output be parsed. Default value is `False`. | No |
- **Conversational Language Understanding:** | Name | Type | Description | Required | |-----|-----|-----|-----| | connection | CustomConnection | The created connection to an Azure AI Language resource. | Yes | | language | string | The ISO 639-1 code for the language of the input. | Yes | | utterances | string | A single user utterance or a json array of user utterances. | Yes | | project\_name | string | The Conversational Language Understanding project to be called. | Yes | | deployment\_name | string | The Conversational Language Understanding project deployment to be called. | Yes | | parse\_response | bool | Should the raw API json output be parsed. Default value is `False`. | No |
- **Translator:** | Name | Type | Description | Required | |-----|-----|-----|-----| | connection | CustomConnection | The created connection to an Azure AI Translator resource. | Yes | | text | string | The input text. | Yes | | to | list[string] | The languages to translate the input text to. | Yes | | source\_language | string | The language of the input text. | No | | parse\_response | bool | Should the raw API json output be parsed. Default value is `False`. | No |

## Outputs

If the input parameter `parse_response` is set to `False` (default value), the raw API json output will be returned as a string. Refer to the [REST API reference \(https://learn.microsoft.com/en-us/rest/api/language/\)](https://learn.microsoft.com/en-us/rest/api/language/) for details on API output. For Conversational Language Understanding, the output will be a list of raw API json responses, one response for each user utterance in the input.

When `parse_response` is set to `True`, the tool will parse API output as follows:

Name	Type	Description
Abstractive Summarization	string	Abstractive summary.
Extractive Summarization	list[string]	Extracted summary sentence strings.
Conversation Summarization	string	Conversation summary based on <code>summary_aspect</code> .
Entity Recognition	dict[string, string]	Recognized entities, where keys are entity names and values are entity categories.
Key Phrase Extraction	list[string]	Extracted key phrases as strings.
Language Detection	string	Detected language's ISO 639-1 code.
PII Entity Recognition	string	Input <code>text</code> with PII entities redacted.
Sentiment Analysis	string	Analyzed sentiment: <code>positive</code> , <code>neutral</code> , or <code>negative</code> .
Conversational Language Understanding	list[dict[string, string]]	List of user utterances and associated intents.
Translator	dict[string, string]	Translated text, where keys are the translated languages and values are the translated texts.

filepath: `promptflow/docs/how-to-guides/execute-flow-as-a-function.md` content: `# Execute flow as a function`

This is an experimental feature, and may change at any time. Learn [more \(faq.md#stable-vs-experimental\)](#).

## Overview

Promptflow allows you to load a flow and use it as a function in your code. This feature is useful when building a service on top of a flow, reference [here \(https://github.com/microsoft/promptflow/tree/main/examples/tutorials/flow-deploy/create-service-with-flow\)](https://github.com/microsoft/promptflow/tree/main/examples/tutorials/flow-deploy/create-service-with-flow) for a simple example service with flow function consumption.

## Load an invoke the flow function

To use the flow-as-function feature, you first need to load a flow using the `load_flow` function. Then you can consume the flow object like a function by providing key-value arguments for it.

```
f = load_flow("../examples/flows/standard/web-classification/")
f(url="sample_url")
```

# Config the flow with context

You can overwrite some flow configs before flow function execution by setting `flow.context`.

## Load flow as a function with in-memory connection override

By providing a connection object to flow context, flow won't need to get connection in execution time, which can save time when for cases where flow function need to be called multiple times.

```
from promptflow.entities import AzureOpenAIConnection

connection_obj = AzureOpenAIConnection(
 name=conn_name,
 api_key=api_key,
 api_base=api_base,
 api_type="azure",
 api_version=api_version,
)

no need to create the connection object.
f.context = FlowContext(
 connections={"classify_with_llm": {"connection": connection_obj}}
)
```

## Local flow as a function with flow inputs override

By providing overrides, the original flow dag will be updated in execution time.

```
f.context = FlowContext(
 # node "fetch_text_content_from_url" will take inputs from the following command instead of from flow input
 overrides={"nodes.fetch_text_content_from_url.inputs.url": sample_url},
)
```

**Note**, the overrides are only doing YAML content replacement on original `flow.dag.yaml`. If the `flow.dag.yaml` become invalid after overrides, validation error will be raised when executing.

## Load flow as a function with streaming output

After set streaming in flow context, the flow function will return an iterator to stream the output.

```
f = load_flow(source="../../examples/flows/chat/basic-chat/")
f.context.streaming = True
result = f(
 chat_history=[
 {
 "inputs": {"chat_input": "Hi"},
 "outputs": {"chat_output": "Hello! How can I assist you today?"},
 }
],
 question="How are you?",
)

answer = ""
the result will be a generator, iterate it to get the result
for r in result["answer"]:
 answer += r
```

Reference our [sample \(https://github.com/microsoft/promptflow/blob/main/examples/tutorials/get-started/flow-as-function.ipynb\)](https://github.com/microsoft/promptflow/blob/main/examples/tutorials/get-started/flow-as-function.ipynb) for usage.

# Flow with multiple overrides



**Note:** the flow context configs may affect each other in some cases. For example, using `connection` & `overrides` to override same node. The behavior is undefined for those scenarios. Please avoid such usage.

```
overriding `classify_with_llm`'s connection and inputs in the same time will lead to undefined behavior.
f.context = FlowContext(
 connections={"classify_with_llm": {"connection": connection_obj}},
 overrides={"nodes.classify_with_llm.inputs.url": sample_url}
)
```

## Next steps

Learn more about:

- [Flow as a function sample \(https://github.com/microsoft/promptflow/blob/main/examples/tutorials/get-started/flow-as-function.ipynb\)](https://github.com/microsoft/promptflow/blob/main/examples/tutorials/get-started/flow-as-function.ipynb).
- [Deploy a flow \(.deploy-a-flow/index.md\)](#).

filepath: promptflow/docs/how-to-guides/quick-start.md content: # Quick Start

This guide will walk you through the first step using of prompt flow code-first experience.

**Prerequisite** - To make the most of this tutorial, you'll need:

- Know how to program with Python :)
- A basic understanding of Machine Learning can be beneficial, but it's not mandatory.

**Learning Objectives** - Upon completing this tutorial, you should learn how to:

- Setup your python environment to run prompt flow
- Clone a sample flow & understand what's a flow
- Understand how to edit the flow using visual editor or yaml
- Test the flow using your favorite experience: CLI, SDK or VS Code Extension.

## Set up your dev environment

1. A python environment with version `python=3.9` or higher version like 3.10. It's recommended to use python environment manager [miniconda \(https://docs.conda.io/en/latest/miniconda.html\)](https://docs.conda.io/en/latest/miniconda.html). After you have installed miniconda, run below commands to create a python environment:

```
conda create --name pf python=3.9
conda activate pf
```

2. Install `promptflow` and `promptflow-tools`.

```
pip install promptflow promptflow-tools
```

3. Check the installation.

```
should print promptflow version, e.g. "0.1.0b3"
pf -v
```

## Understand what's a flow

A flow, represented as a YAML file, is a DAG of functions, which is connected via input/output dependencies, and executed based on the topology by prompt flow executor. See [Flows \(.concepts/concept-flows.md\)](#) for more details.

### Get the flow sample

Clone the sample repo and check flows in folder [examples/flows \(https://github.com/microsoft/promptflow/tree/main/examples/flows\)](https://github.com/microsoft/promptflow/tree/main/examples/flows).

```
git clone https://github.com/microsoft/promptflow.git
```

### Understand flow directory

The sample used in this tutorial is the [web-classification \(https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification\)](https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification) flow, which categorizes URLs into several predefined classes. Classification is a traditional machine learning task, and this sample illustrates how to perform classification using GPT and prompts.

```
cd promptflow/examples/flows/standard/web-classification
```

A flow directory is a directory that contains all contents of a flow. Structure of flow folder:

- **flow.dag.yaml**: The flow definition with inputs/outputs, nodes, tools and variants for authoring purpose.
- **.promptflow/flow.tools.json**: It contains tools meta referenced in flow.dag.yaml.
- **Source code files (.py, .jinja2)**: User managed, the code scripts referenced by tools.
- **requirements.txt**: Python package dependencies for this flow.



In order to run this specific flow, you need to install its requirements first.

```
pip install -r requirements.txt
```

## Understand the flow yaml

The entry file of a flow directory is **flow.dag.yaml** (<https://github.com/microsoft/promptflow/blob/main/examples/flows/standard/web-classification/flow.dag.yaml>) which describes the DAG (Directed Acyclic Graph) of a flow. Below is a sample of flow DAG:



This graph is rendered by VS Code extension which will be introduced in the next section.

## Using VS Code Extension to visualize the flow

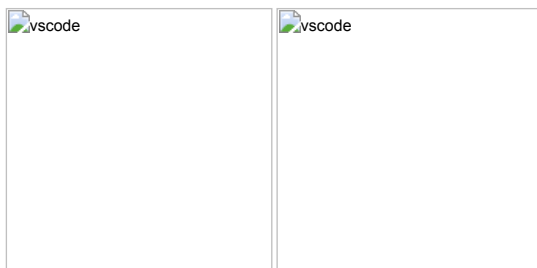
*Note: Prompt flow VS Code Extension is highly recommended for flow development and debugging.*

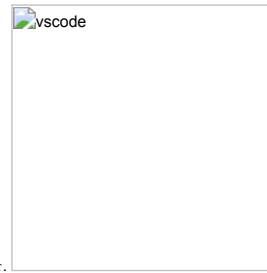
1. Prerequisites for VS Code extension.

- Install latest stable version of **VS Code** (<https://code.visualstudio.com/>).
- Install **VS Code Python extension** (<https://marketplace.visualstudio.com/items?itemName=ms-python.python>).

2. Install **Prompt flow for VS Code extension** (<https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow>)

3. Select python interpreter





4. Open dag in vscode. You can open the `flow.dag.yaml` as yaml file, or you can also open it in visual editor.

## Develop and test your flow

### How to edit the flow

To test your flow with varying input data, you have the option to modify the default input. If you are well-versed with the structure, you may also add or remove nodes to alter the flow's arrangement.

```
$schema: https://azuremlschemas.azureedge.net/promptflow/latest/Flow.schema.json
inputs:
 url:
 type: string
 # change the default value of input url here
 default: https://play.google.com/store/apps/details?id=com.twitter.android
...
```

See more details of this topic in [Develop a flow](#) ([./develop-a-flow/index.md](#)).

### Create necessary connections

If you are using WSL or other OS without default keyring storage backend, you may encounter `StoreConnectionEncryptionKeyError`, please refer to [FAQ](#) ([./faq.md#connection-creation-failed-with-storeconnectionencryptionkeyerror](#)) for the solutions.

The [connection](#) ([./concepts/concept-connections.md](#)) helps securely store and manage secret keys or other sensitive credentials required for interacting with LLM and other external tools for example Azure Content Safety.

The sample flow [web-classification](#) (<https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification>) uses connection `open_ai_connection` inside, e.g. `classify_with_llm` node needs to talk to llm using the connection.

We need to set up the connection if we haven't added it before. Once created, the connection will be stored in local db and can be used in any flow.

:sync: CLI

Firstly we need a connection yaml file `connection.yaml`:

If you are using Azure Open AI, prepare your resource follow with this [instruction](https://learn.microsoft.com/en-us/azure/cognitive-services/openai/how-to/create-resource?pivots=web-portal) (<https://learn.microsoft.com/en-us/azure/cognitive-services/openai/how-to/create-resource?pivots=web-portal>) and get your `api_key` if you don't have one.

```
$schema: https://azuremlschemas.azureedge.net/promptflow/latest/AzureOpenAIConnection.schema.json
name: open_ai_connection
type: azure_open_ai
api_key: <test_key>
api_base: <test_base>
api_type: azure
api_version: <test_version>
```

If you are using OpenAI, sign up account via [OpenAI website](https://openai.com/) (<https://openai.com/>), login and [find personal API key](https://platform.openai.com/account/api-keys) (<https://platform.openai.com/account/api-keys>), then use this yaml:

```
$schema: https://azuremlschemas.azureedge.net/promptflow/latest/OpenAIConnection.schema.json
name: open_ai_connection
type: open_ai
api_key: "<user-input>"
organization: "" # optional
```

Then we can use CLI command to create the connection.

```
pf connection create -f connection.yaml
```

More command details can be found in [CLI reference](#) ([./reference/pf-command-reference.md](#)).

:sync: SDK

In SDK, connections can be created and managed with `PFClient`.

```
from promptflow import PFClient
from promptflow.entities import AzureOpenAIConnection

PFClient can help manage your runs and connections.
pf = PFClient()

try:
 conn_name = "open_ai_connection"
 conn = pf.connections.get(name=conn_name)
 print("using existing connection")
except:
 connection = AzureOpenAIConnection(
 name=conn_name,
 api_key="<test_key>",
 api_base="<test_base>",
 api_type="azure",
 api_version="<test_version>",
)

 # use this if you have an existing OpenAI account
 # from promptflow.entities import OpenAIConnection
 # connection = OpenAIConnection(
 # name=conn_name,
 # api_key="<user-input>",
 #)

 conn = pf.connections.create_or_update(connection)
 print("successfully created connection")

print(conn)
```

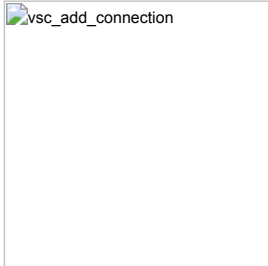
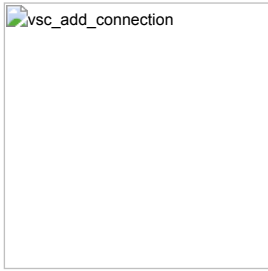
:sync: VS Code Extension

1. Click the promptflow icon to enter promptflow control panel



2. Create your connection.





Learn more on more actions like delete connection in: [Manage connections \(/manage-connections.md\)](#).

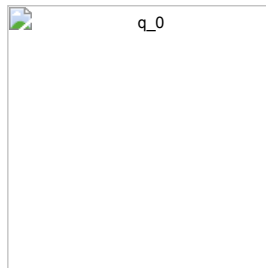
## Test the flow

Testing flow will NOT create a batch run record, therefore it's unable to use commands like `pf run show-details` to get the run information. If you want to persist the run record, see [Run and evaluate a flow \(/run-and-evaluate-a-flow/index.md\)](#).

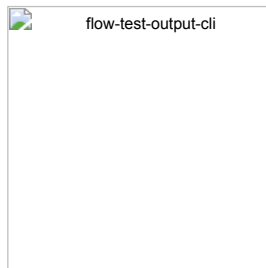
Assuming you are in working directory `promptflow/examples/flows/standard/`

:sync: CLI

Change the default input to the value you want to test.



```
pf flow test --flow web-classification # "web-classification" is the directory name
```



:sync: SDK

The return value of `test` function is the flow/node outputs.

```

from promptflow import PFClient

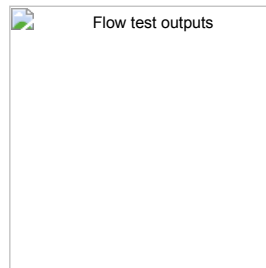
pf = PFClient()

flow_path = "web-classification" # "web-classification" is the directory name

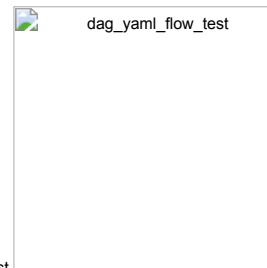
Test flow
flow_inputs = {"url": "https://www.youtube.com/watch?v=o5ZQyXaAv1g", "answer": "Channel", "evidence": "Url"} # The inputs of the flow
flow_result = pf.test(flow=flow_path, inputs=flow_inputs)
print(f"Flow outputs: {flow_result}")

Test node in the flow
node_name = "fetch_text_content_from_url" # The node name in the flow.
node_inputs = {"url": "https://www.youtube.com/watch?v=o5ZQyXaAv1g"} # The inputs of the node.
node_result = pf.test(flow=flow_path, inputs=node_inputs, node=node_name)
print(f"Node outputs: {node_result}")

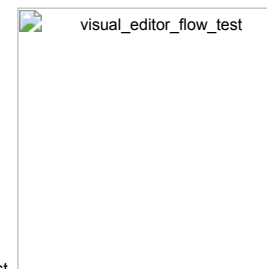
```



:sync: VS Code Extension



Use the code lens action on the top of the yaml editor to trigger flow test.



Click the run flow button on the top of the visual editor to trigger flow test.

See more details of this topic in [Initialize and test a flow \(/init-and-test-a-flow.md\)](#).

## Next steps

Learn more on how to:

- [Develop a flow \(/develop-a-flow/index.md\)](#): details on how to develop a flow by writing a flow yaml from scratch.
- [Initialize and test a flow \(/init-and-test-a-flow.md\)](#): details on how develop a flow from scratch or existing code.
- [Add conditional control to a flow \(/add-conditional-control-to-a-flow.md\)](#): how to use activate config to add conditional control to a flow.
- [Run and evaluate a flow \(/run-and-evaluate-a-flow/index.md\)](#): run and evaluate the flow using multi line data file.
- [Deploy a flow \(/deploy-a-flow/index.md\)](#): how to deploy the flow as a web app.
- [Manage connections \(/manage-connections.md\)](#): how to manage the endpoints/secrets information to access external services including LLMs.
- [Prompt flow in Azure AI \(/cloud/azureai/quick-start/index.md\)](#): run and evaluate flow in Azure AI where you can collaborate with team better.

And you can also check our [examples \(https://github.com/microsoft/promptflow/tree/main/examples\)](https://github.com/microsoft/promptflow/tree/main/examples), especially:

- [Getting started with prompt flow \(https://github.com/microsoft/promptflow/blob/main/examples/tutorials/get-started/quickstart.ipynb\)](https://github.com/microsoft/promptflow/blob/main/examples/tutorials/get-started/quickstart.ipynb): the notebook covering the python sdk experience for sample introduced in this doc.
- [Tutorial: Chat with PDF \(https://github.com/microsoft/promptflow/blob/main/examples/tutorials/e2e-development/chat-with-pdf.md\)](https://github.com/microsoft/promptflow/blob/main/examples/tutorials/e2e-development/chat-with-pdf.md): An end-to-end tutorial on how to build a high quality chat application with prompt flow, including flow development and evaluation with metrics.

filepath: promptflow/docs/how-to-guides/index.md content: # How-to Guides

Simple and short articles grouped by topics, each introduces a core feature of prompt flow and how you can use it to address your specific use cases.

```
:maxdepth: 1

develop-a-flow/index
init-and-test-a-flow
add-conditional-control-to-a-flow
run-and-evaluate-a-flow/index
tune-prompts-with-variants
execute-flow-as-a-function
deploy-a-flow/index
enable-streaming-mode
manage-connections
manage-runs
set-global-configs
develop-a-tool/index
process-image-in-flow
faq
```

filepath: promptflow/docs/how-to-guides/tune-prompts-with-variants.md content: # Tune prompts using variants

This is an experimental feature, and may change at any time. Learn [more \(faq.md#stable-vs-experimental\)](#).

To better understand this part, please read [Quick start \(./quick-start.md\)](#) and [Run and evaluate a flow \(./run-and-evaluate-a-flow/index.md\)](#) first.

## What is variant and why should we care

In order to help users tune the prompts in a more efficient way, we introduce [the concept of variants \(./../concepts/concept-variants.md\)](#), which can help you test the model's behavior under different conditions, such as different wording, formatting, context, temperature, or top-k, compare and find the best prompt and configuration that maximizes the model's accuracy, diversity, or coherence.

## Create a run with different variant node

In this example, we use the flow [web-classification \(https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification\)](https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification), its node `summarize_text_content` has two variants: `variant_0` and `variant_1`. The difference between them is the inputs parameters:

```

...
nodes:
- name: summarize_text_content
 use_variants: true
...
node_variants:
 summarize_text_content:
 default_variant_id: variant_0
 variants:
 variant_0:
 node:
 type: llm
 source:
 type: code
 path: summarize_text_content.jinja2
 inputs:
 deployment_name: text-davinci-003
 max_tokens: '128'
 temperature: '0.2'
 text: ${fetch_text_content_from_url.output}
 provider: AzureOpenAI
 connection: open_ai_connection
 api: completion
 module: promptflow.tools.aiai
 variant_1:
 node:
 type: llm
 source:
 type: code
 path: summarize_text_content__variant_1.jinja2
 inputs:
 deployment_name: text-davinci-003
 max_tokens: '256'
 temperature: '0.3'
 text: ${fetch_text_content_from_url.output}
 provider: AzureOpenAI
 connection: open_ai_connection
 api: completion
 module: promptflow.tools.aiai

```

You can check the whole flow definition in [flow.dag.yaml](https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification/flow.dag.yaml) (<https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification/flow.dag.yaml>).

Now we will create a variant run which uses node `summarize_text_content`'s variant `variant_1`. Assuming you are in working directory `<path-to-the-sample-repo>/examples/flows/standard`

:sync: CLI

Note we pass `--variant` to specify which variant of the node should be running.

```
pf run create --flow web-classification --data web-classification/data.jsonl --variant '${summarize_text_content.variant_1}' --column
```

:sync: SDK



```

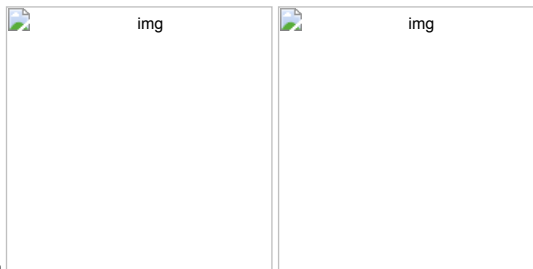
from promptflow import PFClient

pf = PFClient() # get a promptflow client
flow = "web-classification"
data= "web-classification/data.jsonl"

use the variant1 of the summarize_text_content node.
variant_run = pf.run(
 flow=flow,
 data=data,
 variant="${summarize_text_content.variant_1}", # use variant 1.
 column_mapping={"url": "${data.url}"},
)

pf.stream(variant_run)

```



:sync: VS Code Extension

After the variant run is created, you can evaluate the variant run with a evaluation flow, just like you evaluate a standard flow run.

## Next steps

Learn more about:

- [Run and evaluate a flow \(/run-and-evaluate-a-flow/index.md\)](#)
- [Deploy a flow \(/deploy-a-flow/index.md\)](#)
- [Prompt flow in Azure AI \(/cloud/azureai/quick-start/index.md\)](#)

filepath: promptflow/docs/how-to-guides/add-conditional-control-to-a-flow.md content: # Add conditional control to a flow

This is an experimental feature, and may change at any time. Learn [more \(tag.md#stable-vs-experimental\)](#).

In prompt flow, we support control logic by activate config, like if-else, switch. Activate config enables conditional execution of nodes within your flow, ensuring that specific actions are taken only when the specified conditions are met.

This guide will help you learn how to use activate config to add conditional control to your flow.

## Prerequisites

Please ensure that your promptflow version is greater than 0.1.0b5.

## Usage

Each node in your flow can have an associated activate config, specifying when it should execute and when it should bypass. If a node has activate config, it will only be executed when the activate condition is met. The configuration consists of two essential components:

- `activate.when`: The condition that triggers the execution of the node. It can be based on the outputs of a previous node, or the inputs of the flow.
- `activate.is`: The condition's value, which can be a constant value of string, boolean, integer, double.

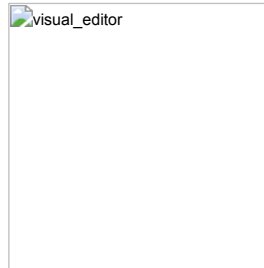
You can manually change the flow.dag.yaml in the flow folder or use the visual editor in VS Code Extension to add activate config to nodes in the flow.

:sync: YAML

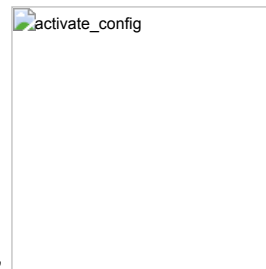
You can add activate config in the node section of flow yaml.

```
activate:
 when: ${node.output}
 is: true
```

:sync: VS Code Extension



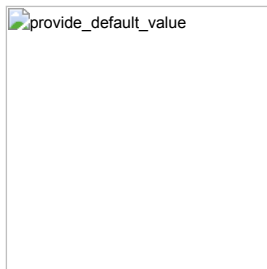
- Click `Visual editor` in the `flow.dag.yaml` to enter the flow interface.



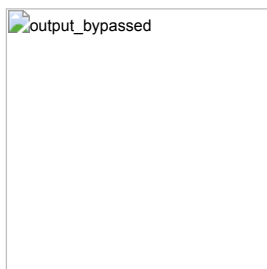
- Click on the `Activation config` section in the node you want to add and fill in the values for "when" and "is".

## Further details and important notes

1. If the node using the python tool has an input that references a node that may be bypassed, please provide a default value for this input whenever possible. If there is no default value for input, the output of the bypassed node will be set to None.



2. It is not recommended to directly connect nodes that might be bypassed to the flow's outputs. If it is connected, the output will be None and a warning will be raised.



3. In a conditional flow, if a node has activate config, we will always use this config to determine whether the node should be bypassed. If a node is bypassed, its status will be marked as "Bypassed", as shown in the figure below Show. There are three situations in which a node is bypassed.



(1) If a node has activate config and the value of `activate.when` is not equals to `activate.is`, it will be bypassed. If you want to force a node to always be executed, you can set the activate config to `when dummy is dummy` which always meets the activate condition.



(2) If a node has activate config and the node pointed to by `activate.when` is bypassed, it will be bypassed.



(3) If a node does not have activate config but depends on other nodes that have been bypassed, it will be bypassed.



## Example flow

Let's illustrate how to use activate config with practical examples.

- If-Else scenario: Learn how to develop a conditional flow for if-else scenarios: [View Example \(https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/conditional-flow-for-if-else\)](https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/conditional-flow-for-if-else).
- Switch scenario: Explore conditional flow for switch scenarios: [View Example \(https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/conditional-flow-for-switch\)](https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/conditional-flow-for-switch).

## Next steps

- [Run and evaluate a flow \(/run-and-evaluate-a-flow/index.md\)](#)

filepath: promptflow/docs/how-to-guides/process-image-in-flow.md content: # Process image in flow PromptFlow defines a contract to represent image data.

## Data class

`promptflow.contracts.multimedia.Image` Image class is a subclass of `bytes`, thus you can access the binary data by directly using the object. It has an extra attribute `source_url` to store the origin url of the image, which would be useful if you want to pass the url instead of content of image to APIs like GPT-4V model.

## Data type in flow input

Set the type of flow input to `image` and promptflow will treat it as an image.

## Reference image in prompt template

In prompt templates that support image (e.g. in OpenAI GPT-4V tool), using markdown syntax to denote that a template input is an image: `![[image]]({{test_image}})`. In this case, `test_image` will be substituted with base64 or `source_url` (if set) before sending to LLM model.

# Serialization/Deserialization

Promptflow uses a special dict to represent image. `{"data:image/<mime-type>;<representation>": "<value>"}`

- `<mime-type>` can be html standard [mime](https://developer.mozilla.org/en-US/docs/Web/HTTP/Basic MIME types/Common types) (<https://developer.mozilla.org/en-US/docs/Web/HTTP/Basic MIME types/Common types>) image types. Setting it to specific type can help previewing the image correctly, or it can be `*` for unknown type.

- `<representation>` is the image serialized representation, there are 3 supported types:

- url

It can point to a public accessible web url. E.g.

```
{"data:image/png:url": "https://developer.microsoft.com/_devcom/images/logo-ms-social.png"}
```

- base64

It can be the base64 encoding of the image. E.g.

```
{"data:image/png;base64":
"iVBORw0KGgoAAAANSUhEUgAAAGQAAABLAQMAAAC81rD0AAAABGdBTUEAALGPC/xhBQAAACBjSFJNAAB6JgAAgIQAAPoAAACA6AAAdTAAAOpgAAA6mAA"}
```

- path

It can reference an image file on local disk. Both absolute path and relative path are supported, but in the cases where the serialized image representation is stored in a file, relative to the containing folder of that file is recommended, as in the case of flow IO data. E.g.

```
{"data:image/png:path": "./my-image.png"}
```

Please note that `path` representation is not supported in Deployment scenario.

## Batch Input data

Batch input data containing image can be of 2 formats:

1. The same jsonl format of regular batch input, except that some column may be serialized image data or composite data type (dict/list) containing images. The serialized images can only be Url or Base64. E.g.

```
{"question": "How many colors are there in the image?", "input_image": {"data:image/png:url": "https://developer.microsoft.com/_devcom/images/logo-ms-social.png"}},
{"question": "What's this image about?", "input_image": {"data:image/png:url": "https://developer.microsoft.com/_devcom/images/logo-ms-social.png"}}
```

2. A folder containing a jsonl file under root path, which contains serialized image in File Reference format. The referenced file are stored in the folder and their relative path to the root path is used as path in the file reference. Here is a sample batch input, note that the name of `input.jsonl` is arbitrary as long as it's a jsonl file:

```
BatchInputFolder
|----input.jsonl
|----image1.png
|----image2.png
```

Content of `input.jsonl`

```
{"question": "How many colors are there in the image?", "input_image": {"data:image/png:path": "image1.png"}},
{"question": "What's this image about?", "input_image": {"data:image/png:path": "image2.png"}}
```

filepath: `promptflow/docs/how-to-guides/manage-connections.md` content: `# Manage connections`

This is an experimental feature, and may change at any time. Learn [more \(faq.md#stable-vs-experimental\)](#).

[Connection \(././concepts/concept-connections.md\)](#) helps securely store and manage secret keys or other sensitive credentials required for interacting with LLM (Large Language Models) and other external tools, for example, Azure Content Safety.

To use azureml workspace connection locally, refer to [this guide \(././how-to-guides/set-global-configs.md#connectionprovider\)](#).

## Connection types

There are multiple types of connections supported in promptflow, which can be simply categorized into **strong type connection** and **custom connection**. The strong type connection includes `AzureOpenAIConnection`, `OpenAIConnection`, etc. The custom connection is a generic connection type that can be used to store custom defined credentials.

We are going to use AzureOpenAIConnection as an example for strong type connection, and CustomConnection to show how to manage connections.

## Create a connection

If you are using WSL or other OS without default keyring storage backend, you may encounter `StoreConnectionEncryptionKeyError`, please refer to [FAQ \(/faq.md#connection-creation-failed-with-storeconnectionencryptionkeyerror\)](#) for the solutions.

:sync: CLI Each of the strong type connection has a corresponding yaml schema, the example below shows the AzureOpenAIConnection yaml:

```
$schema: https://azuremlschemas.azureedge.net/promptflow/latest/AzureOpenAIConnection.schema.json
name: azure_open_ai_connection
type: azure_open_ai
api_key: "<to-be-replaced>"
api_base: "https://<name>.openai.azure.com/"
api_type: "azure"
api_version: "2023-03-15-preview"
```

The custom connection yaml will have two dict fields for secrets and configs, the example below shows the CustomConnection yaml:

```
$schema: https://azuremlschemas.azureedge.net/promptflow/latest/CustomConnection.schema.json
name: custom_connection
type: custom
configs:
 endpoint: "<your-endpoint>"
 other_config: "other_value"
secrets: # required
 my_key: "<your-api-key>"
```

After preparing the yaml file, use the CLI command below to create them:

```
Override keys with --set to avoid yaml file changes
pf connection create -f <path-to-azure-open-ai-connection> --set api_key=<your-api-key>
Create the custom connection
pf connection create -f <path-to-custom-connection> --set configs.endpoint=<endpoint> secrets.my_key=<your-api-key>
```

The expected result is as follows if the connection created successfully.



:sync: SDK Using SDK, each connection type has a corresponding class to create a connection. The following code snippet shows how to import the required class and create the connection:

```

from promptflow import PFClient
from promptflow.entities import AzureOpenAIConnection, CustomConnection

Get a pf client to manage connections
pf = PFClient()

Initialize an AzureOpenAIConnection object
connection = AzureOpenAIConnection(
 name="my_azure_open_ai_connection",
 api_key="<your-api-key>",
 api_base="<your-endpoint>"
 api_version="2023-03-15-preview"
)

Create the connection, note that api_key will be scrubbed in the returned result
result = pf.connections.create_or_update(connection)
print(result)

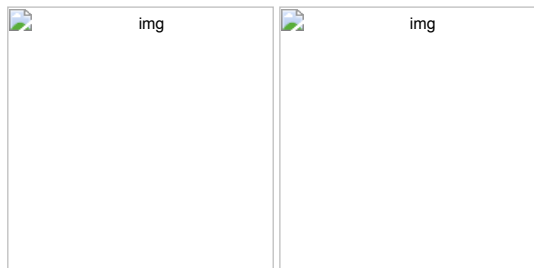
Initialize a custom connection object
connection = CustomConnection(
 name="my_custom_connection",
 # Secrets is a required field for custom connection
 secrets={"my_key": "<your-api-key>"},
 configs={"endpoint": "<your-endpoint>", "other_config": "other_value"}
)

Create the connection, note that all secret values will be scrubbed in the returned result
result = pf.connections.create_or_update(connection)
print(result)

```

:sync: VSC

On the VS Code primary sidebar > prompt flow pane. You can find the connections pane to manage your local connections. Click the "+" icon on the top right of it and follow the popped out instructions to create your new connection.



## Update a connection

:sync: CLI The commands below show how to update existing connections with new values:

```

Update an azure open ai connection with a new api base
pf connection update -n my_azure_open_ai_connection --set api_base='new_value'

Update a custom connection
pf connection update -n my_custom_connection --set configs.other_config='new_value'

```

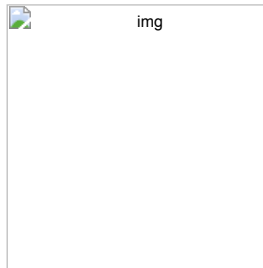
:sync: SDK The code snippet below shows how to update existing connections with new values:

```
Update an azure open ai connection with a new api base
connection = pf.connections.get(name="my_azure_open_ai_connection")
connection.api_base = "new_value"
connection.api_key = "<original-key>" # secrets are required when updating connection using sdk
result = pf.connections.create_or_update(connection)
print(connection)

Update a custom connection
connection = pf.connections.get(name="my_custom_connection")
connection.configs["other_config"] = "new_value"
connection.secrets = {"key1": "val1"} # secrets are required when updating connection using sdk
result = pf.connections.create_or_update(connection)
print(connection)
```

:sync: VSC

On the VS Code primary sidebar > prompt flow pane. You can find the connections pane to manage your local connections. Right click the item of the connection list to update or



delete your connections.

## List connections

:sync: CLI List connection command will return the connections with json list format, note that all secrets and api keys will be scrubbed:

```
pf connection list
```

:sync: SDK List connection command will return the connections object list, note that all secrets and api keys will be scrubbed:

```
from promptflow import PFClient
Get a pf client to manage connections
pf = PFClient()
List and print connections
connection_list = pf.connections.list()
for connection in connection_list:
 print(connection)
```



:sync: VSC

## Delete a connection

:sync: CLI Delete a connection with the following command:

```
pf connection delete -n <connection_name>
```

:sync: SDK Delete a connection with the following code snippet:

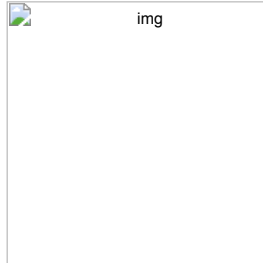
```
from promptflow import PFClient

Get a pf client to manage connections
pf = PFClient()

Delete the connection with specific name
client.connections.delete(name="my_custom_connection")
```

:sync: VSC

On the VS Code primary sidebar > prompt flow pane. You can find the connections pane to manage your local connections. Right click the item of the connection list to update or



delete your connections.

## Next steps

- Reach more detail about [connection concepts](#) ([../concepts/concept-connections.md](#)).
- Try the [connection samples](#) (<https://github.com/microsoft/promptflow/blob/main/examples/connections/connection.ipynb>).
- [Consume connections from Azure AI](#) ([../cloud/azureai/consume-connections-from-azure-ai.md](#)).

filepath: promptflow/docs/how-to-guides/set-global-configs.md content: # Set global configs

This is an experimental feature, and may change at any time. Learn [more \(faq.md#stable-vs-experimental\)](#).

Promptflow supports setting global configs to avoid passing the same parameters to each command. The global configs are stored in a yaml file, which is located at

~/promptflow/pf.yaml by default.

The config file is shared between promptflow extension and sdk/cli. Promptflow extension controls each config through UI, so the following sections will show how to set global configs using promptflow cli.

## Set config

```
pf config set <config_name>=<config_value>
```

For example:

```
pf config set connection.provider="azureml://subscriptions/<your-subscription>/resourceGroups/<your-resourcegroup>/providers/Microsoft.MachineLearningServices" --yes
```

## Show config

The following command will get all configs and show them as json format:

```
pf config show
```

After running the above config set command, show command will return the following result:

```
{
 "connection": {
 "provider": "azureml://subscriptions/<your-subscription>/resourceGroups/<your-resourcegroup>/providers/Microsoft.MachineLearningServices"
 }
}
```

## Supported configs

connection.provider



The connection provider, default to "local". There are 3 possible provider values.

### local

Set connection provider to local with `connection.provider=local`.

Connections will be saved locally. `PfClient`(or `pf connection` commands) will [manage local connections \(manage-connections.md\)](#). Consequently, the flow will be executed using these local connections.

### full azure machine learning workspace resource id

Set connection provider to a specific workspace with:

```
connection.provider=azureml://subscriptions/<your-subscription>/resourceGroups/<your-resourcegroup>/providers/Microsoft.MachineLearn.
```

When `get` or `list` connections, `PfClient`(or `pf connection` commands) will return workspace connections, and flow will be executed using these workspace connections.

*Secrets for workspace connection will not be shown by those commands, which means you may see empty dict `{}` for custom connections.*

Command `create`, `update` and `delete` are not supported for workspace connections, please manage it in workspace portal, `az ml cli` or AzureML SDK.

### azureml

In addition to the full resource id, you can designate the connection provider as "azureml" with `connection.provider=azureml`. In this case, promptflow will attempt to retrieve the workspace configuration by searching `.azureml/config.json` from the current directory, then progressively from its parent folders. So it's possible to set the workspace configuration for different flow by placing the config file in the project folder.

The expected format of the config file is as follows:

```
{
 "workspace_name": "<your-workspace-name>",
 "resource_group": "<your-resource-group>",
 "subscription_id": "<your-subscription-id>"
}
```

❏ *Tips In addition to the CLI command line setting approach, we also support setting this connection provider through the VS Code extension UI. [Click here to learn more](#) ([./cloud/azureai/consume-connections-from-azure-ai.md](#)).*

filepath: promptflow/docs/how-to-guides/enable-streaming-mode.md content: # Use streaming endpoints deployed from prompt flow

In prompt flow, you can [deploy flow as REST endpoint \(./deploy-a-flow/index.md\)](#) for real-time inference.

When consuming the endpoint by sending a request, the default behavior is that the online endpoint will keep waiting until the whole response is ready, and then send it back to the client. This can cause a long delay for the client and a poor user experience.

To avoid this, you can use streaming when you consume the endpoints. Once streaming enabled, you don't have to wait for the whole response ready. Instead, the server will send back the response in chunks as they are generated. The client can then display the response progressively, with less waiting time and more interactivity.

This article will describe the scope of streaming, how streaming works, and how to consume streaming endpoints.

## Create a streaming enabled flow

If you want to use the streaming mode, you need to create a flow that has a node that produces a string generator as the flow's output. A string generator is an object that can return one string at a time when requested. You can use the following types of nodes to create a string generator:

- **LLM node:** This node uses a large language model to generate natural language responses based on the input.

```
{# Sample prompt template for LLM node #}

system:
You are a helpful assistant.

user:
{{question}}
```

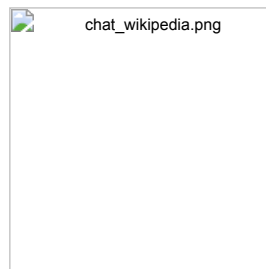
- Python tools node: This node allows you to write custom Python code that can yield string outputs. You can use this node to call external APIs or libraries that support streaming. For example, you can use this code to echo the input word by word:

```
from promptflow import tool

Sample code echo input by yield in Python tool node

@tool
def my_python_tool(paragraph: str) -> str:
 yield "Echo: "
 for word in paragraph.split():
 yield word + " "
```

In this guide, we will use the "Chat with Wikipedia" (<https://github.com/microsoft/promptflow/tree/main/examples/flows/chat/chat-with-wikipedia>) sample flow as an example. This flow processes the user's question, searches Wikipedia for relevant articles, and answers the question with information from the articles. It uses streaming mode to show the progress of the answer generation.



## Deploy the flow as an online endpoint

To use the streaming mode, you need to deploy your flow as an online endpoint. This will allow you to send requests and receive responses from your flow in real time.

Follow [this guide \(/deploy-a-flow/index.md\)](#) to deploy your flow as an online endpoint.

[!NOTE]

You can follow this document to deploy an [online endpoint \(https://learn.microsoft.com/en-us/azure/machine-learning/prompt-flow/how-to-deploy-for-real-time-inference?view=azureml-api-2\)](https://learn.microsoft.com/en-us/azure/machine-learning/prompt-flow/how-to-deploy-for-real-time-inference?view=azureml-api-2). Please deploy with runtime environment version later than version 20230816.v10. You can check your runtime version and update runtime in the [run time detail page](#).

## Understand the streaming process

When you have an online endpoint, the client and the server need to follow specific principles for [content negotiation \(https://developer.mozilla.org/en-US/docs/Web/HTTP/Content\\_negotiation\)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Content_negotiation) to utilize the streaming mode:

Content negotiation is like a conversation between the client and the server about the preferred format of the data they want to send and receive. It ensures effective communication and agreement on the format of the exchanged data.

To understand the streaming process, consider the following steps:

- First, the client constructs an HTTP request with the desired media type included in the `Accept` header. The media type tells the server what kind of data format the client expects. It's like the client saying, "Hey, I'm looking for a specific format for the data you'll send me. It could be JSON, text, or something else." For example, `application/json` indicates a preference for JSON data, `text/event-stream` indicates a desire for streaming data, and `*/*` means the client accepts any data format.

[!NOTE]

*If a request lacks an `Accept` header or has empty `Accept` header, it implies that the client will accept any media type in response. The server treats it as `*/*`.*

- Next, the server responds based on the media type specified in the `Accept` header. It's important to note that the client may request multiple media types in the `Accept` header, and the server must consider its capabilities and format priorities to determine the appropriate response.
  - First, the server checks if `text/event-stream` is explicitly specified in the `Accept` header:
    - For a stream-enabled flow, the server returns a response with a `Content-Type` of `text/event-stream`, indicating that the data is being streamed.

- For a non-stream-enabled flow, the server proceeds to check for other media types specified in the header.
- If `text/event-stream` is not specified, the server then checks if `application/json` or `*/*` is specified in the `Accept` header:
  - In such cases, the server returns a response with a `Content-Type` of `application/json`, providing the data in JSON format.
- If the `Accept` header specifies other media types, such as `text/html`:
  - The server returns a 424 response with a PromptFlow runtime error code `UserError` and a runtime HTTP status 406, indicating that the server cannot fulfill the request with the requested data format.

*Note: Please refer [handle errors](#) for details.*

- Finally, the client checks the `Content-Type` response header. If it is set to `text/event-stream`, it indicates that the data is being streamed.

Let's take a closer look at how the streaming process works. The response data in streaming mode follows the format of [server-sent events \(SSE\)](https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events) ([https://developer.mozilla.org/en-US/docs/Web/API/Server-sent\\_events](https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events)).

The overall process works as follows:

## 0. The client sends a message to the server.

```
POST https://<your-endpoint>.inference.ml.azure.com/score
Content-Type: application/json
Authorization: Bearer <key or token of your endpoint>
Accept: text/event-stream

{
 "question": "Hello",
 "chat_history": []
}
```

**[NOTE]**

*The `Accept` header is set to `text/event-stream` to request a stream response.*

## 1. The server sends back the response in streaming mode.

```
HTTP/1.1 200 OK
Content-Type: text/event-stream; charset=utf-8
Connection: close
Transfer-Encoding: chunked

data: {"answer": ""}

data: {"answer": "Hello"}

data: {"answer": "!"}

data: {"answer": " How"}

data: {"answer": " can"}

data: {"answer": " I"}

data: {"answer": " assist"}

data: {"answer": " you"}

data: {"answer": " today"}

data: {"answer": " ?"}

data: {"answer": ""}
```

Note that the `Content-Type` is set to `text/event-stream; charset=utf-8`, indicating the response is an event stream.

The client should decode the response data as server-sent events and display them incrementally. The server will close the HTTP connection after all the data is sent.

Each response event is the delta to the previous event. It is recommended for the client to keep track of the merged data in memory and send them back to the server as chat history in the next request.

## 2. The client sends another chat message, along with the full chat history, to the server.

```
POST https://<your-endpoint>.inference.ml.azure.com/score
Content-Type: application/json
Authorization: Bearer <key or token of your endpoint>
Accept: text/event-stream

{
 "question": "Glad to know you!",
 "chat_history": [
 {
 "inputs": {
 "question": "Hello"
 },
 "outputs": {
 "answer": "Hello! How can I assist you today?"
 }
 }
]
}
```

## 3. The server sends back the answer in streaming mode.

```
HTTP/1.1 200 OK
Content-Type: text/event-stream; charset=utf-8
Connection: close
Transfer-Encoding: chunked
```

```
data: {"answer": ""}

data: {"answer": "Nice"}

data: {"answer": " to"}

data: {"answer": " know"}

data: {"answer": " you"}

data: {"answer": " too"}

data: {"answer": "!"}

data: {"answer": " Is"}

data: {"answer": " there"}

data: {"answer": " anything"}

data: {"answer": " I"}

data: {"answer": " can"}

data: {"answer": " help"}

data: {"answer": " you"}

data: {"answer": " with"}

data: {"answer": "?"}

data: {"answer": ""}
```

4. The chat continues in a similar way.

## Handle errors

The client should check the HTTP response code first. See [this table \(https://learn.microsoft.com/azure/machine-learning/how-to-troubleshoot-online-endpoints?view=azureml-api-2&tabs=cli#http-status-codes\)](https://learn.microsoft.com/azure/machine-learning/how-to-troubleshoot-online-endpoints?view=azureml-api-2&tabs=cli#http-status-codes) for common error codes returned by online endpoints.

If the response code is "424 Model Error", it means that the error is caused by the model's code. The error response from a PromptFlow model always follows this format:

```
{
 "error": {
 "code": "UserError",
 "message": "Media type text/event-stream in Accept header is not acceptable. Supported media type(s) - application/json",
 }
}
```

- It is always a JSON dictionary with only one key "error" defined.
- The value for "error" is a dictionary, containing "code", "message".
- "code" defines the error category. Currently, it may be "UserError" for bad user inputs and "SystemError" for errors inside the service.
- "message" is a description of the error. It can be displayed to the end user.

## How to consume the server-sent events

## Consume using Python

In this sample usage, we are using the `SSEClient` class. This class is not a built-in Python class and needs to be installed separately. You can install it via pip:

```
pip install sseclient-py
```

A sample usage would like:

```
import requests
from sseclient import SSEClient
from requests.exceptions import HTTPError

try:
 response = requests.post(url, json=body, headers=headers, stream=stream)
 response.raise_for_status()

 content_type = response.headers.get('Content-Type')
 if "text/event-stream" in content_type:
 client = SSEClient(response)
 for event in client.events():
 # Handle event, i.e. print to stdout
 else:
 # Handle json response

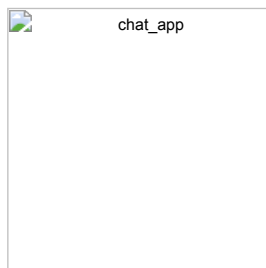
except HTTPError:
 # Handle exceptions
```

## Consume using JavaScript

There are several libraries to consume server-sent events in JavaScript. Here is [one of them as an example \(https://www.npmjs.com/package/sse.js?activeTab=code\)](https://www.npmjs.com/package/sse.js?activeTab=code).

## A sample chat app using Python

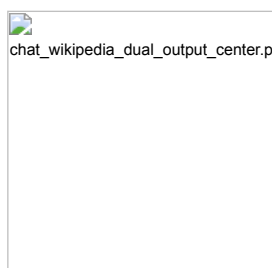
Here is a sample chat app written in Python. (Click [here \(../media/how-to-guides/how-to-enable-streaming-mode/scripts/chat\\_app.py\)](#) to view the source code.)



## Advance usage - hybrid stream and non-stream flow output

Sometimes, you may want to get both stream and non-stream results from a flow output. For example, in the "Chat with Wikipedia" flow, you may want to get not only LLM's answer, but also the list of URLs that the flow searched. To do this, you need to modify the flow to output a combination of stream LLM's answer and non-stream URL list.

In the sample "Chat With Wikipedia" flow, the output is connected to the LLM node `augmented_chat`. To add the URL list to the output, you need to add an output field with the name `url` and the value `${get_wiki_url.output}`.



The output of the flow will be a non-stream field as the base and a stream field as the delta. Here is an example of request and response.

## 0. The client sends a message to the server.

```
POST https://<your-endpoint>.inference.ml.azure.com/score
Content-Type: application/json
Authorization: Bearer <key or token of your endpoint>
Accept: text/event-stream

{
 "question": "When was ChatGPT launched?",
 "chat_history": []
}
```

## 1. The server sends back the answer in streaming mode.

```
HTTP/1.1 200 OK
Content-Type: text/event-stream; charset=utf-8
Connection: close
Transfer-Encoding: chunked

data: {"url": ["https://en.wikipedia.org/w/index.php?search=ChatGPT", "https://en.wikipedia.org/w/index.php?search=GPT-4"]}

data: {"answer": ""}

data: {"answer": "Chat"}

data: {"answer": "G"}

data: {"answer": "PT"}

data: {"answer": " was"}

data: {"answer": " launched"}

data: {"answer": " on"}

data: {"answer": " November"}

data: {"answer": " "}

data: {"answer": "30"}

data: {"answer": ", "}

data: {"answer": " "}

data: {"answer": "202"}

data: {"answer": "2"}

data: {"answer": "."}

data: {"answer": " \n\n"}

...

data: {"answer": "PT"}

data: {"answer": ""}
```

## 2. The client sends another chat message, along with the full chat history, to the server.

```
POST https://<your-endpoint>.inference.ml.azure.com/score
Content-Type: application/json
Authorization: Bearer <key or token of your endpoint>
Accept: text/event-stream

{
 "question": "When did OpenAI announce GPT-4? How long is it between these two milestones?",
 "chat_history": [
 {
 "inputs": {
 "question": "When was ChatGPT launched?"
 },
 "outputs": {
 "url": [
 "https://en.wikipedia.org/w/index.php?search=ChatGPT",
 "https://en.wikipedia.org/w/index.php?search=GPT-4"
],
 "answer": "ChatGPT was launched on November 30, 2022. \n\nSOURCES: https://en.wikipedia.org/w/index.php?search=ChatGPT"
 }
 }
]
}
```

3. The server sends back the answer in streaming mode.



```
HTTP/1.1 200 OK
Content-Type: text/event-stream; charset=utf-8
Connection: close
Transfer-Encoding: chunked

data: {"url": ["https://en.wikipedia.org/w/index.php?search=Generative pre-trained transformer ", "https://en.wikipedia.org/w/index.php?search=Generative pre-trained transformer "]}

data: {"answer": ""}

data: {"answer": "Open"}

data: {"answer": "AI"}

data: {"answer": " released"}

data: {"answer": " G"}

data: {"answer": "PT"}

data: {"answer": "-"}

data: {"answer": "4"}

data: {"answer": " in"}

data: {"answer": " March"}

data: {"answer": " "}

data: {"answer": "202"}

data: {"answer": "3"}

data: {"answer": "."}

data: {"answer": " Chat"}

data: {"answer": "G"}

data: {"answer": "PT"}

data: {"answer": " was"}

data: {"answer": " launched"}

data: {"answer": " on"}

data: {"answer": " November"}

data: {"answer": " "}

data: {"answer": "30"}

data: {"answer": ", "}

data: {"answer": " "}

data: {"answer": "202"}

data: {"answer": "2"}

data: {"answer": "."}

data: {"answer": " The"}
```

```
data: {"answer": " time"}

data: {"answer": " between"}

data: {"answer": " these"}

data: {"answer": " two"}

data: {"answer": " milestones"}

data: {"answer": " is"}

data: {"answer": " approximately"}

data: {"answer": " "}

data: {"answer": "3"}

data: {"answer": " months"}

data: {"answer": ".\n\n"}

...

data: {"answer": "Chat"}

data: {"answer": "G"}

data: {"answer": "PT"}

data: {"answer": ""}
```

filepath: promptflow/docs/how-to-guides/manage-runs.md content: # Manage runs

This is an experimental feature, and may change at any time. Learn [more \(faq.md#stable-vs-experimental\)](#).

This documentation will walk you through how to manage your runs with CLI, SDK and VS Code Extension.

In general:

- For CLI, you can run `pf/pfazure run --help` in terminal to see the help messages.
- For SDK, you can refer to [Promptflow Python Library Reference \(../reference/python-library-reference/promptflow.md\)](#) and check `PFClient.runs` for more run operations.

Let's take a look at the following topics:

- [Manage runs](#)
  - [Create a run](#)
  - [Get a run](#)
  - [Show run details](#)
  - [Show run metrics](#)
  - [Visualize a run](#)
  - [List runs](#)
  - [Update a run](#)
  - [Archive a run](#)
  - [Restore a run](#)
  - [Delete a run](#)

## Create a run

:sync: CLI To create a run against bulk inputs, you can write the following YAML file.

```
$schema: https://azuremlschemas.azureedge.net/promptflow/latest/Run.schema.json
flow: ../web_classification
data: ../webClassification1.jsonl
column_mapping:
 url: "${data.url}"
variant: ${summarize_text_content.variant_0}
```

To create a run against existing run, you can write the following YAML file.

```
$schema: https://azuremlschemas.azureedge.net/promptflow/latest/Run.schema.json
flow: ../classification_accuracy_evaluation
data: ../webClassification1.jsonl
column_mapping:
 groundtruth: "${data.answer}"
 prediction: "${run.outputs.category}"
run: <existing-flow-run-name>
```

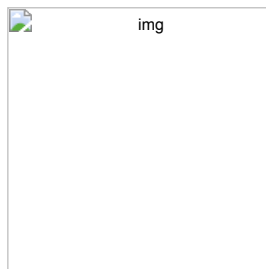
Reference [here \(https://aka.ms/pf/column-mapping\)](https://aka.ms/pf/column-mapping), for detailed information for column mapping. You can find additional information about flow yaml schema in [Run YAML Schema](#) ([./reference/run-yaml-schema-reference.md](#)).

After preparing the yaml file, use the CLI command below to create them:

```
create the flow run
pf run create -f <path-to-flow-run>

create the flow run and stream output
pf run create -f <path-to-flow-run> --stream
```

The expected result is as follows if the run is created successfully.



:sync: SDK Using SDK, create Run object and submit it with PFClient. The following code snippet shows how to import the required class and create the run:

```
from promptflow import PFClient
from promptflow.entities import Run

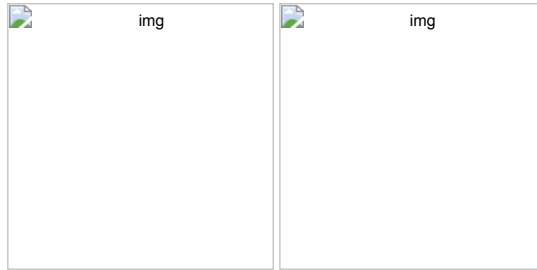
Get a pf client to manage runs
pf = PFClient()

Initialize an Run object
run = Run(
 flow="<path-to-local-flow>",
 # run flow against local data or existing run, only one of data & run can be specified.
 data="<path-to-data>",
 run="<existing-run-name>",
 column_mapping={"url": "${data.url}"},
 variant="${summarize_text_content.variant_0}"
)

Create the run
result = pf.runs.create_or_update(run)
print(result)
```

:sync: VS Code Extension

You can click on the actions on the top of the default yaml editor or the visual editor for the flow.dag.yaml files to trigger flow batch runs.

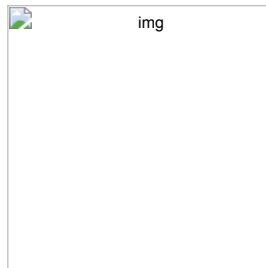


## Get a run

:sync: CLI

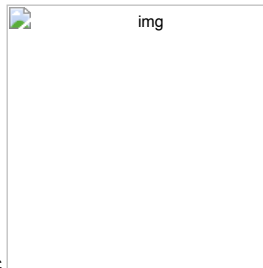
Get a run in CLI with JSON format.

```
pf run show --name <run-name>
```



:sync: SDK Show run with PFClient

```
from promptflow import PFClient
Get a pf client to manage runs
pf = PFClient()
Get and print the run
run = pf.runs.get(name="<run-name>")
print(run)
```



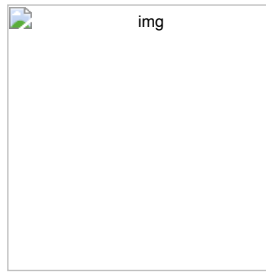
:sync: VSC

## Show run details

:sync: CLI

Get run details with TABLE format.

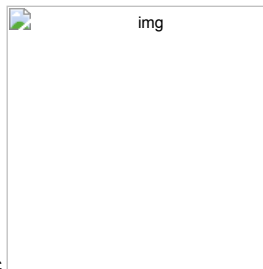
```
pf run show --name <run-name>
```



:sync: SDK Show run details with PFClient

```
from promptflow import PFClient
from tabulate import tabulate

Get a pf client to manage runs
pf = PFClient()
Get and print the run-details
run_details = pf.runs.get_details(name="<run-name>")
print(tabulate(details.head(max_results), headers="keys", tablefmt="grid"))
```



:sync: VSC

## Show run metrics

:sync: CLI

Get run metrics with JSON format.

```
pf run show-metrics --name <run-name>
```



:sync: SDK Show run metrics with PFClient

```
from promptflow import PFClient
import json

Get a pf client to manage runs
pf = PFClient()
Get and print the run-metrics
run_details = pf.runs.get_metrics(name="<run-name>")
print(json.dumps(metrics, indent=4))
```

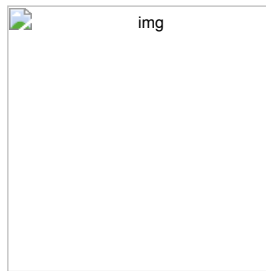
## Visualize a run

:sync: CLI

Visualize run in browser.

```
pf run visualize --names <run-name>
```

A browser will open and display run outputs.



:sync: SDK Visualize run with PFClient

```
from promptflow import PFClient

Get a pf client to manage runs
pf = PFClient()

Visualize the run
client.runs.visualize(runs="<run-name>")
```

:sync: VSC

On the VS Code primary sidebar > the prompt flow pane, there is a run list. It will list all the runs on your machine. Select one or more items and click the "visualize" button on the top-right to visualize the local runs.



## List runs

:sync: CLI

List runs with JSON format.

```
pf run list
```



:sync: SDK List with PFClient

```
from promptflow import PFClient

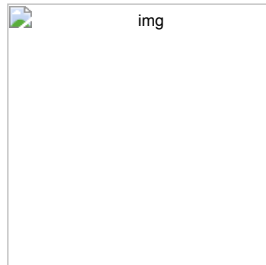
Get a pf client to manage runs
pf = PFClient()

list runs
runs = pf.runs.list()

print(runs)
```

:sync: VSC

On the VS Code primary sidebar > the prompt flow pane, there is a run list. It will list all the runs on your machine. Hover on it to view more details.



## Update a run

:sync: CLI

Get run metrics with JSON format.

```
pf run update --name <run-name> --set display_name=new_display_name
```

:sync: SDK Update run with PFClient

```
from promptflow import PFClient

Get a pf client to manage runs
pf = PFClient()

Get and print the run-metrics
run = pf.runs.update(name="<run-name>", display_name="new_display_name")

print(run)
```

## Archive a run

:sync: CLI

Archive the run so it won't show in run list results.

```
pf run archive --name <run-name>
```

:sync: SDK Archive with PFClient

```
from promptflow import PFClient

Get a pf client to manage runs
pf = PFClient()

archive a run
client.runs.archive(name="<run-name>")
```



:sync: VSC

## Restore a run

:sync: CLI

Restore an archived run so it can show in run list results.

```
pf run restore --name <run-name>
```

:sync: SDK Restore with `PFClient`

```
from promptflow import PFClient

Get a pf client to manage runs
pf = PFClient()
restore a run
client.runs.restore(name="<run-name>")
```

## Delete a run

:sync: CLI

Caution: `pf run delete` is irreversible. This operation will delete the run permanently from your local disk. Both run entity and output data will be deleted.

Delete will fail if the run name is not valid.

```
pf run delete --name <run-name>
```

:sync: SDK Delete with `PFClient`

```
from promptflow import PFClient

Get a pf client to manage runs
pf = PFClient()
delete a run
client.runs.delete(name="run-name")
```

filepath: `promptflow/docs/how-to-guides/faq.md` content: # Frequency asked questions (FAQ)

## General

### Stable vs experimental

Prompt flow provides both stable and experimental features in the same SDK.

Feature status	Description
<b>Production ready</b>	
Stable features	These features are recommended for most use cases and production environments. They are updated less frequently than experimental features.



Feature status	Description
Developmental	These features are newly developed capabilities & updates that may not be ready or fully tested for production usage.
Experimental features	<p>While the features are typically functional, they can include some breaking changes. Experimental features are used to iron out SDK breaking bugs, and will only receive updates for the duration of the testing period. Experimental features are also referred to as features that are in <b>preview</b>.</p> <p>As the name indicates, the experimental (preview) features are for experimenting and is <b>not considered bug free or stable</b>. For this reason, we only recommend experimental features to advanced users who wish to try out early versions of capabilities and updates, and intend to participate in the reporting of bugs and glitches.</p>

## OpenAI 1.x support

Please use the following command to upgrade promptflow for openai 1.x support:

```
pip install promptflow>=1.1.0
pip install promptflow-tools>=1.0.0
```

Note that the command above will upgrade your openai package a version later than 1.0.0, which may introduce breaking changes to custom tool code.

Reach [OpenAI migration guide \(https://github.com/openai/openai-python/discussions/742\)](https://github.com/openai/openai-python/discussions/742) for more details.

## Troubleshooting

### Connection creation failed with StoreConnectionEncryptionKeyError

Connection creation failed with StoreConnectionEncryptionKeyError: System keyring backend service not found in your operating system

This error raised due to keyring can't find an available backend to store keys. For example [macOS Keychain \(https://en.wikipedia.org/wiki/Keychain\\_%28software%29\)](https://en.wikipedia.org/wiki/Keychain_%28software%29) and [Windows Credential Locker \(https://learn.microsoft.com/en-us/windows/uwp/security/credential-locker\)](https://learn.microsoft.com/en-us/windows/uwp/security/credential-locker) are valid keyring backends.

To resolve this issue, install the third-party keyring backend or write your own keyring backend, for example: `pip install keyrings.alt`

For more detail about keyring third-party backend, please refer to 'Third-Party Backends' in [keyring \(https://pypi.org/project/keyring/\)](https://pypi.org/project/keyring/).

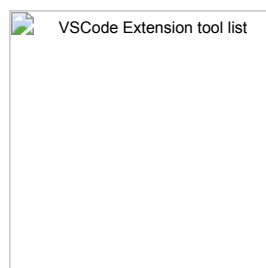
### Pf visualize show error: "tcgetpgrp failed: Not a tty"

If you are using WSL, this is a known issue for `webbrowser` under WSL; see [this issue \(https://github.com/python/cpython/issues/89752\)](https://github.com/python/cpython/issues/89752) for more information. Please try to upgrade your WSL to 22.04 or later, this issue should be resolved.

If you are still facing this issue with WSL 22.04 or later, or you are not even using WSL, please open an issue to us.

### Installed tool not appearing in VSCode Extension tool list

After installing a tool package via `pip install [tool-package-name]`, the new tool may not immediately appear in the tool list within the VSCode Extension, as shown below:



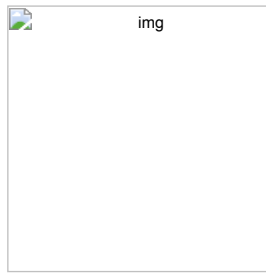
This is often due to outdated cache. To refresh the tool list and make newly installed tools visible:

1. Open the VSCode Extension window.
2. Bring up the command palette by pressing "Ctrl+Shift+P".
3. Type and select the "Developer: Reload Webviews" command.
4. Wait a moment for the tool list refreshing.

Reloading clears the previous cache and populates the tool list with any newly installed tools. So that the missing tools are now visible.

## Set logging level

Promptflow uses logging module to log messages. You can set logging level via environment variable `PF_LOGGING_LEVEL`, valid values includes `CRITICAL`, `ERROR`, `WARNING`, `INFO`, `DEBUG`, default to `INFO`. Below is the serving logs after setting `PF_LOGGING_LEVEL` to `DEBUG`:



Compare to the serving logs with `WARNING` level:



## Set environment variables

Currently, promptflow supports the following environment variables:

### **PF\_WORKER\_COUNT**

Effective for batch run only, count of parallel workers in batch run execution.

The default value is 4 (was 16 when promptflow<1.4.0)

Please take the following points into consideration when changing it:

1. The concurrency should not exceed the total data rows count. Otherwise, the execution may slow down due to additional time spent on process startup and shutdown.
2. High parallelism may cause the underlying API call to reach the rate limit of your LLM endpoint. In which case you can decrease the `PF_WORKER_COUNT` or increase the rate limit. Please refer to [this doc \(https://learn.microsoft.com/en-us/azure/ai-services/openai/how-to/quota\)](https://learn.microsoft.com/en-us/azure/ai-services/openai/how-to/quota) on quota management. Then you can refer to this expression to set up the concurrency.

```
PF_WORKER_COUNT <= TPM * duration_seconds / token_count / 60
```

TPM: token per minute, capacity rate limit of your LLM endpoint

duration\_seconds: single flow run duration in seconds

token\_count: single flow run token count

For example, if your endpoint TPM (token per minute) is 50K, the single flow run takes 10k tokens and runs for 30s, pls do not set up `PF_WORKER_COUNT` bigger than 2. This is a rough estimation. Please also consider collaboration (teammates use the same endpoint at the same time) and tokens consumed in deployed inference endpoints, playground and other cases which might send request to your LLM endpoints.

### **PF\_BATCH\_METHOD**

Valid for batch run only. Optional values: 'spawn', 'fork'.

#### **spawn**

1. The child processes will not inherit resources of the parent process, therefore, each process needs to reinitialize the resources required for the flow, which may use more system memory.
2. Starting a process is slow because it will take some time to initialize the necessary resources.

#### **fork**

1. Use the copy-on-write mechanism, the child processes will inherit all the resources of the parent process, thereby using less system memory.

2. The process starts faster as it doesn't need to reinitialize resources.

Note: Windows only supports spawn, Linux and macOS support both spawn and fork.

## How to configure environment variables

1. Configure environment variables in `flow.dag.yaml`. Example:

```
inputs: []
outputs: []
nodes: []
environment_variables:
 PF_WORKER_COUNT: 2
 PF_BATCH_METHOD: "spawn"
 MY_CUSTOM_SETTING: my_custom_value
```

2. Specify environment variables when submitting runs.

:sync: CLI

Use this parameter: `--environment-variable` to specify environment variables. Example: `--environment-variable PF_WORKER_COUNT="2"`

`PF_BATCH_METHOD="spawn"`.

:sync: SDK

Specify environment variables when creating run. Example:

```
pf = PFClient(
 credential=credential,
 subscription_id="<SUBSCRIPTION_ID>",
 resource_group_name="<RESOURCE_GROUP>",
 workspace_name="<AML_WORKSPACE_NAME>",
)

flow = "web-classification"
data = "web-classification/data.jsonl"
runtime = "example-runtime-ci"

environment_variables = {"PF_WORKER_COUNT": "2", "PF_BATCH_METHOD": "spawn"}

create run
base_run = pf.run(
 flow=flow,
 data=data,
 runtime=runtime,
 environment_variables=environment_variables,
)
```

:sync: VS Code Extension

VSCode Extension supports specifying environment variables only when submitting batch runs. Specify environment variables in `batch_run_create.yaml`. Example:

```
name: flow_name
display_name: display_name
flow: flow_folder
data: data_file
column_mapping:
 customer_info: <Please select a data input>
 history: <Please select a data input>
environment_variables:
 PF_WORKER_COUNT: "2"
 PF_BATCH_METHOD: "spawn"
```

## Priority

The environment variables specified when submitting runs always takes precedence over the environment variables in the `flow.dag.yaml` file.

filepath: promptflow/docs/how-to-guides/init-and-test-a-flow.md content: # Initialize and test a flow

This is an experimental feature, and may change at any time. Learn [more \(faq.md#stable-vs-experimental\)](#).

From this document, customer can initialize a flow and test it.

# Initialize flow

Creating a flow folder with code/prompts and yaml definitions of the flow.

## Initialize flow from scratch

Promptflow can [create three types of flow folder \(https://promptflow.azurewebsites.net/concepts/concept-flows.html#flow-types\)](https://promptflow.azurewebsites.net/concepts/concept-flows.html#flow-types):

- **standard**: Basic structure of flow folder.
- **chat**: Chat flow is designed for conversational application development, building upon the capabilities of standard flow and providing enhanced support for chat inputs/outputs and chat history management.
- **evaluation**: Evaluation flows are special types of flows that assess how well the outputs of a flow align with specific criteria and goals.

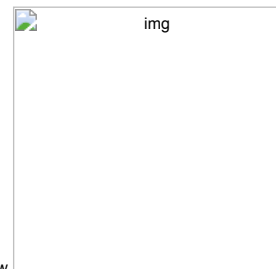
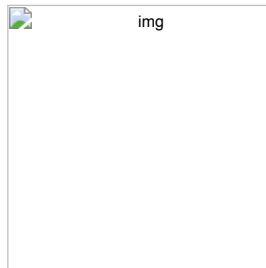
:sync: CLI

```
Create a flow
pf flow init --flow <flow-name>

Create a chat flow
pf flow init --flow <flow-name> --type chat
```

:sync: VS Code Extension

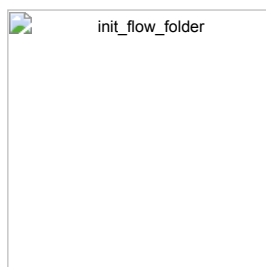
Use VS Code explorer pane > directory icon > right click > the "New flow in this directory" action. Follow the popped out dialog to initialize your flow in the target folder.



Alternatively, you can use the "Create new flow" action on the prompt flow pane > quick access section to create a new flow

Structure of flow folder:

- **flow.dag.yaml**: The flow definition with inputs/outputs, nodes, tools and variants for authoring purpose.
- **.promptflow/flow.tools.json**: It contains tools meta referenced in `flow.dag.yaml`.
- **Source code files (.py, .jinja2)**: User managed, the code scripts referenced by tools.
- **requirements.txt**: Python package dependencies for this flow.



## Create from existing code

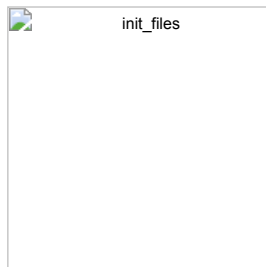
Customer needs to pass the path of tool script to `entry`, and also needs to pass in the promptflow template dict to `prompt-template`, which the key is the input name of the tool and the value is the path to the promptflow template. Promptflow CLI can generate the yaml definitions needed for prompt flow from the existing folder, using the tools script and prompt templates.

```
Create a flow in existing folder
pf flow init --flow <flow-name> --entry <tool-script-path> --function <tool-function-name> --prompt-template <prompt-param-name>=<prompt-template>
```

Take [customer-intent-extraction](https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/customer-intent-extraction) (<https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/customer-intent-extraction>) for example, which demonstrating how to convert a langchain code into a prompt flow.



In this case, promptflow CLI generates `flow.dag.yaml`, `.promptflow/flow.tools.json` and `extract_intent_tool.py`, it is a python tool in the flow.



## Test a flow

Testing flow will NOT create a batch run record, therefore it's unable to use commands like `pf run show-details` to get the run information. If you want to persist the run record, see [Run and evaluate a flow](#) ([./run-and-evaluate-a-flow/index.md](#)).

Promptflow also provides ways to test the initialized flow or flow node. It will help you quickly test your flow.

## Visual editor on the VS Code for prompt flow.

:sync: VS Code Extension

Open the `flow.dag.yaml` file of your flow. On the top of the yaml editor you can find the "Visual editor" action. Use it to open the Visual editor with GUI support.



## Test flow

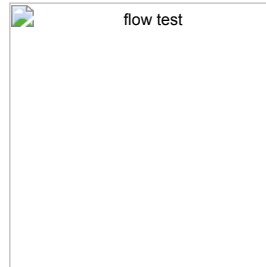
Customer can use CLI or VS Code extension to test the flow.

:sync: CLI

```
Test flow
pf flow test --flow <flow-name>

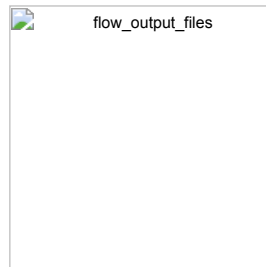
Test flow with specified variant
pf flow test --flow <flow-name> --variant '${<node-name>.<variant-name>}'
```

The log and result of flow test will be displayed in the terminal.



Promptflow CLI will generate test logs and outputs in `.promptflow`:

- **flow.detail.json**: Details info of flow test, include the result of each node.
- **flow.log**: The log of flow test.
- **flow.output.json**: The result of flow test.



:sync: SDK

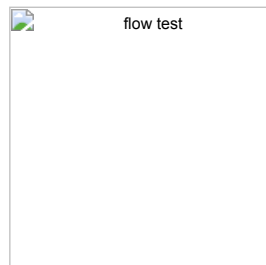
The return value of `test` function is the flow outputs.

```
from promptflow import PFClient

pf_client = PFClient()

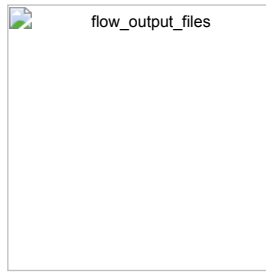
Test flow
inputs = {"<flow_input_name>": "<flow_input_value>"} # The inputs of the flow.
flow_result = pf_client.test(flow="<flow_folder_path>", inputs=inputs)
print(f"Flow outputs: {flow_result}")
```

The log and result of flow test will be displayed in the terminal.

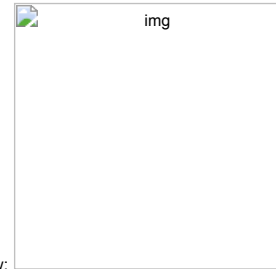


Promptflow CLI will generate test logs and outputs in `.promptflow`:

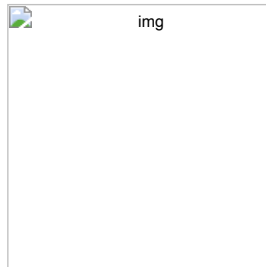
- **flow.detail.json**: Details info of flow test, include the result of each node.
- **flow.log**: The log of flow test.
- **flow.output.json**: The result of flow test.



:sync: VS Code Extension



You can use the action either on the default yaml editor or the visual editor to trigger flow test. See the snapshots below:



## Test a single node in the flow

Customer can test a single python node in the flow. It will use customer provides date or the default value of the node as input. It will only use customer specified node to execute with the input.

:sync: CLI

Customer can execute this command to test the flow.

```
Test flow node
pf flow test --flow <flow-name> --node <node-name>
```

The log and result of flow node test will be displayed in the terminal. And the details of node test will generated to `.promptflow/flow-<node-name>.node.detail.json`.

:sync: SDK

Customer can execute this command to test the flow. The return value of `test` function is the node outputs.

```
from promptflow import PFClient

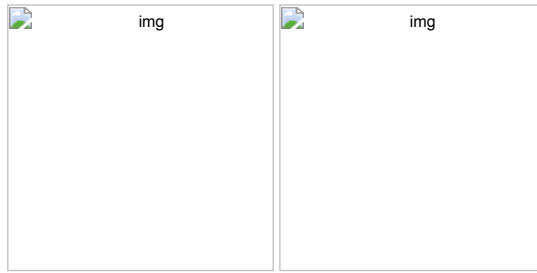
pf_client = PFClient()

Test not iun the flow
inputs = {<node_input_name>: <node_input_value>} # The inputs of the node.
node_result = pf_client.test(flow=<flow_folder_path>, inputs=inputs, node=<node_name>)
print(f"Node outputs: {node_result}")
```

The log and result of flow node test will be displayed in the terminal. And the details of node test will generated to `.promptflow/flow-<node-name>.node.detail.json`.

:sync: VS Code Extension

The prompt flow extension provides inline actions in both default yaml editor and visual editor to trigger single node runs.



## Test with interactive mode

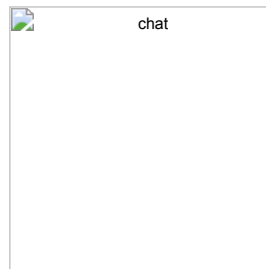
:sync: CLI

Promptflow CLI provides a way to start an interactive chat session for chat flow. Customer can use below command to start an interactive chat session:

```
Chat in the flow
pf flow test --flow <flow-name> --interactive
```

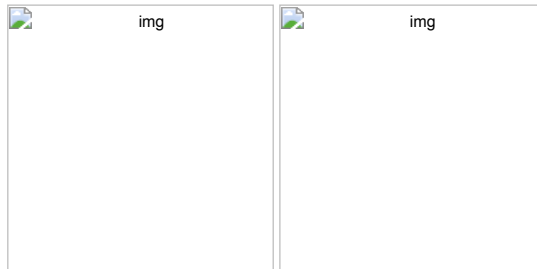
After executing this command, customer can interact with the chat flow in the terminal. Customer can press **Enter** to send the message to chat flow. And customer can quit with **ctrl+C**. Promptflow CLI will distinguish the output of different roles by color, User input, Bot output, Flow script output, Node output.

Using this [chat flow \(https://github.com/microsoft/promptflow/tree/main/examples/flows/chat/basic-chat\)](https://github.com/microsoft/promptflow/tree/main/examples/flows/chat/basic-chat) to show how to use interactive mode.



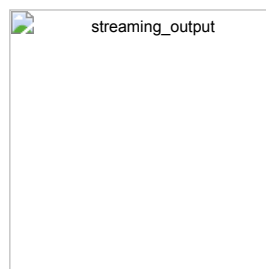
:sync: VS Code Extension

If a flow contains chat inputs or chat outputs in the flow interface, there will be a selection when triggering flow test. You can select the interactive mode if you want to.



When the [LLM node \(https://promptflow.azurewebsites.net/tools-reference/lm-tool.html\)](https://promptflow.azurewebsites.net/tools-reference/lm-tool.html) in the chat flow that is connected to the flow output, Promptflow SDK streams the results of the LLM node.

:sync: CLI The flow result will be streamed in the terminal as shown below.



:sync: SDK

The LLM node return value of `test` function is a generator, you can consume the result by this way:



```

from promptflow import PFClient

pf_client = PFClient()

Test flow
inputs = {"<flow_input_name>": "<flow_input_value>"} # The inputs of the flow.
flow_result = pf_client.test(flow="<flow_folder_path>", inputs=inputs)
for item in flow_result["<LLM_node_output_name>"]:
 print(item)

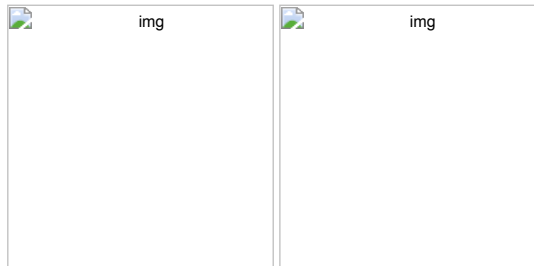
```

## Debug a single node in the flow

Customer can debug a single python node in VScode by the extension.

:sync: VS Code Extension

Break points and debugging functionalities for the Python steps in your flow. Just set the break points and use the debug actions on either default yaml editor or visual editor.



## Next steps

- [Add conditional control to a flow \(./add-conditional-control-to-a-flow.md\)](#)

filepath: promptflow/docs/how-to-guides/develop-a-flow/index.md content: # Develop a flow We provide guides on how to develop a flow by writing a flow yaml from scratch in this section.

```

:maxdepth: 1
:hidden:

develop-standard-flow
develop-chat-flow
develop-evaluation-flow
referencing-external-files-or-folders-in-a-flow

```

filepath: promptflow/docs/how-to-guides/develop-a-flow/develop-chat-flow.md content: # Develop chat flow

This is an experimental feature, and may change at any time. Learn [more \(./faq.md#stable-vs-experimental\)](#).

From this document, you can learn how to develop a chat flow by writing a flow yaml from scratch. You can find additional information about flow yaml schema in [Flow YAML Schema \(./reference/flow-yaml-schema-reference.md\)](#).

## Flow input data

The most important elements that differentiate a chat flow from a standard flow are **chat input** and **chat history**. A chat flow can have multiple inputs, but **chat history** and **chat input** are required inputs in chat flow.

- **Chat Input:** Chat input refers to the messages or queries submitted by users to the chatbot. Effectively handling chat input is crucial for a successful conversation, as it involves understanding user intentions, extracting relevant information, and triggering appropriate responses.
- **Chat History:** Chat history is the record of all interactions between the user and the chatbot, including both user inputs and AI-generated outputs. Maintaining chat history is essential for keeping track of the conversation context and ensuring the AI can generate contextually relevant responses. Chat history is a special type of chat flow input, that stores chat messages in a structured format.

An example of chat history:

```
[
 {"inputs": {"question": "What types of container software there are?"}, "outputs": {"answer": "There are several types of con
 {"inputs": {"question": "What's the different between them?"}, "outputs": {"answer": "The main difference between the various
]
```

You can set `is_chat_input/is_chat_history` to `true` to add chat\_input/chat\_history to the chat flow.

```
inputs:
 chat_history:
 type: list
 is_chat_history: true
 default: []
 question:
 type: string
 is_chat_input: true
 default: What is ChatGPT?
```

For more information see [develop the flow using different tools](#) ([./develop-standard-flow.md#flow-input-data](#)).

## Develop the flow using different tools

In one flow, you can consume different kinds of tools. We now support built-in tool like [LLM](#) ([./reference/tools-reference/llm-tool.md](#)), [Python](#) ([./reference/tools-reference/python-tool.md](#)) and [Prompt](#) ([./reference/tools-reference/prompt-tool.md](#)) and third-party tool like [Serp API](#) ([./reference/tools-reference/serp-api-tool.md](#)), [Vector Search](#) ([./reference/tools-reference/vector\\_db\\_lookup\\_tool.md](#)), etc.

For more information see [develop the flow using different tools](#) ([./develop-standard-flow.md#develop-the-flow-using-different-tools](#)).

## Chain your flow - link nodes together

Before linking nodes together, you need to define and expose an interface.

For more information see [chain your flow](#) ([./develop-standard-flow.md#chain-your-flow---link-nodes-together](#)).

## Set flow output

**Chat output** is required output in the chat flow. It refers to the AI-generated messages that are sent to the user in response to their inputs. Generating contextually appropriate and engaging chat outputs is vital for a positive user experience.

You can set `is_chat_output` to `true` to add chat\_output to the chat flow.

```
outputs:
 answer:
 type: string
 reference: ${chat.output}
 is_chat_output: true
```

filepath: promptflow/docs/how-to-guides/develop-a-flow/develop-standard-flow.md content: # Develop standard flow

This is an experimental feature, and may change at any time. Learn [more](#) ([./faq.md#stable-vs-experimental](#)).

From this document, you can learn how to develop a standard flow by writing a flow yaml from scratch. You can find additional information about flow yaml schema in [Flow YAML Schema](#) ([./reference/flow-yaml-schema-reference.md](#)).

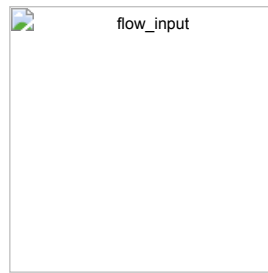
## Flow input data

The flow input data is the data that you want to process in your flow.

:sync: CLI You can add a flow input in inputs section of flow yaml.

```
inputs:
 url:
 type: string
 default: https://www.microsoft.com/en-us/d/xbox-wireless-controller-stellar-shift-special-edition/94fbjc7h0h6h
```

:sync: VS Code Extension When unfolding Inputs section in the authoring page, you can set and view your flow inputs, including input schema (name and type), and the input value.



For Web Classification sample as shown the screenshot above, the flow input is an url of string type. For more input types in a python tool, please refer to [Input types](#) ([./../reference/tools-reference/python-tool.md#types](#)).

## Develop the flow using different tools

In one flow, you can consume different kinds of tools. We now support built-in tool like [LLM](#) ([./../reference/tools-reference/llm-tool.md](#)), [Python](#) ([./../reference/tools-reference/python-tool.md](#)) and [Prompt](#) ([./../reference/tools-reference/prompt-tool.md](#)) and third-party tool like [Serp API](#) ([./../reference/tools-reference/serp-api-tool.md](#)), [Vector Search](#) ([./../reference/tools-reference/vector\\_db\\_lookup\\_tool.md](#)), etc.

## Add tool as your need

:sync: CLI You can add a tool node in nodes section of flow yaml. For example, yaml below shows how to add a Python tool node in the flow.

```
nodes:
- name: fetch_text_content_from_url
 type: python
 source:
 type: code
 path: fetch_text_content_from_url.py
 inputs:
 url: ${inputs.url}
```

:sync: VS Code Extension By selecting the tool card on the very top, you'll add a new tool node to flow.



## Edit tool

:sync: CLI You can edit the tool by simply opening the source file and making edits. For example, we provide a simple Python tool code below.

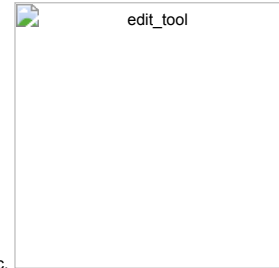
```
from promptflow import tool

The inputs section will change based on the arguments of the tool function, after you save the code
Adding type to arguments and return value will help the system show the types properly
Please update the function name/signature per need
@tool
def my_python_tool(input1: str) -> str:
 return 'hello ' + input1
```

We also provide an LLM tool prompt below.

```
Please summarize the following text in one paragraph. 100 words.
Do not add any information that is not in the text.
Text: {{text}}
Summary:
```

:sync: VS Code Extension When a new tool node is added to flow, it will be appended at the bottom of flatten view with a random name by default. At the top of each tool node card, there's a toolbar for adjusting the tool node. You can move it up or down, you can delete or rename it too. For a python tool node, you can edit the tool code by clicking the code file.



For a LLM tool node, you can edit the tool prompt by clicking the prompt file and adjust input parameters like connection, api and etc.

## Create connection

Please refer to the [Create necessary connections \(/quick-start.md#create-necessary-connections\)](#) for details.

# Chain your flow - link nodes together

Before linking nodes together, you need to define and expose an interface.

## Define LLM node interface

LLM node has only one output, the completion given by LLM provider.

As for inputs, we offer a templating strategy that can help you create parametric prompts that accept different input values. Instead of fixed text, enclose your input name in `{{ }}`, so it can be replaced on the fly. We use Jinja as our templating language. For example:

```
Your task is to classify a given url into one of the following types:
Movie, App, Academic, Channel, Profile, PDF or None based on the text content information.
The classification will be based on the url, the webpage text content summary, or both.

Here are a few examples:
{% for ex in examples %}
URL: {{ex.url}}
Text content: {{ex.text_content}}
OUTPUT:
{"category": "{{ex.category}}", "evidence": "{{ex.evidence}}"}

{% endfor %}

For a given URL : {{url}}, and text content: {{text_content}}.
Classify above url to complete the category and indicate evidence.
OUTPUT:
```

## Define Python node interface

Python node might have multiple inputs and outputs. Define inputs and outputs as shown below. If you have multiple outputs, remember to make it a dictionary so that the downstream node can call each key separately. For example:

```
import json
from promptflow import tool

@tool
def convert_to_dict(input_str: str, input_str2: str) -> dict:
 try:
 print(input_str2)
 return json.loads(input_str)
 except Exception as e:
 print("input is not valid, error: {}".format(e))
 return {"category": "None", "evidence": "None"}
```

## Link nodes together

After the interface is defined, you can use:

- \$ to link with flow input.
- \$ to link with single-output upstream node.
- \$ to link with multi-output upstream node.

Below are common scenarios for linking nodes together.

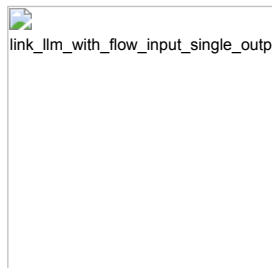
## Scenario 1 - Link LLM node with flow input and single-output upstream node

After you add a new LLM node and edit the prompt file like [Define LLM node interface](#), three inputs called `url`, `examples` and `text_content` are created in inputs section.

:sync: CLI You can link the LLM node input with flow input by `${inputs.url}`. And you can link `examples` to the upstream `prepare_examples` node and `text_content` to the `summarize_text_content` node by `${prepare_examples.output}` and `${summarize_text_content.output}`.

```
- name: classify_with_llm
 type: llm
 source:
 type: code
 path: classify_with_llm.jinja2
 inputs:
 deployment_name: text-davinci-003
 suffix: ""
 max_tokens: 128
 temperature: 0.2
 top_p: 1
 echo: false
 presence_penalty: 0
 frequency_penalty: 0
 best_of: 1
 url: ${inputs.url} # Link with flow input
 examples: ${prepare_examples.output} # Link LLM node with single-output upstream node
 text_content: ${summarize_text_content.output} # Link LLM node with single-output upstream node
```

:sync: VS Code Extension In the value drop-down, select `${inputs.url}`, `${prepare_examples.output}` and `${summarize_text_content.output}`, then you'll see in the graph view that the newly created LLM node is linked to the flow input, upstream `prepare_examples` and `summarize_text_content` node.



When running the flow, the `url` input of the node will be replaced by flow input on the fly, and the `examples` and `text_content` input of the node will be replaced by `prepare_examples` and `summarize_text_content` node output on the fly.

## Scenario 2 - Link LLM node with multi-output upstream node

Suppose we want to link the newly created LLM node with `convert_to_dict` Python node whose output is a dictionary with two keys: `category` and `evidence`.

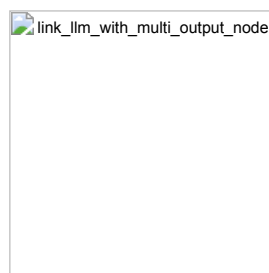
:sync: CLI You can link `examples` to the `evidence` output of upstream `convert_to_dict` node by `${convert_to_dict.output.evidence}` like below:

```

- name: classify_with_llm
 type: llm
 source:
 type: code
 path: classify_with_llm.jinja2
 inputs:
 deployment_name: text-davinci-003
 suffix: ""
 max_tokens: 128
 temperature: 0.2
 top_p: 1
 echo: false
 presence_penalty: 0
 frequency_penalty: 0
 best_of: 1
 text_content: ${convert_to_dict.output.evidence} # Link LLM node with multi-output upstream node

```

:sync: VS Code Extension In the value drop-down, select `${convert_to_dict.output}`, then manually append evidence, then you'll see in the graph view that the newly created LLM node is linked to the upstream `convert_to_dict` node.



When running the flow, the `text_content` input of the node will be replaced by evidence value from `convert_to_dict` node output dictionary on the fly.

## Scenario 3 - Link Python node with upstream node/flow input

After you add a new Python node and edit the code file like [Define Python node interface](#), two inputs called `input_str` and `input_str2` are created in inputs section. The linkage is the same as LLM node, using `${flow.input_name}` to link with flow input or `${upstream_node_name.output}` to link with upstream node.

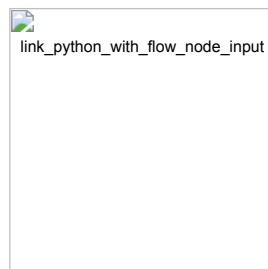
:sync: CLI

```

- name: prepare_examples
 type: python
 source:
 type: code
 path: prepare_examples.py
 inputs:
 input_str: ${inputs.url} # Link Python node with flow input
 input_str2: ${fetch_text_content_from_url.output} # Link Python node with single-output upstream node

```

:sync: VS Code Extension



When running the flow, the `input_str` input of the node will be replaced by flow input on the fly and the `input_str2` input of the node will be replaced by `fetch_text_content_from_url` node output dictionary on the fly.

## Set flow output

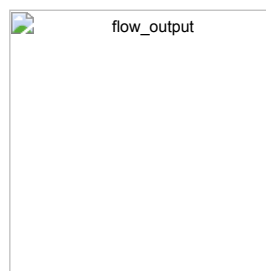
When the flow is complicated, instead of checking outputs on each node, you can set flow output and check outputs of multiple nodes in one place. Moreover, flow output helps:

- Check bulk test results in one single table.
- Define evaluation interface mapping.
- Set deployment response schema.

:sync: CLI You can add flow outputs in outputs section of flow yaml . The linkage is the same as LLM node, using `${convert_to_dict.output.category}` to link category flow output with with category value of upstream node `convert_to_dict`.

```
outputs:
 category:
 type: string
 reference: ${convert_to_dict.output.category}
 evidence:
 type: string
 reference: ${convert_to_dict.output.evidence}
```

:sync: VS Code Extension First define flow output schema, then select in drop-down the node whose output you want to set as flow output. Since `convert_to_dict` has a dictionary output with two keys: `category` and `evidence`, you need to manually append `category` and `evidence` to each. Then run flow, after a while, you can check flow output in a table.



filepath: promptflow/docs/how-to-guides/develop-a-flow/develop-evaluation-flow.md content: # Develop evaluation flow

This is an experimental feature, and may change at any time. Learn [more \(./faq.md#stable-vs-experimental\)](#).

The evaluation flow is a flow to test/evaluate the quality of your LLM application (standard/chat flow). It usually runs on the outputs of standard/chat flow, and compute key metrics that can be used to determine whether the standard/chat flow performs well. See [Flows \(./concepts/concept-flows.md\)](#) for more information.

Before proceeding with this document, it is important to have a good understanding of the standard flow. Please make sure you have read [Develop standard flow \(./develop-standard-flow.md\)](#), since they share many common features and these features won't be repeated in this doc, such as:

- Inputs/Outputs definition
- Nodes
- Chain nodes in a flow

While the evaluation flow shares similarities with the standard flow, there are some important differences that set it apart. The main distinctions are as follows:

- Inputs from an existing run: The evaluation flow contains inputs that are derived from the outputs of the standard/chat flow. These inputs are used for evaluation purposes.
- Aggregation node: The evaluation flow contains one or more aggregation nodes, where the actual evaluation takes place. These nodes are responsible for computing metrics and determining the performance of the standard/chat flow.

## Evaluation flow example

In this guide, we use [eval-classification-accuracy \(https://github.com/microsoft/promptflow/tree/main/examples/flows/evaluation/eval-classification-accuracy\)](https://github.com/microsoft/promptflow/tree/main/examples/flows/evaluation/eval-classification-accuracy) flow as an example of the evaluation flow. This is a flow illustrating how to evaluate the performance of a classification flow. It involves comparing each prediction to the groundtruth and assigns a `Correct` or `Incorrect` grade, and aggregating the results to produce metrics such as `accuracy`, which reflects how good the system is at classifying the data.

## Flow inputs

The flow `eval-classification-accuracy` contains two inputs:

```
inputs:
 groundtruth:
 type: string
 description: Groundtruth of the original question, it's the correct label that you hope your standard flow could predict.
 default: APP
 prediction:
 type: string
 description: The actual predicted outputs that your flow produces.
 default: APP
```

As evident from the inputs description, the evaluation flow requires two specific inputs:

- **groundtruth:** This input represents the actual or expected values against which the performance of the standard/chat flow will be evaluated.
- **prediction:** The prediction input is derived from the outputs of another standard/chat flow. It contains the predicted values generated by the standard/chat flow, which will be compared to the groundtruth values during the evaluation process.

From the definition perspective, there is no difference compared with adding an input/output in a `standard/chat flow`. However when running an evaluation flow, you may need to specify the data source from both data file and flow run outputs. For more details please refer to [Run and evaluate a flow \(./run-and-evaluate-a-flow/index.md#evaluate-your-flow\)](#).

## Aggregation node

Before introducing the aggregation node, let's see what a regular node looks like, we use node `grade` in the example flow for instance:

```
- name: grade
 type: python
 source:
 type: code
 path: grade.py
 inputs:
 groundtruth: ${inputs.groundtruth}
 prediction: ${inputs.prediction}
```

It takes both `groundtruth` and `prediction` from the flow inputs, compare them in the source code to see if they match:

```
from promptflow import tool

@tool
def grade(groundtruth: str, prediction: str):
 return "Correct" if groundtruth.lower() == prediction.lower() else "Incorrect"
```

When it comes to an aggregation node, there are two key distinctions that set it apart from a regular node:

1. It has an attribute `aggregation` set to be `true`.

```
- name: calculate_accuracy
 type: python
 source:
 type: code
 path: calculate_accuracy.py
 inputs:
 grades: ${grade.output}
 aggregation: true # Add this attribute to make it an aggregation node
```

2. Its source code accepts a `List` type parameter which is a collection of the previous regular node's outputs.



```

from typing import List
from promptflow import log_metric, tool

@tool
def calculate_accuracy(grades: List[str]):
 result = []
 for index in range(len(grades)):
 grade = grades[index]
 result.append(grade)

 # calculate accuracy for each variant
 accuracy = round((result.count("Correct") / len(result)), 2)
 log_metric("accuracy", accuracy)

 return result

```

The parameter `grades` in above function, contains all results that are produced by the regular node `grade`. Assuming the referred standard flow run has 3 outputs:

```

{"prediction": "App"}
{"prediction": "Channel"}
{"prediction": "Academic"}

```

And we provides a data file like this:

```

{"groundtruth": "App"}
{"groundtruth": "Channel"}
{"groundtruth": "Wiki"}

```

Then the `grades` value would be `["Correct", "Correct", "Incorrect"]`, and the final accuracy is 0.67.

This example provides a straightforward demonstration of how to evaluate the classification flow. Once you have a solid understanding of the evaluation mechanism, you can customize and design your own evaluation method to suit your specific needs.

## More about the list parameter

What if the number of referred standard flow run outputs does not match the provided data file? We know that a standard flow can be executed against multiple line data and some of them could fail while others succeed. Consider the same standard flow run mentioned in above example but the 2nd line run has failed, thus we have below run outputs:

```

{"prediction": "App"}
{"prediction": "Academic"}

```

The promptflow flow executor has the capability to recognize the index of the referred run's outputs and extract the corresponding data from the provided data file. This means that during the execution process, even if the same data file is provided(3 lines), only the specific data mentioned below will be processed:

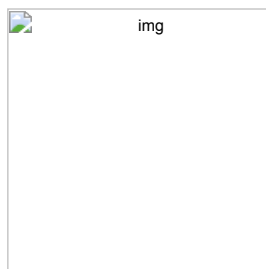
```

{"groundtruth": "App"}
{"groundtruth": "Wiki"}

```

In this case, the `grades` value would be `["Correct", "Incorrect"]` and the accuracy is 0.5.

## How to set aggregation node in VS Code Extention



## How to log metrics

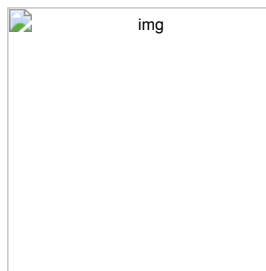
You can only log metrics in an aggregation node, otherwise the metric will be ignored.

Promptflow supports logging and tracking experiments using `log_metric` function. A metric is a key-value pair that records a single float measure. In a python node, you can log a metric with below code:

```
from typing import List
from promptflow import log_metric, tool

@tool
def example_log_metrics(grades: List[str]):
 # this node is an aggregation node so it accepts a list of grades
 metric_key = "accuracy"
 metric_value = round((grades.count("Correct") / len(result)), 2)
 log_metric(metric_key, metric_value)
```

After the run is completed, you can run `pf run show-metrics -n <run_name>` to see the metrics.



filepath: `promptflow/docs/how-to-guides/develop-a-flow/referencing-external-files-or-folders-in-a-flow.md` content: `# Referencing external files/folders in a flow`

Sometimes, pre-existing code assets are essential for the flow reference. In most cases, you can accomplish this by importing a Python package into your flow. However, if a Python package is not available or it is heavy to create a package, you can still reference external files or folders located outside of the current flow folder by using our **additional includes** feature in your flow configuration.

This feature provides an efficient mechanism to list relative file or folder paths that are outside of the flow folder, integrating them seamlessly into your `flow.dag.yaml`. For example:

```
additional_includes:
- ../web-classification/classify_with_llm.jinja2
- ../web-classification/convert_to_dict.py
- ../web-classification/fetch_text_content_from_url.py
- ../web-classification/prepare_examples.py
- ../web-classification/summarize_text_content.jinja2
- ../web-classification/summarize_text_content__variant_1.jinja2
```

You can add this field `additional_includes` into the `flow.dag.yaml`. The value of this field is a list of the **relative file/folder path** to the flow folder.

Just as with the common definition of the tool node entry, you can define the tool node entry in the `flow.dag.yaml` using only the file name, eliminating the need to specify the relative path again. For example:

```
nodes:
- name: fetch_text_content_from_url
 type: python
 source:
 type: code
 path: fetch_text_content_from_url.py
 inputs:
 url: ${inputs.url}
- name: summarize_text_content
 use_variants: true
- name: prepare_examples
 type: python
 source:
 type: code
 path: prepare_examples.py
 inputs: {}
```

The entry file "fetch\_text\_content\_from\_url.py" of the tool node "fetch\_text\_content\_from\_url" is located in "../web-classification/fetch\_text\_content\_from\_url.py", as specified in the additional\_includes field. The same applies to the "summarize\_text\_content" tool nodes.

**Note:**

1. If you have two files with the same name located in different folders specified in the additional\_includes field, and the file name is also specified as the entry of a tool node, the system will reference the **last one** it encounters in the additional\_includes field.

1. If you have a file in the flow folder with the same name as a file specified in the additional\_includes field, the system will prioritize the file listed in the additional\_includes field. Take the following YAML structure as an example:

```
additional_includes:
- ../web-classification/prepare_examples.py
- ../tmp/prepare_examples.py
...
nodes:
- name: summarize_text_content
 use_variants: true
- name: prepare_examples
 type: python
 source:
 type: code
 path: prepare_examples.py
 inputs: {}
```

In this case, the system will use "../tmp/prepare\_examples.py" as the entry file for the tool node "prepare\_examples". Even if there is a file named "prepare\_examples.py" in the flow folder, the system will still use the file "../tmp/prepare\_examples.py" specified in the additional\_includes field.

*Tips: The additional includes feature can significantly streamline your workflow by eliminating the need to manually handle these references.*

1. To get a hands-on experience with this feature, practice with our sample [flow-with-additional-includes](https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/flow-with-additional-includes) (<https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/flow-with-additional-includes>).
2. You can learn more about [How the 'additional includes' flow operates during the transition to the cloud](#) ([./cloud/azureai/quick-start/index.md#run-snapshot-of-the-flow-with-additional-includes](#)).

filepath: promptflow/docs/how-to-guides/deploy-a-flow/index.md content: # Deploy a flow A flow can be deployed to multiple platforms, such as a local development service, Docker container, Kubernetes cluster, etc.

```
:grid-columns: 1 2 2 3
- image: ../../media/how-to-guides/local.png
 content: "<center>Development server</center>"
 website: deploy-using-dev-server.html

- image: ../../media/how-to-guides/docker.png
 content: "<center>Docker</center>"
 website: deploy-using-docker.html

- image: ../../media/how-to-guides/kubernetes.png
 content: "<center>Kubernetes</center>"
 website: deploy-using-kubernetes.html
```

We also provide guides to deploy to cloud, such as azure app service:

```
:grid-columns: 1 2 2 3

- image: ../../media/how-to-guides/appservice.png
 content: "<center>Azure App Service</center>"
 website: ../../cloud/azureai/deploy-to-azure-appservice.html
```

We are working on more official deployment guides for other hosting providers, and welcome user submitted guides.

```
:maxdepth: 1
:hidden:

deploy-using-dev-server
deploy-using-docker
deploy-using-kubernetes
distribute-flow-as-executable-app
```

filepath: promptflow/docs/how-to-guides/deploy-a-flow/deploy-using-dev-server.md content: # Deploy a flow using development server

This is an experimental feature, and may change at any time. Learn [more \(../faq.md#stable-vs-experimental\)](#).

Once you have created and thoroughly tested a flow, you can use it as an HTTP endpoint.

:sync: CLI We are going to use the [web-classification \(https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification/\)](https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification/) as an example to show how to deploy a flow.

Please ensure you have [create the connection \(../manage-connections.md#create-a-connection\)](#) required by flow, if not, you could refer to [Setup connection for web-classification \(https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification/\)](https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification/).

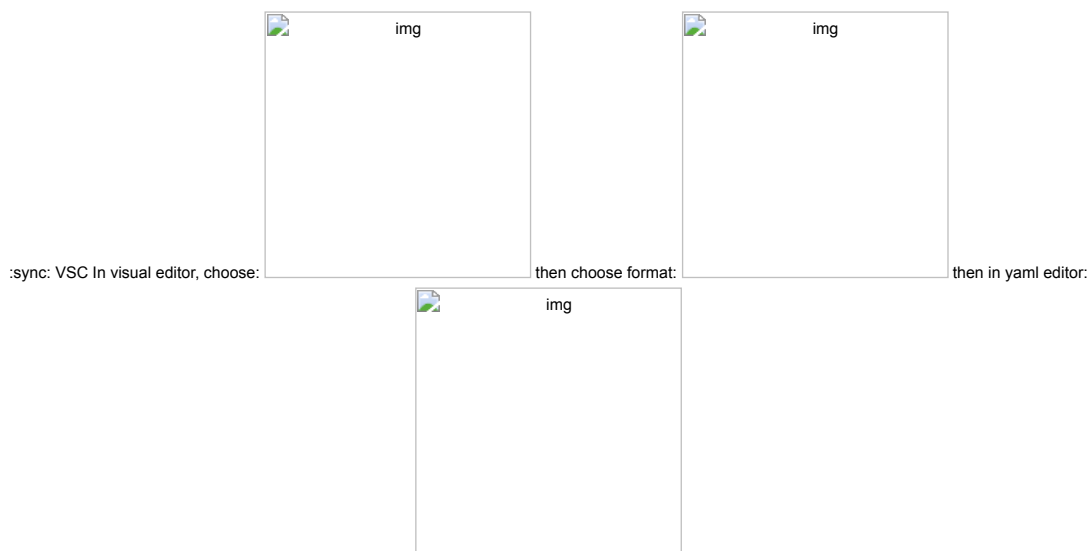
Note: We will use relevant environment variable ( ) to override connection configurations in serving mode, white space in connection name will be removed directly from environment variable name. For instance, if there is a custom connection named 'custom\_connection' with a configuration key called 'chat\_deployment\_name,' the function will attempt to retrieve 'chat\_deployment\_name' from the environment variable 'CUSTOM\_CONNECTION\_CHAT\_DEPLOYMENT\_NAME' by default. If the environment variable is not set, it will use the original value as a fallback.

The following CLI commands allows you serve a flow folder as an endpoint. By running this command, a [flask \(https://flask.palletsprojects.com/en/\)](https://flask.palletsprojects.com/en/) app will start in the environment where command is executed, please ensure all prerequisites required by flow have been installed.

```
Serve the flow at localhost:8080
pf flow serve --source <path-to-your-flow-folder> --port 8080 --host localhost
```

The expected result is as follows if the flow served successfully, and the process will keep alive until it be killed manually.





## Test endpoint

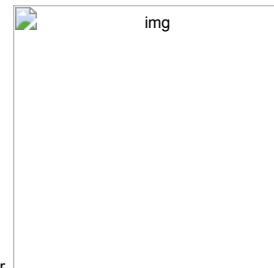
You could open another terminal to test the endpoint with the following command:

```
curl http://localhost:8080/score --data '{"url":"https://play.google.com/store/apps/details?id=com.twitter.android"}' -X POST -H "Content-Type: application/json"
```

You could open another terminal to test the endpoint with the following command:

```
Invoke-WebRequest -URI http://localhost:8080/score -Body '{"url":"https://play.google.com/store/apps/details?id=com.twitter.android"}'
```

The development server has a built-in web page you can use to test the flow. Open 'http://localhost:8080' in your browser.



## Next steps

- Try the example [here \(https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification/\)](https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification/).
- See how to [deploy a flow using docker \(deploy-using-docker.md\)](#).
- See how to [deploy a flow using kubernetes \(deploy-using-kubernetes.md\)](#).

filepath: promptflow/docs/how-to-guides/deploy-a-flow/deploy-using-kubernetes.md content: # Deploy a flow using Kubernetes

This is an experimental feature, and may change at any time. Learn [more \(../faq.md#stable-vs-experimental\)](#).

There are four steps to deploy a flow using Kubernetes:

1. Build the flow as docker format.
2. Build the docker image.
3. Create Kubernetes deployment yaml.
4. Apply the deployment.

## Build a flow as docker format

:sync: CLI

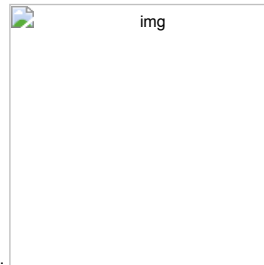
Note that all dependent connections must be created before building as docker.

```
create connection if not created before
pf connection create --file ../../examples/connections/azure_openai.yml --set api_key=<your_api_key> api_base=<your_api_base> --name
```

Use the command below to build a flow as docker format:

```
pf flow build --source <path-to-your-flow-folder> --output <your-output-dir> --format docker
```

:sync: VSC



Click the button below to build a flow as docker format:

Note that all dependent connections must be created before exporting as docker.

## Docker format folder structure

Exported Dockerfile & its dependencies are located in the same folder. The structure is as below:

- flow: the folder contains all the flow files
  - ...
- connections: the folder contains yaml files to create all related connections
  - ...
- Dockerfile: the dockerfile to build the image
- start.sh: the script used in CMD of Dockerfile to start the service
- runit: the folder contains all the runit scripts
  - ...
- settings.json: a json file to store the settings of the docker image
- README.md: Simple introduction of the files

## Deploy with Kubernetes

We are going to use the [web-classification](https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification/) (<https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification/>), as an example to show how to deploy with Kubernetes.

Please ensure you have [create the connection](#) ([./manage-connections.md#create-a-connection](#)) required by flow, if not, you could refer to [Setup connection for web-classification](#) (<https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification/>).

Additionally, please ensure that you have installed all the required dependencies. You can refer to the "Prerequisites" section in the README of the [web-classification](https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification/) (<https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification/>) for a comprehensive list of prerequisites and installation instructions.

## Build Docker image

Like other Dockerfile, you need to build the image first. You can tag the image with any name you want. In this example, we use `web-classification-serve`.

Then run the command below:

```
cd <your-output-dir>
docker build . -t web-classification-serve
```

## Create Kubernetes deployment yaml.

The Kubernetes deployment yaml file acts as a guide for managing your docker container in a Kubernetes pod. It clearly specifies important information like the container image, port configurations, environment variables, and various settings. Below, you'll find a simple deployment template that you can easily customize to meet your needs.

**Note:** You need encode the secret using base64 firstly and input the `<encoded_secret>` as 'open-ai-connection-api-key' in the deployment configuration. For example, you can run below commands in linux:

```
encoded_secret=$(echo -n <your_api_key> | base64)
```

```

kind: Namespace
apiVersion: v1
metadata:
 name: <your-namespace>

apiVersion: v1
kind: Secret
metadata:
 name: open-ai-connection-api-key
 namespace: <your-namespace>
type: Opaque
data:
 open-ai-connection-api-key: <encoded_secret>

apiVersion: v1
kind: Service
metadata:
 name: web-classification-service
 namespace: <your-namespace>
spec:
 type: NodePort
 ports:
 - name: http
 port: 8080
 targetPort: 8080
 nodePort: 30123
 selector:
 app: web-classification-serve-app

apiVersion: apps/v1
kind: Deployment
metadata:
 name: web-classification-serve-app
 namespace: <your-namespace>
spec:
 selector:
 matchLabels:
 app: web-classification-serve-app
 template:
 metadata:
 labels:
 app: web-classification-serve-app
 spec:
 containers:
 - name: web-classification-serve-container
 image: <your-docker-image>
 imagePullPolicy: Never
 ports:
 - containerPort: 8080
 env:
 - name: OPEN_AI_CONNECTION_API_KEY
 valueFrom:
 secretKeyRef:
 name: open-ai-connection-api-key
 key: open-ai-connection-api-key
```

## Apply the deployment.

Before you can deploy your application, ensure that you have set up a Kubernetes cluster and installed [kubectl](https://kubernetes.io/docs/reference/kubectl/) (<https://kubernetes.io/docs/reference/kubectl/>) if it's not already installed. In this documentation, we will use [Minikube](https://minikube.sigs.k8s.io/docs/) (<https://minikube.sigs.k8s.io/docs/>) as an example. To start the cluster, execute the following command:

```
minikube start
```

Once your Kubernetes cluster is up and running, you can proceed to deploy your application by using the following command:

```
kubectl apply -f deployment.yaml
```

This command will create the necessary pods to run your application within the cluster.

**Note:** You need replace <pod\_name> below with your specific pod\_name. You can retrieve it by running `kubectl get pods -n web-classification`.

## Retrieve flow service logs of the container

The `kubectl logs` command is used to retrieve the logs of a container running within a pod, which can be useful for debugging, monitoring, and troubleshooting applications deployed in a Kubernetes cluster.

```
kubectl -n <your-namespace> logs <pod-name>
```

## Connections

If the service involves connections, all related connections will be exported as yaml files and recreated in containers. Secrets in connections won't be exported directly. Instead, we will export them as a reference to environment variables:

```
$schema: https://azuremlschemas.azureedge.net/promptflow/latest/OpenAIConnection.schema.json
type: open_ai
name: open_ai_connection
module: promptflow.connections
api_key: ${env:OPEN_AI_CONNECTION_API_KEY} # env reference
```

You'll need to set up the environment variables in the container to make the connections work.

## Test the endpoint

- Option1:

Once you've started the service, you can establish a connection between a local port and a port on the pod. This allows you to conveniently test the endpoint from your local terminal. To achieve this, execute the following command:

```
kubectl port-forward <pod_name> <local_port>:<container_port> -n <your-namespace>
```

With the port forwarding in place, you can use the `curl` command to initiate the endpoint test:

```
curl http://localhost:<local_port>/score --data '{"url":"https://play.google.com/store/apps/details?id=com.twitter.android"}'
```

- Option2:

`minikube service web-classification-service --url -n <your-namespace>` runs as a process, creating a tunnel to the cluster. The command exposes the service directly to any program running on the host operating system.

The command above will retrieve the URL of a service running within a Minikube Kubernetes cluster (e.g. `http://<assigned_port>`), which you can click to interact with the flow service in your web browser. Alternatively, you can use the following command to test the endpoint:

**Note:** Minikube will use its own external port instead of nodePort to listen to the service. So please substitute <assigned\_port> with the port obtained above.

```
curl http://localhost:<assigned_port>/score --data '{"url":"https://play.google.com/store/apps/details?id=com.twitter.android"}'
```

## Next steps

- Try the example [here](https://github.com/microsoft/promptflow/tree/main/examples/tutorials/flow-deploy/kubernetes) (<https://github.com/microsoft/promptflow/tree/main/examples/tutorials/flow-deploy/kubernetes>).

filepath: promptflow/docs/how-to-guides/deploy-a-flow/distribute-flow-as-executable-app.md content: # Distribute flow as executable app

This is an experimental feature, and may change at any time. Learn [more](#) ([../faq.md#stable-vs-experimental](#)).



We are going to use the [web-classification](https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification/) (<https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification/>) as an example to show how to distribute flow as executable app with [Pyinstaller](https://pyinstaller.org/en/stable/requirements.html#) (<https://pyinstaller.org/en/stable/requirements.html#>).

Please ensure that you have installed all the required dependencies. You can refer to the "Prerequisites" section in the README of the [web-classification](https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification/) (<https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification/>) for a comprehensive list of prerequisites and installation instructions. And we recommend you to add a `requirements.txt` to indicate all the required dependencies for each flow.

[Pyinstaller](https://pyinstaller.org/en/stable/installation.html) (<https://pyinstaller.org/en/stable/installation.html>) is a popular tool used for converting Python applications into standalone executables. It allows you to package your Python scripts into a single executable file, which can be run on a target machine without requiring the Python interpreter to be installed. [Streamlit](https://docs.streamlit.io/library/get-started) (<https://docs.streamlit.io/library/get-started>) is an open-source Python library used for creating web applications quickly and easily. It's designed for data scientists and engineers who want to turn data scripts into shareable web apps with minimal effort. We use Pyinstaller to package the flow and Streamlit to create custom web apps. Prior to distributing the workflow, kindly ensure that you have installed them.

## Build a flow as executable format

Note that all dependent connections must be created before building as executable.

```
create connection if not created before
pf connection create --file ../../examples/connections/azure_openai.yml --set api_key=<your_api_key> api_base=<your_api_base> --no
```

Use the command below to build a flow as executable format:

```
pf flow build --source <path-to-your-flow-folder> --output <your-output-dir> --format executable
```

## Executable format folder structure

Exported files & its dependencies are located in the same folder. The structure is as below:

- flow: the folder contains all the flow files.
- connections: the folder contains yaml files to create all related connections.
- app.py: the entry file is included as the entry point for the bundled application.
- app.spec: the spec file tells PyInstaller how to process your script.
- main.py: it will start streamlit service and be called by the entry file.
- settings.json: a json file to store the settings of the executable application.
- build: a folder contains various log and working files.
- dist: a folder contains the executable application.
- README.md: Simple introduction of the files.

## A template script of the entry file

PyInstaller reads a spec file or Python script written by you. It analyzes your code to discover every other module and library your script needs in order to execute. Then it collects copies of all those files, including the active Python interpreter, and puts them with your script in a single folder, or optionally in a single executable file.

We provide a Python entry script named `app.py` as the entry point for the bundled app, which enables you to serve a flow folder as an endpoint.

```

import os
import sys

from promptflow_cli_pf_connection import create_connection
from streamlit.web import cli as st_cli
from streamlit.runtime import exists

from main import start

def is_yaml_file(file_path):
 _, file_extension = os.path.splitext(file_path)
 return file_extension.lower() in ('.yaml', '.yml')

def create_connections(directory_path) -> None:
 for root, dirs, files in os.walk(directory_path):
 for file in files:
 file_path = os.path.join(root, file)
 if is_yaml_file(file_path):
 create_connection(file_path)

if __name__ == "__main__":
 create_connections(os.path.join(os.path.dirname(__file__), "connections"))
 if exists():
 start()
 else:
 main_script = os.path.join(os.path.dirname(__file__), "main.py")
 sys.argv = ["streamlit", "run", main_script, "--global.developmentMode=false"]
 st_cli.main(prog_name="streamlit")

```

## A template script of the spec file

The spec file tells PyInstaller how to process your script. It encodes the script names and most of the options you give to the pyinstaller command. The spec file is actually executable Python code. PyInstaller builds the app by executing the contents of the spec file.

To streamline this process, we offer a `app.spec` spec file that bundles the application into a single file. For additional information on spec files, you can refer to the [Using Spec Files \(https://pyinstaller.org/en/stable/spec-files.html\)](https://pyinstaller.org/en/stable/spec-files.html). Please replace `streamlit_runtime_interpreter_path` with the path of streamlit runtime interpreter in your environment.

```

-*- mode: python ; coding: utf-8 -*-
from PyInstaller.utils.hooks import collect_data_files
from PyInstaller.utils.hooks import copy_metadata

datas = [('connections', 'connections'), ('flow', 'flow'), ('settings.json', '.'), ('main.py', '.'), ('{{streamlit_runtime_interpreter}}', 'streamlit_runtime_interpreter')]
datas += collect_data_files('streamlit')
datas += copy_metadata('streamlit')
datas += collect_data_files('keyrings.alt', include_py_files=True)
datas += copy_metadata('keyrings.alt')
datas += collect_data_files('streamlit_quill')

block_cipher = None

a = Analysis(
 ['app.py', 'main.py'],
 pathex=[],
 binaries=[],
 datas=datas,
 hiddenimports=['bs4'],
 hookspath=[],
 hooksconfig={},
 runtime_hooks=[],
 excludes=[],
 win_no_prefer_redirects=False,
 win_private_assemblies=False,
 cipher=block_cipher,
 noarchive=False,
)
pyz = PYZ(a.pure, a.zipped_data, cipher=block_cipher)

exe = EXE(
 pyz,
 a.scripts,
 a.binaries,
 a.zipfiles,
 a.datas,
 [],
 name='app',
 debug=False,
 bootloader_ignore_signals=False,
 strip=False,
 upx=True,
 upx_exclude=[],
 runtime_tmpdir=None,
 console=True,
 disable_windowed_traceback=False,
 argv_emulation=False,
 target_arch=None,
 codesign_identity=None,
 entitlements_file=None,
)

```

## The bundled application using Pyinstaller

Once you've build a flow as executable format following [Build a flow as executable format](#). It will create two folders named `build` and `dist` within your specified output directory, denoted as `.`. The `build` folder houses various log and working files, while the `dist` folder contains the `app` executable application.

## Connections

If the service involves connections, all related connections will be exported as yaml files and recreated in the executable package. Secrets in connections won't be exported directly. Instead, we will export them as a reference to environment variables:

```
$schema: https://azuremlschemas.azureedge.net/promptflow/latest/OpenAIConnection.schema.json
type: open_ai
name: open_ai_connection
module: promptflow.connections
api_key: ${env:OPEN_AI_CONNECTION_API_KEY} # env reference
```

## Test the endpoint

Finally, You can distribute the bundled application `app` to other people. They can execute your program by double clicking the executable file, e.g. `app.exe` in Windows system or running the binary file, e.g. `app` in Linux system.

The development server has a built-in web page they can use to test the flow by opening 'http://localhost:8501' in the browser. The expected result is as follows: if the flow served successfully, the process will keep alive until it is killed manually.

To your users, the app is self-contained. They do not need to install any particular version of Python or any modules. They do not need to have Python installed at all.

**Note:** The executable generated is not cross-platform. One platform (e.g. Windows) packaged executable can't run on others (Mac, Linux).

## Known issues

1. Note that Python 3.10.0 contains a bug making it unsupportable by PyInstaller. PyInstaller will also not work with beta releases of Python 3.13.

## Next steps

- Try the example [here](https://github.com/microsoft/promptflow/blob/main/examples/tutorials/flow-deploy) (https://github.com/microsoft/promptflow/blob/main/examples/tutorials/flow-deploy).

filepath: promptflow/docs/how-to-guides/deploy-a-flow/deploy-using-docker.md content: # Deploy a flow using Docker

This is an experimental feature, and may change at any time. Learn [more \(../faq.md#stable-vs-experimental\)](#).

There are two steps to deploy a flow using docker:

1. Build the flow as docker format.
2. Build and run the docker image.

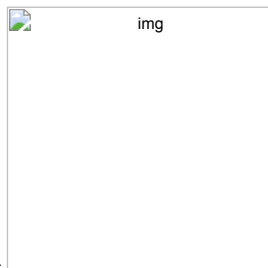
## Build a flow as docker format

:sync: CLI

Use the command below to build a flow as docker format:

```
pf flow build --source <path-to-your-flow-folder> --output <your-output-dir> --format docker
```

:sync: VSC



In visual editor, choose:

Click the button below to build a flow as docker format:



Note that all dependent connections must be created before exporting as docker.

## Docker format folder structure

Exported Dockerfile & its dependencies are located in the same folder. The structure is as below:

- flow: the folder contains all the flow files
  - ...
- connections: the folder contains yaml files to create all related connections
  - ...

- Dockerfile: the dockerfile to build the image
- start.sh: the script used in CMD of Dockerfile to start the service
- runit: the folder contains all the runit scripts
  - ...
- settings.json: a json file to store the settings of the docker image
- README.md: Simple introduction of the files

## Deploy with Docker

We are going to use the [web-classification](https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification/) (https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification/) as an example to show how to deploy with docker.

Please ensure you have [create the connection](https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification/) (./manage-connections.md#create-a-connection) required by flow, if not, you could refer to [Setup connection for web-classification](https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification/) (https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification/).

## Build a flow as docker format app

Use the command below to build a flow as docker format app:

```
pf flow build --source ../../flows/standard/web-classification --output dist --format docker
```

Note that all dependent connections must be created before exporting as docker.

## Build Docker image

Like other Dockerfile, you need to build the image first. You can tag the image with any name you want. In this example, we use web-classification-serve.

Run the command below to build image:

```
docker build dist -t web-classification-serve
```

## Run Docker image

Run the docker image will start a service to serve the flow inside the container.

### Connections

If the service involves connections, all related connections will be exported as yaml files and recreated in containers. Secrets in connections won't be exported directly. Instead, we will export them as a reference to environment variables:

```
$schema: https://azuremlschemas.azureedge.net/promptflow/latest/OpenAIConnection.schema.json
type: open_ai
name: open_ai_connection
module: promptflow.connections
api_key: ${env:OPEN_AI_CONNECTION_API_KEY} # env reference
```

You'll need to set up the environment variables in the container to make the connections work.

## Run with docker run

You can run the docker image directly set via below commands:

```
The started service will listen on port 8080.You can map the port to any port on the host machine as you want.
docker run -p 8080:8080 -e OPEN_AI_CONNECTION_API_KEY=<secret-value> web-classification-serve
```

## Test the endpoint

After start the service, you can use curl to test it:

```
curl http://localhost:8080/score --data '{"url":"https://play.google.com/store/apps/details?id=com.twitter.android"}' -X POST -H "C"
```

## Next steps

- Try the example [here \(https://github.com/microsoft/promptflow/blob/main/examples/tutorials/flow-deploy/docker\)](https://github.com/microsoft/promptflow/blob/main/examples/tutorials/flow-deploy/docker).
- See how to [deploy a flow using kubernetes \(deploy-using-kubernetes.md\)](#).

filepath: promptflow/docs/how-to-guides/run-and-evaluate-a-flow/use-column-mapping.md content: # Use column mapping

In this document, we will introduce how to map inputs with column mapping when running a flow.

## Column mapping introduction

Column mapping is a mapping from flow input name to specified values. If specified, the flow will be executed with provided value for specified inputs. The following types of values in column mapping are supported:

- `${data.<column_name>}` to reference from your test dataset.
- `${run.inputs.<input_name>}` to reference from referenced run's input. **Note:** this only supported when `--run` is provided for `pf run`.
- `${run.outputs.<output_name>}` to reference from referenced run's output. **Note:** this only supported when `--run` is provided for `pf run`.
- `STATIC_VALUE` to create static value for all lines for specified column.

## Flow inputs override priority

Flow input values are overridden according to the following priority:

"specified in column mapping" > "default value" > "same name column in provided data".

For example, if we have a flow with following inputs:

```
inputs:
 input1:
 type: string
 default: "default_val1"
 input2:
 type: string
 default: "default_val2"
 input3:
 type: string
 input4:
 type: string
 ...
```

And the flow will return each inputs in outputs.

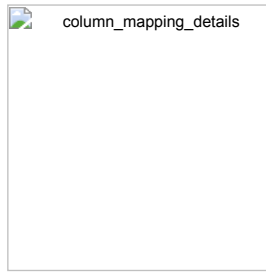
With the following data

```
{"input3": "val3_in_data", "input4": "val4_in_data"}
```

And use the following YAML to run

```
$schema: https://azuremlschemas.azureedge.net/promptflow/latest/Run.schema.json
flow: path/to/flow
my_flow has default value val2 for key2
data: path/to/data
my_data has column key3 with value val3
column_mapping:
 input1: "val1_in_column_mapping"
 input3: ${data.input3}
```

Since the flow will return each inputs in output, we can get the actual inputs from `outputs.output` field in run details:



- Input "input1" has value "val1\_in\_column\_mapping" since it's specified as constance in `column_mapping`.
- Input "input2" has value "default\_val2" since it used default value in flow dag.
- Input "input3" has value "val3\_in\_data" since it's specified as data reference in `column_mapping`.
- Input "input4" has value "val4\_in\_data" since it has same name column in provided data.

filepath: `promptflow/docs/how-to-guides/run-and-evaluate-a-flow/index.md` content: # Run and evaluate a flow

This is an experimental feature, and may change at any time. Learn [more \(../faq.md#stable-vs-experimental\)](#).

After you have developed and tested the flow in [init and test a flow \(../init-and-test-a-flow.md\)](#), this guide will help you learn how to run a flow with a larger dataset and then evaluate the flow you have created.

## Create a batch run

Since you have run your flow successfully with a small set of data, you might want to test if it performs well in large set of data, you can run a batch test and check the outputs.

A bulk test allows you to run your flow with a large dataset and generate outputs for each data row, and the run results will be recorded in local db so you can use [pf commands \(../reference/pf-command-reference.md\)](#) to view the run results at anytime. (e.g. `pf run list`)

Let's create a run with flow [web-classification \(https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification\)](https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification). It is a flow demonstrating multi-class classification with LLM. Given an url, it will classify the url into one web category with just a few shots, simple summarization and classification prompts.

To begin with the guide, you need:

- Git clone the sample repository(above flow link) and set the working directory to `<path-to-the-sample-repo>/examples/flows/.`
- Make sure you have already created the necessary connection following [Create necessary connections \(../quick-start.md#create-necessary-connections\)](#).

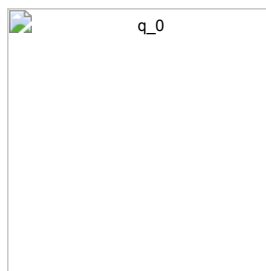
:sync: CLI

Create the run with flow and data, can add `--stream` to stream the run.

```
pf run create --flow standard/web-classification --data standard/web-classification/data.jsonl --column-mapping url='${data.url}' --
```

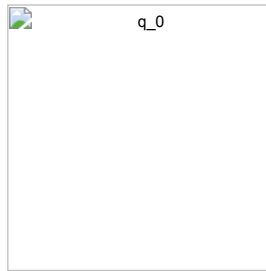
**Note** `column-mapping` is a mapping from flow input name to specified values, see more details in [Use column mapping \(https://aka.ms/pf/column-mapping\)](https://aka.ms/pf/column-mapping).

You can also name the run by specifying `--name my_first_run` in above command, otherwise the run name will be generated in a certain pattern which has timestamp inside.

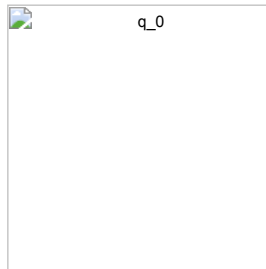


With a run name, you can easily view or visualize the run details using below commands:

```
pf run show-details -n my_first_run
```



```
pf run visualize -n my_first_run
```



More details can be found with `pf run --help`

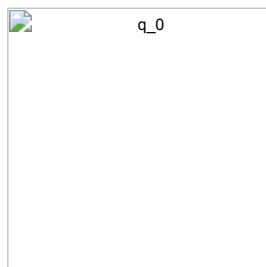
:sync: SDK

```
from promptflow import PFClient

Please protect the entry point by using `if __name__ == '__main__':`,
otherwise it would cause unintended side effect when promptflow spawn worker processes.
Ref: https://docs.python.org/3/library/multiprocessing.html#the-spawn-and-forkserver-start-methods
if __name__ == "__main__":
 # PFClient can help manage your runs and connections.
 pf = PFClient()

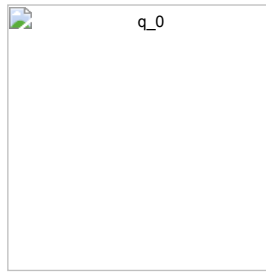
 # Set flow path and run input data
 flow = "standard/web-classification" # set the flow directory
 data = "standard/web-classification/data.jsonl" # set the data file

 # create a run, stream it until it's finished
 base_run = pf.run(
 flow=flow,
 data=data,
 stream=True,
 # map the url field from the data to the url input of the flow
 column_mapping={"url": "${data.url}"},
)
```

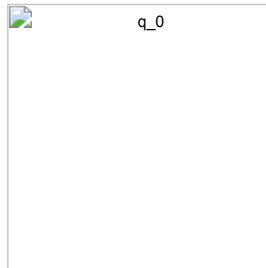


```
get the inputs/outputs details of a finished run.
details = pf.get_details(base_run)
details.head(10)
```

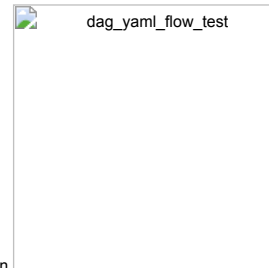




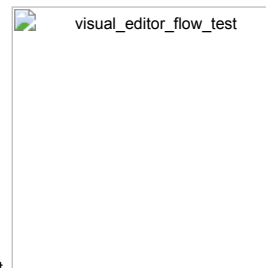
```
visualize the run in a web browser
pf.visualize(base_run)
```



Feel free to check [Promptflow Python Library Reference](#) ([./reference/python-library-reference/promptflow.md](#)) for all SDK public interfaces.



:sync: VS Code Extension Use the code lens action on the top of the yaml editor to trigger batch run



Click the bulk test button on the top of the visual editor to trigger flow test.

We also have a more detailed documentation [Manage runs](#) ([./manage-runs.md](#)) demonstrating how to manage your finished runs with CLI, SDK and VS Code Extension.

## Evaluate your flow

You can use an evaluation method to evaluate your flow. The evaluation methods are also flows which use Python or LLM etc., to calculate metrics like accuracy, relevance score.

Please refer to [Develop evaluation flow](#) ([./develop-a-flow/develop-evaluation-flow.md](#)) to learn how to develop an evaluation flow.

In this guide, we use [eval-classification-accuracy](#) (<https://github.com/microsoft/promptflow/tree/main/examples/flows/evaluation/eval-classification-accuracy>) flow to evaluate. This is a flow illustrating how to evaluate the performance of a classification system. It involves comparing each prediction to the groundtruth and assigns a `Correct` or `Incorrect` grade, and aggregating the results to produce metrics such as `accuracy`, which reflects how good the system is at classifying the data.

## Run evaluation flow against run

:sync: CLI

### Evaluate the finished flow run

After the run is finished, you can evaluate the run with below command, compared with the normal run create command, note there are two extra arguments:

- `column-mapping`: A mapping from flow input name to specified data values. Reference [here](#) (<https://aka.ms/pf/column-mapping>) for detailed information.
- `run`: The run name of the flow run to be evaluated.

More details can be found in [Use column mapping \(https://aka.ms/pf/column-mapping\)](https://aka.ms/pf/column-mapping).

```
pf run create --flow evaluation/eval-classification-accuracy --data standard/web-classification/data.jsonl --column-mapping groundtruth
```

Same as the previous run, you can specify the evaluation run name with `--name my_first_eval_run` in above command.

You can also stream or view the run details with:

```
pf run stream -n my_first_eval_run # same as "--stream" in command "run create"
pf run show-details -n my_first_eval_run
pf run show-metrics -n my_first_eval_run
```

Since now you have two different runs `my_first_run` and `my_first_eval_run`, you can visualize the two runs at the same time with below command.

```
pf run visualize -n "my_first_run,my_first_eval_run"
```

A web browser will be opened to show the visualization result.



:sync: SDK

#### Evaluate the finished flow run

After the run is finished, you can evaluate the run with below command, compared with the normal run create command, note there are two extra arguments:

- `column-mapping`: A dictionary represents sources of the input data that are needed for the evaluation method. The sources can be from the flow run output or from your test dataset.
  - If the data column is in your test dataset, then it is specified as `${data.<column_name>}`.
  - If the data column is from your flow output, then it is specified as `${run.outputs.<output_name>}`.
- `run`: The run name or run instance of the flow run to be evaluated.

More details can be found in [Use column mapping \(https://aka.ms/pf/column-mapping\)](https://aka.ms/pf/column-mapping).

```

from promptflow import PFClient

PFClient can help manage your runs and connections.
pf = PFClient()

set eval flow path
eval_flow = "evaluation/eval-classification-accuracy"
data= "standard/web-classification/data.jsonl"

run the flow with existing run
eval_run = pf.run(
 flow=eval_flow,
 data=data,
 run=base_run,
 column_mapping={ # map the url field from the data to the url input of the flow
 "groundtruth": "${data.answer}",
 "prediction": "${run.outputs.category}",
 }
)

stream the run until it's finished
pf.stream(eval_run)

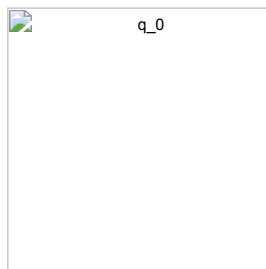
get the inputs/outputs details of a finished run.
details = pf.get_details(eval_run)
details.head(10)

view the metrics of the eval run
metrics = pf.get_metrics(eval_run)
print(json.dumps(metrics, indent=4))

visualize both the base run and the eval run
pf.visualize([base_run, eval_run])

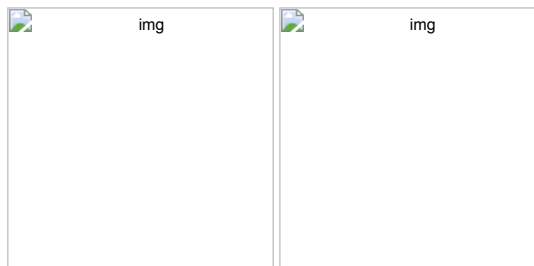
```

A web browser will be opened to show the visualization result.



:sync: VS Code Extension

There are actions to trigger local batch runs. To perform an evaluation you can use the run against "existing runs" actions.



## Next steps

Learn more about:

- [Tune prompts with variants \(./tune-prompts-with-variants.md\)](#)

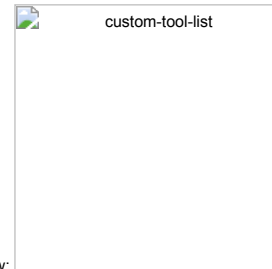
- [Deploy a flow \(../deploy-a-flow/index.md\)](#)
- [Manage runs \(../manage-runs.md\)](#)
- [Python library reference \(../reference/python-library-reference/promptflow.md\)](#)

```
:maxdepth: 1
:hidden:

use-column-mapping
```

filepath: promptflow/docs/how-to-guides/develop-a-tool/create-and-use-tool-package.md content: # Create and Use Tool Package In this document, we will guide you through the process of developing your own tool package, offering detailed steps and advice on how to utilize your creation.

The custom tool is the prompt flow tool developed by yourself. If you find it useful, you can follow this guidance to make it a tool package. This will enable you to conveniently reuse it, share it with your team, or distribute it to anyone in the world.



After successful installation of the package, your custom "tool" will show up in VSCode extension as below:

## Create your own tool package

Your tool package should be a python package. To try it quickly, just use [my-tools-package 0.0.1](https://pypi.org/project/my-tools-package/) (<https://pypi.org/project/my-tools-package/>), and skip this section.

### Prerequisites

Create a new conda environment using python 3.9 or 3.10. Run below command to install PromptFlow dependencies:

```
pip install promptflow
```

Install Pytest packages for running tests:

```
pip install pytest pytest-mock
```

Clone the PromptFlow repository from GitHub using the following command:

```
git clone https://github.com/microsoft/promptflow.git
```

### Create custom tool package

Run below command under the root folder to create your tool project quickly:

```
python <promptflow github repo>\scripts\tool\generate_tool_package_template.py --destination <your-tool-project> --package-name <your-package-name>
```

For example:

```
python D:\proj\github\promptflow\scripts\tool\generate_tool_package_template.py --destination hello-world-proj --package-name hello-world
```

This auto-generated script will create one tool for you. The parameters *destination* and *package-name* are mandatory. The parameters *tool-name* and *function-name* are optional. If left unfilled, the *tool-name* will default to *hello\_world\_tool*, and the *function-name* will default to *tool-name*.

The command will generate the tool project as follows with one tool `hello_world_tool.py` in it:

```

hello-world-proj/
|
├─ hello_world/
| | ── tools/
| | | ── __init__.py
| | | ── hello_world_tool.py
| | | ── utils.py
| | ── yamls/
| | | ── hello_world_tool.yaml
| | ── __init__.py
|
├─ tests/
| | ── __init__.py
| | ── test_hello_world_tool.py
|
├─ MANIFEST.in
|
└─ setup.py

```

The points outlined below explain the purpose of each folder/file in the package. If your aim is to develop multiple tools within your package, please make sure to closely examine point 2 and 5.

1. **hello-world-proj**: This is the source directory. All of your project's source code should be placed in this directory.
2. **hello-world/tools**: This directory contains the individual tools for your project. Your tool package can contain either one tool or many tools. When adding a new tool, you should create another `*_tool.py` under the `tools` folder.
3. **hello-world/tools/hello\_world\_tool.py**: Develop your tool within the `def` function. Use the `@tool` decorator to identify the function as a tool.

*[!Note] There are two ways to write a tool. The default and recommended way is the function implemented way. You can also use the class implementation way, referring to [my\\_tool\\_2.py](https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/my_tool_package/tools/my_tool_2.py) ([https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/my\\_tool\\_package/tools/my\\_tool\\_2.py](https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/my_tool_package/tools/my_tool_2.py)) as an example.*

4. **hello-world/tools/utils.py**: This file implements the tool list method, which collects all the tools defined. It is required to have this tool list method, as it allows the User Interface (UI) to retrieve your tools and display them within the UI.

*[!Note] There's no need to create your own list method if you maintain the existing folder structure. You can simply use the auto-generated list method provided in the `utils.py` file.*

5. **hello\_world/yamls/hello\_world\_tool.yaml**: Tool YAMLS defines the metadata of the tool. The tool list method, as outlined in the `utils.py`, fetches these tool YAMLS.

*[!Note] If you create a new tool, don't forget to also create the corresponding tool YAML. You can run below command under your tool project to auto generate your tool YAML. You may want to specify `-n` for name and `-d` for description, which would be displayed as the tool name and tooltip in prompt flow UI.*

```
python <promptflow github repo>\scripts\tool\generate_package_tool_meta.py -m <tool_module> -o <tool_yaml_path> -n <tool_name>
```

For example:

```
python D:\proj\github\promptflow\scripts\tool\generate_package_tool_meta.py -m hello_world.tools.hello_world_tool -o hello_world
```

To populate your tool module, adhere to the pattern `<package_name>.tools.<tool_name>`, which represents the folder path to your tool within the package.

6. **tests**: This directory contains all your tests, though they are not required for creating your custom tool package. When adding a new tool, you can also create corresponding tests and place them in this directory. Run below command under your tool project:

```
pytest tests
```

7. **MANIFEST.in:** This file is used to determine which files to include in the distribution of the project. Tool YAML files should be included in MANIFEST.in so that your tool YAMLS would be packaged and your tools can show in the UI.

*[!Note] There's no need to update this file if you maintain the existing folder structure.*

8. **setup.py:** This file contains metadata about your project like the name, version, author, and more. Additionally, the entry point is automatically configured for you in the `generate_tool_package_template.py` script. In Python, configuring the entry point in `setup.py` helps establish the primary execution point for a package, streamlining its integration with other software.

The `package_tools` entry point together with the tool list method are used to retrieve all the tools and display them in the UI.

```
entry_points={
 "package_tools": ["<your_tool_name> = <list_module>:<list_method>"],
},
```

*[!Note] There's no need to update this file if you maintain the existing folder structure.*

## Build and share the tool package

Execute the following command in the tool package root directory to build your tool package:

```
python setup.py sdist bdist_wheel
```

This will generate a tool package `<your-package>-0.0.1.tar.gz` and corresponding `whl` file inside the `dist` folder.

Create an account on PyPI if you don't already have one, and install `twine` package by running `pip install twine`.

Upload your package to PyPI by running `twine upload dist/*`, this will prompt you for your PyPI username and password, and then upload your package on PyPI. Once your package is uploaded to PyPI, others can install it using `pip` by running `pip install your-package-name`. Make sure to replace `your-package-name` with the name of your package as it appears on PyPI.

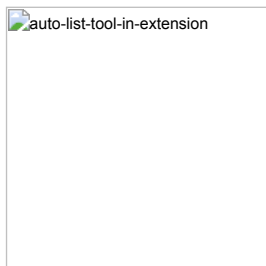
If you only want to put it on Test PyPI, upload your package by running `twine upload --repository-url https://test.pypi.org/legacy/ dist/*`. Once your package is uploaded to Test PyPI, others can install it using `pip` by running `pip install --index-url https://test.pypi.org/simple/ your-package-name`.

## Use your tool from VSCode Extension

- Step1: Install [Prompt flow for VS Code extension](https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow) (<https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow>).
- Step2: Go to terminal and install your tool package in conda environment of the extension. Assume your conda env name is `prompt-flow`.

```
(local_test) PS D:\projects\promptflow\tool-package-quickstart> conda activate prompt-flow
(prompt-flow) PS D:\projects\promptflow\tool-package-quickstart> pip install .\dist\my_tools_package-0.0.1-py3-none-any.whl
```

- Step3: Go to the extension and open one flow folder. Click 'flow.dag.yaml' and preview the flow. Next, click + button and you will see your tools. You may need to reload the



windows to clean previous cache if you don't see your tool in the list.

## FAQs

Why is my custom tool not showing up in the UI?

Confirm that the tool YAML files are included in your custom tool package. You can add the YAML files to [MANIFEST.in](https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/MANIFEST.in) (<https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/MANIFEST.in>) and include the package data in [setup.py](https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/setup.py) (<https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/setup.py>). Alternatively, you can test your tool package using the script below to ensure that you've packaged your tool YAML files and configured the package tool entry point correctly.

1. Make sure to install the tool package in your conda environment before executing this script.
2. Create a python file anywhere and copy the content below into it.

```
import importlib
import importlib.metadata

def test():
 """List all package tools information using the `package-tools` entry point.

 This function iterates through all entry points registered under the group "package_tools."
 For each tool, it imports the associated module to ensure its validity and then prints
 information about the tool.

 Note:
 - Make sure your package is correctly packed to appear in the list.
 - The module is imported to validate its presence and correctness.

 Example of tool information printed:
 ----identifier
 {'module': 'module_name', 'package': 'package_name', 'package_version': 'package_version', ...}
 """
 entry_points = importlib.metadata.entry_points()
 if isinstance(entry_points, list):
 entry_points = entry_points.select(group=PACKAGE_TOOLS_ENTRY)
 else:
 entry_points = entry_points.get(PACKAGE_TOOLS_ENTRY, [])
 for entry_point in entry_points:
 list_tool_func = entry_point.load()
 package_tools = list_tool_func()

 for identifier, tool in package_tools.items():
 importlib.import_module(tool["module"]) # Import the module to ensure its validity
 print(f"----{identifier}\n{tool}")

if __name__ == "__main__":
 test()
```

3. Run this script in your conda environment. This will return the metadata of all tools installed in your local environment, and you should verify that your tools are listed.

## Why am I unable to upload package to PyPI?

- Make sure that the entered username and password of your PyPI account are accurate.
- If you encounter a 403 Forbidden Error, it's likely due to a naming conflict with an existing package. You will need to choose a different name. Package names must be unique on PyPI to avoid confusion and conflicts among users. Before creating a new package, it's recommended to search PyPI (<https://pypi.org/>) to verify that your chosen name is not already taken. If the name you want is unavailable, consider selecting an alternative name or a variation that clearly differentiates your package from the existing one.

## Advanced features

- [Add a Tool Icon \(add-a-tool-icon.md\)](#)
- [Add Category and Tags for Tool \(add-category-and-tags-for-tool.md\)](#)
- [Create and Use Your Own Custom Strong Type Connection \(create-your-own-custom-strong-type-connection.md\)](#)
- [Customize an LLM Tool \(customize\\_an\\_llm\\_tool.md\)](#)
- [Use File Path as Tool Input \(use-file-path-as-tool-input.md\)](#)
- [Create a Dynamic List Tool Input \(create-dynamic-list-tool-input.md\)](#)
- [Create Cascading Tool Inputs \(create-cascading-tool-inputs.md\)](#)

filepath: promptflow/docs/how-to-guides/develop-a-tool/create-your-own-custom-strong-type-connection.md content: # Create and Use Your Own Custom Strong Type Connection  
Connections provide a secure method for managing credentials for external APIs and data sources in prompt flow. This guide explains how to create and use a custom strong type connection.

# What is a Custom Strong Type Connection?

A custom strong type connection in prompt flow allows you to define a custom connection class with strongly typed keys. This provides the following benefits:

- Enhanced user experience - no need to manually enter connection keys.
- Rich intellisense experience - defining key types enables real-time suggestions and auto-completion of available keys as you work in VS Code.
- Central location to view available keys and data types.

For other connections types, please refer to [Connections](https://microsoft.github.io/promptflow/concepts/concept-connections.html) (<https://microsoft.github.io/promptflow/concepts/concept-connections.html>).

## Prerequisites

- Please ensure that your [Prompt flow for VS Code](https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow) (<https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow>) is updated to at least version 1.2.1.
- Please install promptflow package and ensure that its version is 0.1.0b8 or later.

```
pip install promptflow>=0.1.0b8
```

## Create a custom strong type connection

Follow these steps to create a custom strong type connection:

1. Define a Python class inheriting from `CustomStrongTypeConnection`.

*[!Note] Please avoid using the `CustomStrongTypeConnection` class directly.*

2. Use the `Secret` type to indicate secure keys. This enhances security by scrubbing secret keys.
3. Document with docstrings explaining each key.

For example:

```
from promptflow.connections import CustomStrongTypeConnection
from promptflow.contracts.types import Secret

class MyCustomConnection(CustomStrongTypeConnection):
 """My custom strong type connection.

 :param api_key: The api key.
 :type api_key: Secret
 :param api_base: The api base.
 :type api_base: String
 """
 api_key: Secret
 api_base: str = "This is a fake api base."
```

See [this example](https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/my_tool_package/tools/tool_with_custom_strong_type_connection.py) ([https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/my\\_tool\\_package/tools/tool\\_with\\_custom\\_strong\\_type\\_connection.py](https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/my_tool_package/tools/tool_with_custom_strong_type_connection.py)) for a complete implementation.

## Use the connection in a flow

Once you create a custom strong type connection, here are two ways to use it in your flows:

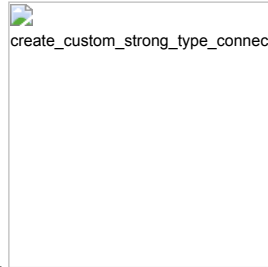
### With Package Tools:

1. Refer to the [Create and Use Tool Package](#) ([create-and-use-tool-package.md#create-custom-tool-package](#)) to build and install your tool package containing the connection.
2. Develop a flow with custom tools. Please take [this folder](https://github.com/microsoft/promptflow/tree/main/examples/tools/use-cases/custom-strong-type-connection-package-tool-showcase) (<https://github.com/microsoft/promptflow/tree/main/examples/tools/use-cases/custom-strong-type-connection-package-tool-showcase>) as an example.
3. Create a custom strong type connection using one of the following methods:

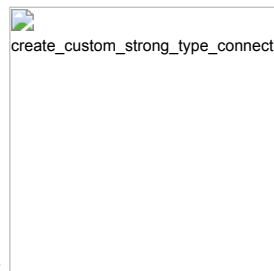




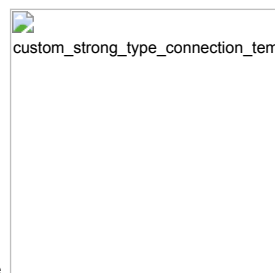
- If the connection type hasn't been created previously, click the 'Add connection' button to create the connection.



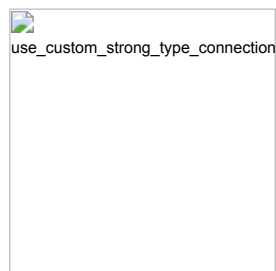
- Click the 'Create connection' plus sign in the CONNECTIONS section.



- Click 'Create connection' plus sign in the Custom category.



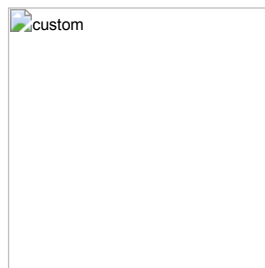
4. Fill in the values starting with `to-replace-with` in the connection template.



5. Run the flow with the created custom strong type connection.

## With Script Tools:

1. Develop a flow with python script tools. Please take [this folder \(https://github.com/microsoft/promptflow/tree/main/examples/tools/use-cases/custom-strong-type-connection-script-tool-showcase\)](https://github.com/microsoft/promptflow/tree/main/examples/tools/use-cases/custom-strong-type-connection-script-tool-showcase) as an example.



2. Create a `CustomConnection`. Fill in the keys and values in the connection template.



3. Run the flow with the created custom connection.

## Local to cloud

When creating the necessary connections in Azure AI, you will need to create a `CustomConnection`. In the node interface of your flow, this connection will be displayed as the `CustomConnection` type.

Please refer to [Run prompt flow in Azure AI \(.NET/Cloud/AzureAI/quick-start/index.md\)](#) for more details.

Here is an example command:

```
pfazure run create --subscription 96aed12-2f73-41cb-b983-6d11a904839b -g promptflow -w my-pf-eus --flow D:\proj\github\ms\promptflow
```

## FAQs

### I followed the steps to create a custom strong type connection, but it's not showing up. What could be the issue?

Once the new tool package is installed in your local environment, a window reload is necessary. This action ensures that the new tools and custom strong type connections become visible and accessible.

filepath: promptflow/docs/how-to-guides/develop-a-tool/index.md content: # Develop a tool We provide guides on how to develop a tool and use it.

```
:maxdepth: 1
:hidden:

create-and-use-tool-package
add-a-tool-icon
add-category-and-tags-for-tool
use-file-path-as-tool-input
customize_an_llm_tool
create-cascading-tool-inputs
create-your-own-custom-strong-type-connection
create-dynamic-list-tool-input
```

filepath: promptflow/docs/how-to-guides/develop-a-tool/create-dynamic-list-tool-input.md content: # Creating a Dynamic List Tool Input

Tool input options can be generated on the fly using a dynamic list. Instead of having predefined static options, the tool author defines a request function that queries backends like APIs to retrieve real-time options. This enables flexible integration with various data sources to populate dynamic options. For instance, the function could call a storage API to list current files. Rather than a hardcoded list, the user sees up-to-date options when running the tool.

## Prerequisites

- Please make sure you have the latest version of [Prompt flow for VS Code](https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow) (<https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow>), installed (v1.3.1+).
- Please install promptflow package and ensure that its version is 1.0.0 or later.

```
pip install promptflow>=1.0.0
```

## Create a tool input with dynamic listing

### Create a list function

To enable dynamic listing, the tool author defines a request function with the following structure:

- **Type:** Regular Python function, can be in tool file or separate file
- **Input:** Accepts parameters needed to fetch options
- **Output:** Returns a list of option objects as `List[Dict[str, Union[str, int, float, list, Dict]]]`:
  - **Required key:**
    - **value:** Internal option value passed to tool function
  - **Optional keys:**
    - **display\_value:** Display text shown in dropdown (defaults to value)
    - **hyperlink:** URL to open when option clicked
    - **description:** Tooltip text on hover

This function can make backend calls to retrieve the latest options, returning them in a standardized dictionary structure for the dynamic list. The required and optional keys enable configuring how each option appears and behaves in the tool input dropdown. See [my\\_list\\_func \(https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/my\\_tool\\_package/tools/tool\\_with\\_dynamic\\_list\\_input.py\)](https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/my_tool_package/tools/tool_with_dynamic_list_input.py) as an example.

```
def my_list_func(prefix: str = "", size: int = 10, **kwargs) -> List[Dict[str, Union[str, int, float, list, Dict]]]:
 """This is a dummy function to generate a list of items.

 :param prefix: prefix to add to each item.
 :param size: number of items to generate.
 :param kwargs: other parameters.
 :return: a list of items. Each item is a dict with the following keys:
 - value: for backend use. Required.
 - display_value: for UI display. Optional.
 - hyperlink: external link. Optional.
 - description: information icon tip. Optional.
 """
 import random

 words = ["apple", "banana", "cherry", "date", "elderberry", "fig", "grape", "honeydew", "kiwi", "lemon"]
 result = []
 for i in range(size):
 random_word = f"{random.choice(words)}{i}"
 cur_item = {
 "value": random_word,
 "display_value": f"{prefix}_{random_word}",
 "hyperlink": f'https://www.bing.com/search?q={random_word}',
 "description": f"this is {i} item",
 }
 result.append(cur_item)

 return result
```

## Configure a tool input with the list function

In `inputs` section of tool YAML, add following properties to the input that you want to make dynamic:

- **dynamic\_list:**
  - **func\_path:** Path to the list function (module\_name.function\_name).
  - **func\_kwargs:** Parameters to pass to the function, can reference other input values.
- **allow\_manual\_entry:** Allow user to enter input value manually. Default to false.
- **is\_multi\_select:** Allow user to select multiple values. Default to false.

See [tool\\_with\\_dynamic\\_list\\_input.yaml \(https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/my\\_tool\\_package/yamls/tool\\_with\\_dynamic\\_list\\_input.yaml\)](https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/my_tool_package/yamls/tool_with_dynamic_list_input.yaml) as an example.

```

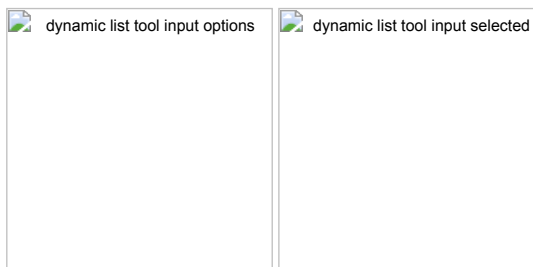
my_tool_package.tools.tool_with_dynamic_list_input.my_tool:
 function: my_tool
 inputs:
 input_text:
 type:
 - list
 dynamic_list:
 func_path: my_tool_package.tools.tool_with_dynamic_list_input.my_list_func
 func_kwargs:
 - name: prefix # argument name to be passed to the function
 type:
 - string
 # if optional is not specified, default to false.
 # this is for UX pre-validation. If optional is false, but no input. UX can throw error in advanced.
 optional: true
 reference: ${inputs.input_prefix} # dynamic reference to another input parameter
 - name: size # another argument name to be passed to the function
 type:
 - int
 optional: true
 default: 10
 # enum and dynamic list may need below setting.
 # allow user to enter input value manually, default false.
 allow_manual_entry: true
 # allow user to select multiple values, default false.
 is_multi_select: true
 # used to filter
 input_prefix:
 type:
 - string
 module: my_tool_package.tools.tool_with_dynamic_list_input
 name: My Tool with Dynamic List Input
 description: This is my tool with dynamic list input
 type: python

```

## Use the tool in VS Code

Once you package and share your tool, you can use it in VS Code per the [tool package guide \(create-and-use-tool-package.md#use-your-tool-from-vscode-extension\)](#). You could try `my-tools-package` for a quick test.

```
pip install my-tools-package>=0.0.8
```



*Note: If your dynamic list function call Azure APIs, you need to login to Azure and set default workspace. Otherwise, the tool input will be empty and you can't select anything. See [FAQs](#) for more details.*

## FAQs

I'm a tool author, and want to dynamically list Azure resources in my tool input. What should I pay attention to?

1. Clarify azure workspace triple "subscription\_id", "resource\_group\_name", "workspace\_name" in the list function signature. System helps append workspace triple to function input parameters if they are in function signature. See [list\\_endpoint\\_names \(https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/my\\_tool\\_package/tools/tool\\_with\\_dynamic\\_list\\_input.py\)](https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/my_tool_package/tools/tool_with_dynamic_list_input.py) as an example.

```
def list_endpoint_names(subscription_id, resource_group_name, workspace_name, prefix: str = "") -> List[Dict[str, str]]:
 """This is an example to show how to get Azure ML resource in tool input list function.

 :param subscription_id: Azure subscription id.
 :param resource_group_name: Azure resource group name.
 :param workspace_name: Azure ML workspace name.
 :param prefix: prefix to add to each item.
 """
 from azure.ai.ml import MLClient
 from azure.identity import DefaultAzureCredential

 credential = DefaultAzureCredential()
 credential.get_token("https://management.azure.com/.default")

 ml_client = MLClient(
 credential=credential,
 subscription_id=subscription_id,
 resource_group_name=resource_group_name,
 workspace_name=workspace_name)
 result = []
 for ep in ml_client.online_endpoints.list():
 hyperlink = (
 f"https://ml.azure.com/endpoints/realtime/{ep.name}/detail?wsid=/subscriptions/"
 f"{subscription_id}/resourceGroups/{resource_group_name}/providers/Microsoft."
 f"MachineLearningServices/workspaces/{workspace_name}"
)
 cur_item = {
 "value": ep.name,
 "display_value": f"{prefix}_{ep.name}",
 # external link to jump to the endpoint page.
 "hyperlink": hyperlink,
 "description": f"this is endpoint: {ep.name}",
 }
 result.append(cur_item)
 return result
```

2. Note in your tool doc that if your tool user want to use the tool at local, they should login to azure and set ws triple as default. Or the tool input will be empty and user can't select anything.

```
az login
az account set --subscription <subscription_id>
az configure --defaults group=<resource_group_name> workspace=<workspace_name>
```

Install azure dependencies.

```
pip install azure-ai-ml
```

```
pip install my-tools-package[azure]>=0.0.8
```

 dynamic list function azure

I'm a tool user, and cannot see any options in dynamic list tool input. What should I do?

If you are unable to see any options in a dynamic list tool input, you may see an error message below the input field stating:

"Unable to display list of items due to XXX. Please contact the tool author/support team for troubleshooting assistance."

If this occurs, follow these troubleshooting steps:

- Note the exact error message shown. This provides details on why the dynamic list failed to populate.
- Contact the tool author/support team and report the issue. Provide the error message so they can investigate the root cause.

filepath: promptflow/docs/how-to-guides/develop-a-tool/customize\_an\_llm\_tool.md content: # Customizing an LLM Tool In this document, we will guide you through the process of customizing an LLM tool, allowing users to seamlessly connect to a large language model with prompt tuning experience using a `PromptTemplate`.

## Prerequisites

- Please ensure that your [Prompt flow for VS Code \(https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow\)](https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow) is updated to version 1.2.0 or later.

## How to customize an LLM tool

Here we use [an existing tool package \(https://github.com/microsoft/promptflow/tree/main/examples/tools/tool-package-quickstart/my\\_tool\\_package\)](https://github.com/microsoft/promptflow/tree/main/examples/tools/tool-package-quickstart/my_tool_package) as an example. If you want to create your own tool, please refer to [create and use tool package \(create-and-use-tool-package.md\)](#).

1. Develop the tool code as in [this example \(https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/my\\_tool\\_package/tools/tool\\_with\\_custom\\_llm\\_type.py\)](https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/my_tool_package/tools/tool_with_custom_llm_type.py).

- Add a `CustomConnection` input to the tool, which is used to authenticate and establish a connection to the large language model.
- Add a `PromptTemplate` input to the tool, which serves as an argument to be passed into the large language model.

```
from jinja2 import Template
from promptflow import tool
from promptflow.connections import CustomConnection
from promptflow.contracts.types import PromptTemplate

@tool
def my_tool(connection: CustomConnection, prompt: PromptTemplate, **kwargs) -> str:
 # Customize your own code to use the connection and prompt here.
 rendered_prompt = Template(prompt, trim_blocks=True, keep_trailing_newline=True).render(**kwargs)
 return rendered_prompt
```

2. Generate the custom LLM tool YAML.

Run the command below in your tool project directory to automatically generate your tool YAML, use `-t "custom_llm"` or `--tool-type "custom_llm"` to indicate this is a custom LLM tool:

```
python <promptflow github repo>\scripts\tool\generate_package_tool_meta.py -m <tool_module> -o <tool_yaml_path> -t "custom_llm"
```

Here we use [an existing tool \(https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/my\\_tool\\_package/yaml/tool\\_with\\_custom\\_llm\\_type.yaml\)](https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/my_tool_package/yaml/tool_with_custom_llm_type.yaml) as an example.

```
cd D:\proj\github\promptflow\examples\tools\tool-package-quickstart

python D:\proj\github\promptflow\scripts\tool\generate_package_tool_meta.py -m my_tool_package.tools.tool_with_custom_llm_type
```

This command will generate a YAML file as follows:

```

my_tool_package.tools.tool_with_custom_llm_type.my_tool:
 name: My Custom LLM Tool
 description: This is a tool to demonstrate how to customize an LLM tool with a PromptTemplate.
 # The type is custom_llm.
 type: custom_llm
 module: my_tool_package.tools.tool_with_custom_llm_type
 function: my_tool
 inputs:
 connection:
 type:
 - CustomConnection

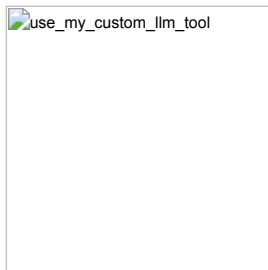
```

## Use the tool in VS Code

Follow the steps to [build and install your tool package \(create-and-use-tool-package.md#build-and-share-the-tool-package\)](#) and [use your tool from VS Code extension \(create-and-use-tool-package.md#use-your-tool-from-vscode-extension\)](#).

Here we use an existing flow to demonstrate the experience, open [this flow \(https://github.com/microsoft/promptflow/blob/main/examples/tools/use-cases/custom\\_llm\\_tool\\_showcase/flow.dag.yaml\)](https://github.com/microsoft/promptflow/blob/main/examples/tools/use-cases/custom_llm_tool_showcase/flow.dag.yaml) in VS Code extension.

- There is a node named "my\_custom\_llm\_tool" with a prompt template file. You can either use an existing file or create a new one as the prompt template file.



filepath: promptflow/docs/how-to-guides/develop-a-tool/add-category-and-tags-for-tool.md content: # Adding Category and Tags for Tool

This document is dedicated to guiding you through the process of categorizing and tagging your tools for optimal organization and efficiency. Categories help you organize your tools into specific folders, making it much easier to find what you need. Tags, on the other hand, work like labels that offer more detailed descriptions. They enable you to quickly search and filter tools based on specific characteristics or functions. By using categories and tags, you'll not only tailor your tool library to your preferences but also save time by effortlessly finding the right tool for any task.

### Attribute Type Required

### Description

category	str	No	Organizes tools into folders by common features.
tags	dict	No	Offers detailed, searchable descriptions of tools through key-value pairs.

### Important Notes:

- Tools without an assigned category will be listed in the root folder.
- Tools lacking tags will display an empty tags field.

## Prerequisites

- Please ensure that your [Prompt flow for VS Code \(https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow\)](https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow) is updated to version 1.1.0 or later.

## How to add category and tags for a tool

Run the command below in your tool project directory to automatically generate your tool YAML, use `-c` or `--category` to add category, and use `--tags` to add tags for your tool:

```
python <promptflow github repo>\scripts\tool\generate_package_tool_meta.py -m <tool_module> -o <tool_yaml_path> --category <tool_cat>
```

Here, we use [an existing tool \(https://github.com/microsoft/promptflow/tree/main/examples/tools/tool-package-quickstart/my\\_tool\\_package/yamls/my\\_tool\\_1.yaml\)](https://github.com/microsoft/promptflow/tree/main/examples/tools/tool-package-quickstart/my_tool_package/yamls/my_tool_1.yaml) as an example. If you wish to create your own tool, please refer to the [create and use tool package \(create-and-use-tool-package.md#create-custom-tool-package\)](#) guide.

```

cd D:\proj\github\promptflow\examples\tools\tool-package-quickstart

python D:\proj\github\promptflow\scripts\tool\generate_package_tool_meta.py -m my_tool_package.tools.my_tool_1 -o my_tool_package\yar

```

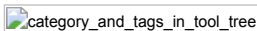
In the auto-generated tool YAML file, the category and tags are shown as below:

```
my_tool_package.tools.my_tool_1.my_tool:
 function: my_tool
 inputs:
 connection:
 type:
 - CustomConnection
 input_text:
 type:
 - string
 module: my_tool_package.tools.my_tool_1
 name: My First Tool
 description: This is my first tool
 type: python
 # Category and tags are shown as below.
 category: test_tool
 tags:
 tag1: value1
 tag2: value2
```

## Tool with category and tags experience in VS Code extension

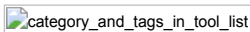
Follow the [steps \(create-and-use-tool-package.md#use-your-tool-from-vscode-extension\)](#) to use your tool via the VS Code extension.

- Experience in the tool tree

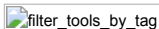
category\_and\_tags\_in\_tool\_tree

- Experience in the tool list

By clicking **More** in the visual editor, you can view your tools along with their category and tags:

category\_and\_tags\_in\_tool\_list

Furthermore, you have the option to search or filter tools based on tags:

filter\_tools\_by\_tag

filepath: [promptflow/docs/how-to-guides/develop-a-tool/add-a-tool-icon.md](#) content: # Adding a Tool Icon A tool icon serves as a graphical representation of your tool in the user interface (UI). Follow this guidance to add a custom tool icon when developing your own tool package.

Adding a custom tool icon is optional. If you do not provide one, the system uses a default icon.

## Prerequisites



- Please ensure that your [Prompt flow for VS Code](https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow) (<https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow>) is updated to version 1.4.2 or later.
- Create a tool package as described in [Create and Use Tool Package](#) ([create-and-use-tool-package.md](#)).
- Prepare custom icon image that meets these requirements:
  - Use PNG, JPG or BMP format.
  - 16x16 pixels to prevent distortion when resizing.
  - Avoid complex images with lots of detail or contrast, as they may not resize well.

See [this example](https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/my_tool_package/icons/custom-tool-icon.png) ([https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/my\\_tool\\_package/icons/custom-tool-icon.png](https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/my_tool_package/icons/custom-tool-icon.png)) as a reference.

- Install dependencies to generate icon data URI:

```
pip install pillow
```

## Add tool icon with *icon* parameter

Run the command below in your tool project directory to automatically generate your tool YAML, use *-i* or *--icon* parameter to add a custom tool icon:

```
python <promptflow github repo>\scripts\tool\generate_package_tool_meta.py -m <tool_module> -o <tool_yaml_path> -i <tool-icon-path>
```

Here we use [an existing tool project](https://github.com/microsoft/promptflow/tree/main/examples/tools/tool-package-quickstart) (<https://github.com/microsoft/promptflow/tree/main/examples/tools/tool-package-quickstart>) as an example.

```
cd D:\proj\github\promptflow\examples\tools\tool-package-quickstart

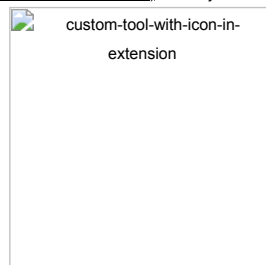
python D:\proj\github\promptflow\scripts\tool\generate_package_tool_meta.py -m my_tool_package.tools.my_tool_1 -o my_tool_package\ya
```

In the auto-generated tool YAML file, the custom tool icon data URI is added in the *icon* field:

```
my_tool_package.tools.my_tool_1.my_tool:
 function: my_tool
 icon: data:image/png;base64,iVBORw0KGgoAAAANSUgAAABAAAAQCAyAAAAf8/9hAAACR01EQVR4nKWS3UuTcRTHP79nm9uJM+f0cqFGI5viRRpjJgkJ3hiCEN
 inputs:
 connection:
 type:
 - CustomConnection
 input_text:
 type:
 - string
 module: my_tool_package.tools.my_tool_1
 name: my_tool
 type: python
```

## Verify the tool icon in VS Code extension

Follow [steps](#) ([create-and-use-tool-package.md#use-your-tool-from-vscode-extension](#)) to use your tool from VS Code extension. Your tool displays with the custom icon:



## FAQ

### Can I preview the tool icon image before adding it to a tool?

Yes, you could run below command under the root folder to generate a data URI for your custom tool icon. Make sure the output file has an *.html* extension.

```
python <path-to-scripts>\tool\convert_image_to_data_url.py --image-path <image_input_path> -o <html_output_path>
```

For example:

```
python D:\proj\github\promptflow\scripts\tool\convert_image_to_data_url.py --image-path D:\proj\github\promptflow\examples\tools\tool-package-quickstart\tool\convert_image_to_data_url.py -o D:\proj\github\promptflow\examples\tools\tool-package-quickstart\tool\output.html
```

The content of `output.html` looks like the following, open it in a web browser to preview the icon.

```
<html>
<body>

</body>
</html>
```

## Can I add a tool icon to an existing tool package?

Yes, you can refer to the [preview icon \(add-a-tool-icon.md#can-i-preview-the-tool-icon-image-before-adding-it-to-a-tool\)](#) section to generate the data URI and manually add the data URI to the tool's YAML file.

## Can I add tool icons for dark and light mode separately?

Yes, you can add the tool icon data URIs manually or run the command below in your tool project directory to automatically generate your tool YAML, use `--icon-light` to add a custom tool icon for the light mode and use `--icon-dark` to add a custom tool icon for the dark mode:

```
python <promptflow github repo>\scripts\tool\generate_package_tool_meta.py -m <tool_module> -o <tool_yaml_path> --icon-light <light-mode-data-uri> --icon-dark <dark-mode-data-uri>
```

Here we use [an existing tool project \(https://github.com/microsoft/promptflow/tree/main/examples/tools/tool-package-quickstart\)](https://github.com/microsoft/promptflow/tree/main/examples/tools/tool-package-quickstart) as an example.

```
cd D:\proj\github\promptflow\examples\tools\tool-package-quickstart

python D:\proj\github\promptflow\scripts\tool\generate_package_tool_meta.py -m my_tool_package.tools.my_tool_1 -o my_tool_package\yaml\my_tool_1.yaml
```

In the auto-generated tool YAML file, the light and dark tool icon data URIs are added in the `icon` field:

```
my_tool_package.tools.my_tool_1.my_tool:
 function: my_tool
 icon:
 dark: data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAABAAAAQCAIAAACQkKg2AAAB00lEQVR4nI1SO2iTURT+7iNNb16a+Cg6iJWqRKwVRirWV6GVUkrF0
 light: data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAABAAAAQCAIAAACQkKg2AAAB2U1EQVR4nH1SO2hUQRQ9c18K33u72cXs7jOL8UeQCCJoJaIgKAI
 inputs:
 connection:
 type:
 - CustomConnection
 input_text:
 type:
 - string
 module: my_tool_package.tools.my_tool_1
 name: my_tool
 type: python
```

Note: Both light and dark icons are optional. If you set either a light or dark icon, it will be used in its respective mode, and the system default icon will be used in the other mode.

filepath: promptflow/docs/how-to-guides/develop-a-tool/use-file-path-as-tool-input.md content: # Using File Path as Tool Input

Users sometimes need to reference local files within a tool to implement specific logic. To simplify this, we've introduced the `FilePath` input type. This input type enables users to either select an existing file or create a new one, then pass it to a tool, allowing the tool to access the file's content.

In this guide, we will provide a detailed walkthrough on how to use `FilePath` as a tool input. We will also demonstrate the user experience when utilizing this type of tool within a flow.

## Prerequisites

- Please install promptflow package and ensure that its version is 0.1.0b8 or later.

```
pip install promptflow>=0.1.0b8
```

- Please ensure that your [Prompt flow for VS Code \(https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow\)](https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow) is updated to version 1.1.0 or later.

# Using File Path as Package Tool Input

## How to create a package tool with file path input

Here we use [an existing tool package \(https://github.com/microsoft/promptflow/tree/main/examples/tools/tool-package-quickstart/my\\_tool\\_package\)](https://github.com/microsoft/promptflow/tree/main/examples/tools/tool-package-quickstart/my_tool_package) as an example. If you want to create your own tool, please refer to [create and use tool package \(create-and-use-tool-package.md#create-custom-tool-package\)](https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/my_tool_package/yamls/tool_with_file_path_input.yaml).

1. Add a `FilePath` input for your tool, like in [this example \(https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/my\\_tool\\_package/tools/tool\\_with\\_file\\_path\\_input.py\)](https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/my_tool_package/tools/tool_with_file_path_input.py).

```
import importlib
from pathlib import Path
from promptflow import tool

1. import the FilePath type
from promptflow.contracts.types import FilePath

2. add a FilePath input for your tool method
@tool
def my_tool(input_file: FilePath, input_text: str) -> str:
 # 3. customise your own code to handle and use the input_file here
 new_module = importlib.import_module(Path(input_file).stem)

 return new_module.hello(input_text)
```

2. `FilePath` input format in a tool YAML, like in [this example \(https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/my\\_tool\\_package/yamls/tool\\_with\\_file\\_path\\_input.yaml\)](https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/my_tool_package/yamls/tool_with_file_path_input.yaml).

```
my_tool_package.tools.tool_with_file_path_input.my_tool:
 function: my_tool
 inputs:
 # yaml format for FilePath input
 input_file:
 type:
 - file_path
 input_text:
 type:
 - string
 module: my_tool_package.tools.tool_with_file_path_input
 name: Tool with FilePath Input
 description: This is a tool to demonstrate the usage of FilePath input
 type: python
```

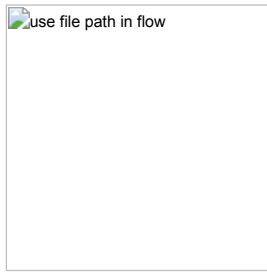
*[!Note] tool yaml file can be generated using a python script. For further details, please refer to [create custom tool package \(create-and-use-tool-package.md#create-custom-tool-package\)](https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/my_tool_package/yamls/tool_with_file_path_input.yaml).*

## Use tool with a file path input in VS Code extension

Follow steps to [build and install your tool package \(create-and-use-tool-package.md#build-and-share-the-tool-package\)](https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/my_tool_package/yamls/tool_with_file_path_input.yaml) and [use your tool from VS Code extension \(create-and-use-tool-package.md#use-your-tool-from-vscode-extension\)](https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/my_tool_package/yamls/tool_with_file_path_input.yaml).

Here we use an existing flow to demonstrate the experience, open [this flow \(https://github.com/microsoft/promptflow/blob/main/examples/tools/use-cases/filepath-input-tool-showcase/flow.dag.yaml\)](https://github.com/microsoft/promptflow/blob/main/examples/tools/use-cases/filepath-input-tool-showcase/flow.dag.yaml) in VS Code extension:

- There is a node named "Tool\_with\_FilePath\_Input" with a `file_path` type input called `input_file`.
- Click the picker icon to open the UI for selecting an existing file or creating a new file to use as input.



## Using File Path as Script Tool Input

We can also utilize the `FilePath` input type directly in a script tool, eliminating the need to create a package tool.

1. Initiate an empty flow in the VS Code extension and add a python node titled 'python\_node\_with\_filepath' into it in the Visual Editor page.
2. Select the link `python_node_with_filepath.py` in the node to modify the python method to include a `FilePath` input as shown below, and save the code change.

```
import importlib
from pathlib import Path
from promptflow import tool

1. import the FilePath type
from promptflow.contracts.types import FilePath

2. add a FilePath input for your tool method
@tool
def my_tool(input_file: FilePath, input_text: str) -> str:
 # 3. customise your own code to handle and use the input_file here
 new_module = importlib.import_module(Path(input_file).stem)

 return new_module.hello(input_text)
```

3. Return to the flow Visual Editor page, click the picker icon to launch the UI for selecting an existing file or creating a new file to use as input, here we select [this file](https://github.com/microsoft/promptflow/blob/main/examples/tools/use-cases/filepath-input-tool-showcase/hello_method.py) ([https://github.com/microsoft/promptflow/blob/main/examples/tools/use-cases/filepath-input-tool-showcase/hello\\_method.py](https://github.com/microsoft/promptflow/blob/main/examples/tools/use-cases/filepath-input-tool-showcase/hello_method.py)), as an example.



## FAQ

### What are some practical use cases for this feature?

The `FilePath` input enables several useful workflows:

1. **Dynamically load modules** - As shown in the demo, you can load a Python module from a specific script file selected by the user. This allows flexible custom logic.
2. **Load arbitrary data files** - The tool can load data from files like .csv, .txt, .json, etc. This provides an easy way to inject external data into a tool.

So in summary, `FilePath` input gives tools flexible access to external files provided by users at runtime. This unlocks many useful scenarios like the ones above.

filepath: promptflow/docs/how-to-guides/develop-a-tool/create-cascading-tool-inputs.md content: # Creating Cascading Tool Inputs

Cascading input settings are useful when the value of one input field determines which subsequent inputs are shown. This makes the input process more streamlined, user-friendly, and error-free. This guide will walk through how to create cascading inputs for your tools.

## Prerequisites

Please make sure you have the latest version of [Prompt flow for VS Code](https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow) (<https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow>) installed (v1.2.0+).

# Create a tool with cascading inputs

We'll build out an example tool to show how cascading inputs work. The `student_id` and `teacher_id` inputs will be controlled by the value selected for the `user_type` input.

Here's how to configure this in the tool code and YAML.

1. Develop the tool function, following the [cascading inputs example](https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/my_tool_package/tools/tool_with_cascading_inputs.py) ([https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/my\\_tool\\_package/tools/tool\\_with\\_cascading\\_inputs.py](https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/my_tool_package/tools/tool_with_cascading_inputs.py)). Key points:

- Use the `@tool` decorator to mark the function as a tool.
- Define `UserType` as an Enum class, as it accepts only a specific set of fixed values in this example.
- Conditionally use inputs in the tool logic based on `user_type`.

```
from enum import Enum

from promptflow import tool

class UserType(str, Enum):
 STUDENT = "student"
 TEACHER = "teacher"

@tool
def my_tool(user_type: Enum, student_id: str = "", teacher_id: str = "") -> str:
 """This is a dummy function to support cascading inputs.

 :param user_type: user type, student or teacher.
 :param student_id: student id.
 :param teacher_id: teacher id.
 :return: id of the user.
 If user_type is student, return student_id.
 If user_type is teacher, return teacher_id.
 """
 if user_type == UserType.STUDENT:
 return student_id
 elif user_type == UserType.TEACHER:
 return teacher_id
 else:
 raise Exception("Invalid user.")
```

2. Generate a starting YAML for your tool per the [tool package guide \(create-and-use-tool-package.md\)](#), then update it to enable cascading:

Add `enabled_by` and `enabled_by_value` to control visibility of dependent inputs. See the [example YAML](#)

([https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/my\\_tool\\_package/yaml/tool\\_with\\_cascading\\_inputs.yaml](https://github.com/microsoft/promptflow/blob/main/examples/tools/tool-package-quickstart/my_tool_package/yaml/tool_with_cascading_inputs.yaml)) for reference.

- The `enabled_by` attribute specifies the input field, which must be an enum type, that controls the visibility of the dependent input field.
- The `enabled_by_value` attribute defines the accepted enum values from the `enabled_by` field that will make this dependent input field visible.

**Note:** `enabled_by_value` takes a list, allowing multiple values to enable an input.

```

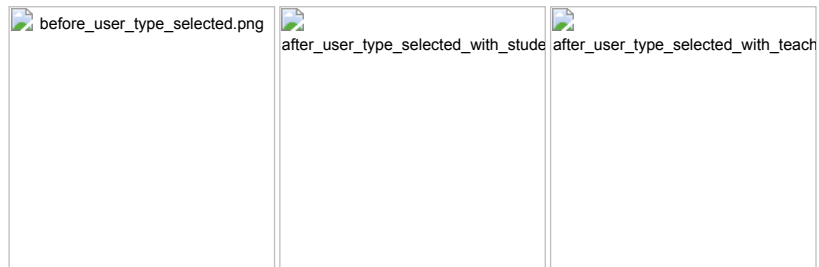
my_tool_package.tools.tool_with_cascading_inputs.my_tool:
 function: my_tool
 inputs:
 user_type:
 type:
 - string
 enum:
 - student
 - teacher
 student_id:
 type:
 - string
 # This input is enabled by the input "user_type".
 enabled_by: user_type
 # This input is enabled when "user_type" is "student".
 enabled_by_value: [student]
 teacher_id:
 type:
 - string
 enabled_by: user_type
 enabled_by_value: [teacher]
 module: my_tool_package.tools.tool_with_cascading_inputs
 name: My Tool with Cascading Inputs
 description: This is my tool with cascading inputs
 type: python

```

## Use the tool in VS Code

Once you package and share your tool, you can use it in VS Code per the [tool package guide \(create-and-use-tool-package.md\)](#). We have a [demo flow \(https://github.com/microsoft/promptflow/tree/main/examples/tools/use-cases/cascading-inputs-tool-showcase\)](#) you can try.

Before selecting a `user_type`, the `student_id` and `teacher_id` inputs are hidden. Once you pick the `user_type`, the corresponding input appears.



## FAQs

### How do I create multi-layer cascading inputs?

If you are dealing with multiple levels of cascading inputs, you can effectively manage the dependencies between them by using the `enabled_by` and `enabled_by_value` attributes. For example:

```

my_tool_package.tools.tool_with_multi_layer_cascading_inputs.my_tool:
 function: my_tool
 inputs:
 event_type:
 type:
 - string
 enum:
 - corporate
 - private
 corporate_theme:
 type:
 - string
 # This input is enabled by the input "event_type".
 enabled_by: event_type
 # This input is enabled when "event_type" is "corporate".
 enabled_by_value: [corporate]
 enum:
 - seminar
 - team_building
 seminar_location:
 type:
 - string
 # This input is enabled by the input "corporate_theme".
 enabled_by: corporate_theme
 # This input is enabled when "corporate_theme" is "seminar".
 enabled_by_value: [seminar]
 private_theme:
 type:
 - string
 # This input is enabled by the input "event_type".
 enabled_by: event_type
 # This input is enabled when "event_type" is "private".
 enabled_by_value: [private]
 module: my_tool_package.tools.tool_with_multi_layer_cascading_inputs
 name: My Tool with Multi-Layer Cascading Inputs
 description: This is my tool with multi-layer cascading inputs
 type: python

```

Inputs will be enabled in a cascading way based on selections.

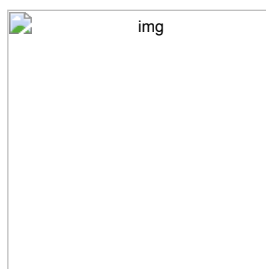
filepath: promptflow/docs/cloud/index.md content: # Cloud

Prompt flow streamlines the process of developing AI applications based on LLM, easing prompt engineering, prototyping, evaluating, and fine-tuning for high-quality products.

Transitioning to production, however, typically requires a comprehensive **LLMOps** process, LLMOps is short for large language model operations. This can often be a complex task, demanding high availability and security, particularly vital for large-scale team collaboration and lifecycle management when deploying to production.

To assist in this journey, we've introduced **Azure AI**, a **cloud-based platform** tailored for executing LLMOps, focusing on boosting productivity for enterprises.

- Private data access and controls
- Collaborative development
- Automating iterative experimentation and CI/CD
- Deployment and optimization
- Safe and Responsible AI



## Transitioning from local to cloud (Azure AI)

In prompt flow, You can develop your flow locally and then seamlessly transition to Azure AI. Here are a few scenarios where this might be beneficial: | Scenario | Benefit | How to | --  
- | --- | --- | | Collaborative development | Azure AI provides a cloud-based platform for flow development and management, facilitating sharing and collaboration across multiple teams, organizations, and tenants. | [Submit a run using pfazure \(.azureai/quick-start/index.md\)](#), based on the flow file in your code base. | | Processing large amounts of data in parallel pipelines | Transitioning to Azure AI allows you to use your flow as a parallel component in a pipeline job, enabling you to process large amounts of data and integrate with existing pipelines. | Learn how to [Use flow in Azure ML pipeline job \(.azureai/use-flow-in-azure-ml-pipeline.md\)](#). | | Large-scale Deployment | Azure AI allows for seamless deployment and optimization when your flow is ready for production and requires high availability and security. | Use `pf flow build` to deploy your flow to [Azure App Service \(.azureai/deploy-to-azure-appservice.md\)](#). | | Data Security and Responsible AI Practices | If your flow handling sensitive data or requiring ethical AI practices, Azure AI offers robust security, responsible AI services, and features for data storage, identity, and access control. | Follow the steps mentioned in the above scenarios. |

For more resources on Azure AI, visit the cloud documentation site: [Build AI solutions with prompt flow \(https://learn.microsoft.com/en-us/azure/machine-learning/prompt-flow/get-started-prompt-flow?view=azureml-api-2\)](https://learn.microsoft.com/en-us/azure/machine-learning/prompt-flow/get-started-prompt-flow?view=azureml-api-2).

```
:caption: AzureAI
:maxdepth: 2
azureai/quick-start/index
azureai/manage-flows
azureai/consume-connections-from-azure-ai
azureai/deploy-to-azure-appservice
azureai/use-flow-in-azure-ml-pipeline.md
azureai/faq
azureai/runtime-change-log.md
```

filepath: promptflow/docs/cloud/azureai/use-flow-in-azure-ml-pipeline.md content: # Use flow in Azure ML pipeline job

After you have developed and tested the flow in [init and test a flow \(./.how-to-guides/init-and-test-a-flow.md\)](#), this guide will help you learn how to use a flow as a parallel component in a pipeline job on AzureML, so that you can integrate the created flow with existing pipelines and process a large amount of data.

- Customer need to install the extension `ml>=2.21.0` to enable this feature in CLI and package `azure-ai-ml>=1.11.0` to enable this feature in SDK;
- Customer need to put `$schema` in the target `flow.dag.yaml` to enable this feature;
  - `flow.dag.yaml`: `$schema:https://azuremlschemas.azureedge.net/promptflow/latest/Flow.schema.json`
  - `run.yaml`: `$schema:https://azuremlschemas.azureedge.net/promptflow/latest/Run.schema.json`
- Customer need to generate `flow.tools.json` for the target flow before below usage. The generation can be done by `pf flow validate`.

For more information about AzureML and component:

- [Install and set up the CLI\(v2\) \(https://learn.microsoft.com/en-us/azure/machine-learning/how-to-configure-cli?view=azureml-api-2&tabs=public\)](https://learn.microsoft.com/en-us/azure/machine-learning/how-to-configure-cli?view=azureml-api-2&tabs=public).
- [Install and set up the SDK\(v2\) \(https://learn.microsoft.com/en-us/python/api/overview/azure/ai-ml-readme?view=azure-python\)](https://learn.microsoft.com/en-us/python/api/overview/azure/ai-ml-readme?view=azure-python).
- [What is a pipeline \(https://learn.microsoft.com/en-us/azure/machine-learning/concept-ml-pipelines?view=azureml-api-2\)](https://learn.microsoft.com/en-us/azure/machine-learning/concept-ml-pipelines?view=azureml-api-2).
- [What is a component \(https://learn.microsoft.com/en-us/azure/machine-learning/concept-component?view=azureml-api-2\)](https://learn.microsoft.com/en-us/azure/machine-learning/concept-component?view=azureml-api-2).

## Register a flow as a component

Customer can register a flow as a component with either CLI or SDK.

:sync: CLI

```
Register flow as a component
Default component name will be the name of flow folder, which is not a valid component name, so we override it here; default version is 1
az ml component create --file standard/web-classification/flow.dag.yaml --set name=web_classification

Register flow as a component with parameters override
az ml component create --file standard/web-classification/flow.dag.yaml --version 2 --set name=web_classification_updated
```

:sync: SDK



```

from azure.ai.ml import MLClient, load_component

ml_client = MLClient()

Register flow as a component
flow_component = load_component("standard/web-classification/flow.dag.yaml")
Default component name will be the name of flow folder, which is not a valid component name, so we override it here; default version is 1
flow_component.name = "web_classification"
ml_client.components.create_or_update(flow_component)

Register flow as a component with parameters override
ml_client.components.create_or_update(
 "standard/web-classification/flow.dag.yaml",
 version="2",
 params_override=[
 {"name": "web_classification_updated"}
]
)

```

After registered a flow as a component, they can be referred in a pipeline job like [regular registered components \(https://github.com/Azure/azureml-examples/tree/main/cli/jobs/pipelines-with-components/basics/1b\\_e2e\\_registered\\_components\)](https://github.com/Azure/azureml-examples/tree/main/cli/jobs/pipelines-with-components/basics/1b_e2e_registered_components).

## Directly use a flow in a pipeline job

Besides explicitly registering a flow as a component, customer can also directly use flow in a pipeline job:

All connections and flow inputs will be exposed as input parameters of the component. Default value can be provided in flow/run definition; they can also be set/overwrite on job submission:

:sync: CLI

```

...
jobs:
 flow_node:
 type: parallel
 component: standard/web-classification/flow.dag.yaml
 inputs:
 data: ${parent.inputs.web_classification_input}
 url: "${data.url}"
 connections.summarize_text_content.connection: azure_open_ai_connection
 connections.summarize_text_content.deployment_name: text-davinci-003
 ...

```

Above is part of the pipeline job yaml, see here for [full example \(https://github.com/Azure/azureml-examples/tree/main/cli/jobs/pipelines-with-components/pipeline\\_job\\_with\\_flow\\_as\\_component\)](https://github.com/Azure/azureml-examples/tree/main/cli/jobs/pipelines-with-components/pipeline_job_with_flow_as_component).

:sync: SDK

```

from azure.identity import DefaultAzureCredential
from azure.ai.ml import MLClient, load_component, Input
from azure.ai.ml.dsl import pipeline

credential = DefaultAzureCredential()
ml_client = MLClient.from_config(credential=credential)
data_input = Input(path="standard/web-classification/data.jsonl", type='uri_file')

Load flow as a component
flow_component = load_component("standard/web-classification/flow.dag.yaml")

@pipeline
def pipeline_func_with_flow(data):
 flow_node = flow_component(
 data=data,
 url="{data.url}",
 connections={
 "summarize_text_content": {
 "connection": "azure_open_ai_connection",
 "deployment_name": "text-davinci-003",
 },
 },
)
 flow_node.compute = "cpu-cluster"

pipeline_with_flow = pipeline_func_with_flow(data=data_input)

pipeline_job = ml_client.jobs.create_or_update(pipeline_with_flow)
ml_client.jobs.stream(pipeline_job.name)

```

Above is part of the pipeline job python code, see here for [full example \(https://github.com/Azure/azureml-examples/tree/main/sdk/python/jobs/pipelines/1l\\_flow\\_in\\_pipeline\)](https://github.com/Azure/azureml-examples/tree/main/sdk/python/jobs/pipelines/1l_flow_in_pipeline).

## Difference across flow in prompt flow and pipeline job

In prompt flow, flow runs on [runtime \(https://learn.microsoft.com/en-us/azure/machine-learning/prompt-flow/concept-runtime\)](https://learn.microsoft.com/en-us/azure/machine-learning/prompt-flow/concept-runtime), which is designed for prompt flow and easy to customize; while in pipeline job, flow runs on different types of compute, and usually compute cluster.

Given above, if your flow has logic relying on identity or environment variable, please be aware of this difference as you might run into some unexpected error(s) when the flow runs in pipeline job, and you might need some extra configurations to make it work.

filepath: promptflow/docs/cloud/azureai/runtime-change-log.md content: # Change log of default runtime image

## Runtime image

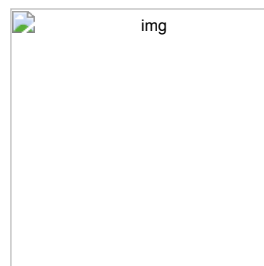
In Azure Machine Learning prompt flow, runtime provides the environment to execute flows. The default runtime includes a pre-built Docker image, which contains all necessary dependent packages.

### Pull image

The image can be pulled by specifying a runtime version and executing the following command:

```
docker pull mcr.microsoft.com/azureml/promptflow/promptflow-runtime-stable:<runtime_version>
```

### Check image version



You can check the runtime image version from the flow execution log:

# Change log

Default runtime image is continuously updated, and here we record the new features and fixed bugs of each image version.

## 20240124.v3

### New features

- Support downloading data from Azure Machine Learning registry for batch run.
- Show node status when one line of a batch run times out.

### Bugs fixed

- Fix the bug that exception raised during preparing data is not set in run history.
- Fix the bug that unexpected exception is raised when executor process crushes.

## 20240116.v1

### New features

NA

### Bugs fixed

- Add validation for wrong connection type for LLM tool.

## 20240111.v2

### New features

- Support error log scrubbing for heron jobs.

### Bugs fixed

- Fixed the compatibility issue between runtime and promptflow package < 1.3.0

filepath: promptflow/docs/cloud/azureai/consume-connections-from-azure-ai.md content: # Consume connections from Azure AI

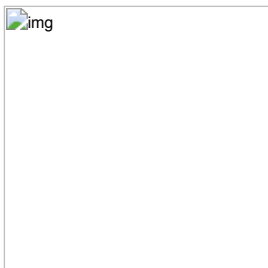
For a smooth development flow that transitions from cloud (Azure AI) to local environments, you can directly utilize the connection already established on the cloud by setting the connection provider to "Azure AI connections".

You can set the connection provider using the following steps:

1. Navigate to the connection list in the VS Code primary sidebar.
2. Click on the ... (more options icon) at the top and select the `Set connection provider` option.



3. Choose one of the "Azure AI connections" provider types that you wish to use. [Click to learn more about the differences between the connection providers.](#)



1. If you choose "Azure AI Connections - for current working directory", then you need to specify the cloud resources in the `config.json` file within the project folder.



2. If you choose "Azure AI Connections - for this machine", specify the cloud resources in the connection string. You can do this in one of two ways:

(1) Input connection string in the input box above. For example `azureml://subscriptions/<your-subscription>/resourceGroups/<your-resourcegroup>/providers/Microsoft.MachineLearningServices/workspaces/<your-workspace>`



(2) Follow the wizard to set up your config step by step.



4. Once the connection provider is set, the connection list will automatically refresh, displaying the connections retrieved from the selected provider.

Note:

1. You need to have a project folder open to use the "Azure AI connections - for current working directory" option.
2. Once you change the connection provider, it will stay that way until you change it again and save the new setting.

## Different connection providers

Currently, we support three types of connections:

Connection provider	Type	Description	Provider Specification	Use Case
Local Connections	Local	Enables consume the connections created and locally and stored in local sqlite.	NA	Ideal when connections need to be stored and managed locally.
Azure AI connection - For current working directory	Cloud provider	Enables the consumption of connections from a cloud provider, such as a specific Azure Machine Learning workspace or Azure AI project.	Specify the resource ID in a <code>config.json</code> file placed in the project folder. <a href="#">Click here for more details</a> ( <a href="#">./../how-to-guides/set-global-configs.md#azureml</a> ) Use a <code>connection string</code> to specify a cloud resource as the provider on your local machine. <a href="#">Click here for more details</a> ( <a href="#">./../how-to-guides/set-global-configs.md#full-azure-machine-learning-workspace-resource-id</a> )	A dynamic approach for consuming connections from different providers in specific projects. Allows for setting different provider configurations for different flows by updating the <code>config.json</code> in the project folder.
Azure AI connection - For this machine	Cloud	Enables the consumption of connections from a cloud provider, such as a specific Azure Machine Learning workspace or Azure AI project.	<a href="#">Click here for more details</a> ( <a href="#">./../how-to-guides/set-global-configs.md#full-azure-machine-learning-workspace-resource-id</a> )	A global provider setting that applies across all working directories on your machine.

## Next steps

- Set global configs on [connection.provider](#) ([../how-to-guides/set-global-configs.md#connectionprovider](#)).
- [Manage connections on local](#) ([../how-to-guides/manage-connections.md](#)).

filepath: promptflow/docs/cloud/azureai/deploy-to-azure-appservice.md content: # Deploy to Azure App Service

[Azure App Service](#) (<https://learn.microsoft.com/azure/app-service/>) is an HTTP-based service for hosting web applications, REST APIs, and mobile back ends. The scripts (deploy.sh for bash and deploy.ps1 for powershell) under [this folder](#) (<https://github.com/microsoft/promptflow/tree/main/examples/tutorials/flow-deploy/azure-app-service>) are here to help deploy the docker image to Azure App Service.

This example demos how to deploy [web-classification](#) (<https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification/>) deploy a flow using Azure App Service.

## Build a flow as docker format app

Use the command below to build a flow as docker format app:

```
pf flow build --source ../../flows/standard/web-classification --output dist --format docker
```

Note that all dependent connections must be created before building as docker.

## Deploy with Azure App Service

The two scripts will do the following things:

1. Create a resource group if not exists.
2. Build and push the image to docker registry.
3. Create an app service plan with the given sku.
4. Create an app with specified name, set the deployment container image to the pushed docker image.
5. Set up the environment variables for the app.

Example command to use bash script:

```
bash deploy.sh --path dist -i <image_tag> --name my-app-23d8m -r <docker_registry> -g <resource_group>
```

See the full parameters by `bash deploy.sh -h`.

Example command to use powershell script:

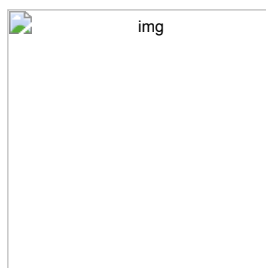
```
.\deploy.ps1 -Path dist -i <image_tag> -n my-app-23d8m -r <docker_registry> -g <resource_group>
```

See the full parameters by `.\deploy.ps1 -h`.

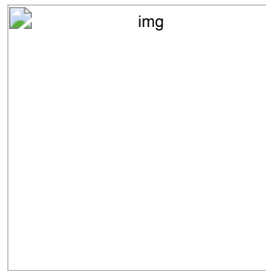
Note that the `name` will produce a unique FQDN as `AppName.azurewebsites.net`.

## View and test the web app

The web app can be found via [azure portal](#) (<https://portal.azure.com/>).



After the app created, you will need to go to <https://portal.azure.com/> (<https://portal.azure.com/>) find the app and set up the environment variables at (Settings>Configuration) or (Settings>Environment variables), then restart the app.

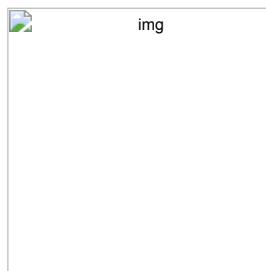


The app can be tested by sending a POST request to the endpoint or browse the test page.

```
curl https://<name>.azurewebsites.net/score --data '{"url":"https://play.google.com/store/apps/details?id=com.twitter.android"}' -X POST
```

```
Invoke-WebRequest -URI https://<name>.azurewebsites.net/score -Body '{"url":"https://play.google.com/store/apps/details?id=com.twitter.android"}'
```

Browse the app at Overview and see the test page:



Tips:

- Reach deployment logs at (Deployment>Deployment Central) and app logs at (Monitoring>Log stream).
- Reach advanced deployment tools at (Development Tools>Advanced Tools).
- Reach more details about app service at [Azure App Service](https://learn.microsoft.com/azure/app-service/) (<https://learn.microsoft.com/azure/app-service/>).

## Next steps

- Try the example [here](https://github.com/microsoft/promptflow/blob/main/examples/tutorials/flow-deploy/azure-app-service/) (<https://github.com/microsoft/promptflow/blob/main/examples/tutorials/flow-deploy/azure-app-service/>).

filepath: promptflow/docs/cloud/azureai/manage-flows.md content: # Manage flows

This is an experimental feature, and may change at any time. Learn [more](#) ([./../how-to-guides/faq.md#stable-vs-experimental](#)).

This documentation will walk you through how to manage your flow with CLI and SDK on [Azure AI](#) (<https://learn.microsoft.com/en-us/azure/machine-learning/prompt-flow/overview-what-is-prompt-flow?view=azureml-api-2>). The flow examples in this guide come from [examples/flows/standard](#) (<https://github.com/microsoft/promptflow/tree/main/examples/flows/standard>).

In general:

- For CLI, you can run `pfazure flow --help` in the terminal to see help messages.
- For SDK, you can refer to [Promptflow Python Library Reference](#) ([./../reference/python-library-reference/promptflow.md](#)) and check `promptflow.azure.PFClient.flows` for more flow operations.
- Refer to the prerequisites in [Quick start](#) ([./quick-start/index.md#prerequisites](#)).
- Use the `az login` command in the command line to log in. This enables promptflow to access your credentials.

Let's take a look at the following topics:

- [Manage flows](#)
  - [Create a flow](#)
  - [List flows](#)

## Create a flow

:sync: CLI

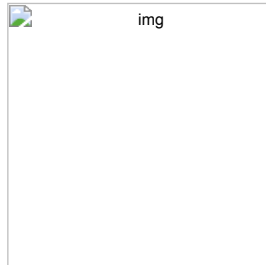
To set the target workspace, you can either specify it in the CLI command or set default value in the Azure CLI. You can refer to [Quick start](#) ([./quick-start/index.md#submit-a-run-to-workspace](#)) for more information.

To create a flow to Azure from local flow directory, you can use

```
create the flow
pfazure flow create --flow <path-to-flow-folder>

create the flow with metadata
pfazure flow create --flow <path-to-flow-folder> --set display_name=<display-name> description=<description> tags.key1=value1
```

After the flow is created successfully, you can see the flow summary in the command line.



:sync: SDK

#### 1. Import the required libraries

```
from azure.identity import DefaultAzureCredential, InteractiveBrowserCredential
azure version promptflow apis
from promptflow.azure import PFClient
```

#### 2. Get credential

```
try:
 credential = DefaultAzureCredential()
 # Check if given credential can get token successfully.
 credential.get_token("https://management.azure.com/.default")
except Exception as ex:
 # Fall back to InteractiveBrowserCredential in case DefaultAzureCredential not work
 credential = InteractiveBrowserCredential()
```

#### 3. Get a handle to the workspace

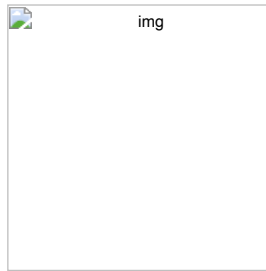
```
Get a handle to workspace
pf = PFClient(
 credential=credential,
 subscription_id="<SUBSCRIPTION_ID>", # this will look like xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
 resource_group_name="<RESOURCE_GROUP>",
 workspace_name="<AML_WORKSPACE_NAME>",
)
```

#### 4. Create the flow

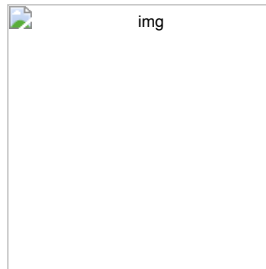
```
specify flow path
flow = "./web-classification"

create flow to Azure
flow = pf.flows.create_or_update(
 flow=flow, # path to the flow folder
 display_name="my-web-classification", # it will be "web-classification-{timestamp}" if not specified
 type="standard", # it will be "standard" if not specified
)
```

On Azure portal, you can see the created flow in the flow list.



And the flow source folder on file share is `Users/<alias>/promptflow/<flow-display-name>`:



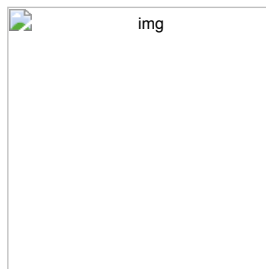
Note that if the flow display name is not specified, it will default to the flow folder name + timestamp. (e.g. `web-classification-11-13-2023-14-19-10`)

## List flows

:sync: CLI

List flows with default json format:

```
pfazure flow list --max-results 1
```



:sync: SDK

```
reuse the pf client created in "create a flow" section
flows = pf.flows.list(max_results=1)
```

filepath: `promptflow/docs/cloud/azureai/faq.md` content: # Frequency asked questions (FAQ)

## Troubleshooting

### Token expired when run pfazure cmd

If hit error "AADSTS700082: The refresh token has expired due to inactivity." when running pfazure cmd, it's caused by local cached token expired. Please clear the cached token under `"%LOCALAPPDATA%\IdentityService\msal.cache"`. Then run below command to login again:

```
az login
```

filepath: `promptflow/docs/cloud/azureai/quick-start/index.md` content: # Run prompt flow in Azure AI

This is an experimental feature, and may change at any time. Learn [more \(./../how-to-guides/faq.md#stable-vs-experimental\)](#).

Assuming you have learned how to create and run a flow following [Quick start \(./../how-to-guides/quick-start.md\)](#). This guide will walk you through the main process of how to submit a promptflow run to [Azure AI \(https://learn.microsoft.com/en-us/azure/machine-learning/prompt-flow/overview-what-is-prompt-flow?view=azureml-api-2\)](#).



Benefits of use Azure AI comparison to just run locally:

- **Designed for team collaboration:** Portal UI is a better fix for sharing & presentation your flow and runs. And workspace can better organize team shared resources like connections.
- **Enterprise Readiness Solutions:** prompt flow leverages Azure AI's robust enterprise readiness solutions, providing a secure, scalable, and reliable foundation for the development, experimentation, and deployment of flows.

## Prerequisites

1. An Azure account with an active subscription - [Create an account for free \(https://azure.microsoft.com/free/?WT.mc\\_id=A261C142F\)](https://azure.microsoft.com/free/?WT.mc_id=A261C142F)
2. An Azure AI ML workspace - [Create workspace resources you need to get started with Azure AI \(https://learn.microsoft.com/en-us/azure/machine-learning/quickstart-create-resources\)](https://learn.microsoft.com/en-us/azure/machine-learning/quickstart-create-resources).
3. A python environment, python=3.9 or higher version like 3.10 is recommended.
4. Install promptflow with extra dependencies and promptflow-tools.

```
pip install promptflow[azure] promptflow-tools
```

5. Clone the sample repo and check flows in folder [examples/flows \(https://github.com/microsoft/promptflow/tree/main/examples/flows\)](https://github.com/microsoft/promptflow/tree/main/examples/flows).

```
git clone https://github.com/microsoft/promptflow.git
```

## Create necessary connections

Connection helps securely store and manage secret keys or other sensitive credentials required for interacting with LLM and other external tools for example Azure Content Safety.

In this guide, we will use flow web-classification which uses connection open\_ai\_connection inside, we need to set up the connection if we haven't added it before.

Please go to workspace portal, click Prompt flow -> Connections -> Create, then follow the instruction to create your own connections. Learn more on [connections \(https://learn.microsoft.com/en-us/azure/machine-learning/prompt-flow/concept-connections?view=azureml-api-2\)](https://learn.microsoft.com/en-us/azure/machine-learning/prompt-flow/concept-connections?view=azureml-api-2).

## Submit a run to workspace

Assuming you are in working directory <path-to-the-sample-repo>/examples/flows/standard/

:sync: CLI

Use `az login` to login so promptflow can get your credential.

```
az login
```

Submit a run to workspace.

```
pfazure run create --subscription <my_sub> -g <my_resource_group> -w <my_workspace> --flow web-classification --data web-classification/data.jsonl --stream --runtime <my-runtime>
```

### Default subscription/resource-group/workspace

Note --subscription, -g and -w can be omitted if you have installed the [Azure CLI \(https://learn.microsoft.com/en-us/cli/azure/install-azure-cli\)](https://learn.microsoft.com/en-us/cli/azure/install-azure-cli) and [set the default configurations \(https://learn.microsoft.com/en-us/cli/azure/azure-cli-configuration\)](https://learn.microsoft.com/en-us/cli/azure/azure-cli-configuration).

```
az account set --subscription <my-sub>
az configure --defaults group=<my_resource_group> workspace=<my_workspace>
```

### Serverless runtime and named runtime

Runtimes serve as computing resources so that the flow can be executed in workspace. Above command does not specify any runtime which means it will run in serverless mode. In this mode the workspace will automatically create a runtime and you can use it as the default runtime for any flow run later.

Instead, you can also [create a runtime \(https://learn.microsoft.com/en-us/azure/machine-learning/prompt-flow/how-to-create-manage-runtime?view=azureml-api-2\)](https://learn.microsoft.com/en-us/azure/machine-learning/prompt-flow/how-to-create-manage-runtime?view=azureml-api-2) and use it with --

runtime <my-runtime>:

```
pfazure run create --flow web-classification --data web-classification/data.jsonl --stream --runtime <my-runtime>
```

### Specify run name and view a run

You can also name the run by specifying `--name my_first_cloud_run` in the run create command, otherwise the run name will be generated in a certain pattern which has timestamp inside.

With a run name, you can easily stream or view the run details using below commands:

```
pfazure run stream -n my_first_cloud_run # same as "--stream" in command "run create"
pfazure run show-details -n my_first_cloud_run
pfazure run visualize -n my_first_cloud_run
```

More details can be found in [CLI reference: pfazure \(./../reference/pfazure-command-reference.md\)](#).

:sync: SDK

#### 1. Import the required libraries

```
from azure.identity import DefaultAzureCredential, InteractiveBrowserCredential
azure version promptflow apis
from promptflow.azure import PFClient
```

#### 2. Get credential

```
try:
 credential = DefaultAzureCredential()
 # Check if given credential can get token successfully.
 credential.get_token("https://management.azure.com/.default")
except Exception as ex:
 # Fall back to InteractiveBrowserCredential in case DefaultAzureCredential not work
 credential = InteractiveBrowserCredential()
```

#### 3. Get a handle to the workspace

```
Get a handle to workspace
pf = PFClient(
 credential=credential,
 subscription_id="<SUBSCRIPTION_ID>", # this will look like xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
 resource_group_name="<RESOURCE_GROUP>",
 workspace_name="<AML_WORKSPACE_NAME>",
)
```

#### 4. Submit the flow run

```
load flow
flow = "web-classification"
data = "web-classification/data.jsonl"
runtime = "example-runtime-ci" # assume you have existing runtime with this name provisioned
runtime = None # un-comment use automatic runtime

create run
base_run = pf.run(
 flow=flow,
 data=data,
 runtime=runtime,
)

pf.stream(base_run)
```

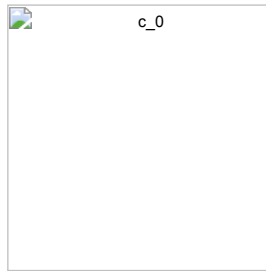
#### 5. View the run info

```
details = pf.get_details(base_run)
details.head(10)

pf.visualize(base_run)
```

# View the run in workspace

At the end of stream logs, you can find the `portal_url` of the submitted run, click it to view the run in the workspace.



## Run snapshot of the flow with additional includes

Flows that enabled [additional include](#) ([./../how-to-guides/develop-a-flow/referencing-external-files-or-folders-in-a-flow.md](#)) files can also be submitted for execution in the workspace. Please note that the specific additional include files or folders will be uploaded and organized within the **Files** folder of the run snapshot in the cloud.



## Next steps

Learn more about:

- [CLI reference: pfazure](#) ([./../reference/pfazure-command-reference.md](#)).

```
:maxdepth: 1
:hidden:

create-run-with-automatic-runtime
```

filepath: promptflow/docs/cloud/azureai/quick-start/create-run-with-automatic-runtime.md content: # Create run with automatic runtime

A prompt flow runtime provides computing resources that are required for the application to run, including a Docker image that contains all necessary dependency packages. This reliable and scalable runtime environment enables prompt flow to efficiently execute its tasks and functions for a seamless user experience.

If you're a new user, we recommend that you use the automatic runtime (preview). You can easily customize the environment by adding packages in the `requirements.txt` file in `flow.dag.yaml` in the flow folder.

## Create a run with automatic runtime

Create a run with automatic runtime is simple, just omit the `runtime` field and system will use automatic runtime to create a session to execute.

:sync: CLI

```
pfazure run create --flow path/to/flow --data path/to/data --stream
```

:sync: SDK

```

from promptflow.azure import PFClient

pf = PFClient(
 credential=credential,
 subscription_id="<SUBSCRIPTION_ID>", # this will look like xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
 resource_group_name="<RESOURCE_GROUP>",
 workspace_name="<AML_WORKSPACE_NAME>",
)

pf.run(
 flow=flow,
 data=data,
)

```

## Specify pip requirements for automatic runtime

If `requirements.txt` exists in the same folder with `flow.dag.yaml`. The dependencies in it will be automatically installed for automatic runtime.

You can also specify which requirements file to use in `flow.dag.yaml` like this:

```

$schema: https://azuremlschemas.azureedge.net/promptflow/latest/Flow.schema.json
environment:
 python_requirements_txt: path/to/requirement/file
...

```

Reference [Flow YAML Schema \(./../reference/flow-yaml-schema-reference.md\)](#) for details.

## Customize automatic runtime

In automatic runtime case, you can also specify the instance type, if you don't specify the instance type, Azure Machine Learning chooses an instance type (VM size) based on factors like quota, cost, performance and disk size, learn more about [serverless compute \(https://docs.microsoft.com/en-us/azure/machine-learning/how-to-use-serverless-compute\)](https://docs.microsoft.com/en-us/azure/machine-learning/how-to-use-serverless-compute).

```

$schema: https://azuremlschemas.azureedge.net/promptflow/latest/Run.schema.json
flow: <path_to_flow>
data: <path_to_flow>/data.jsonl

column_mapping:
 url: ${data.url}

define instance type only work for automatic runtime, will be ignored if you specify the runtime name.
resources:
 instance_type: <instance_type>

```

:sync: CLI

```
pfazure run create --file run.yaml
```

:sync: SDK

```

from promptflow import load_run

run = load_run(source="run.yaml")
pf = PFClient(
 credential=credential,
 subscription_id="<SUBSCRIPTION_ID>", # this will look like xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
 resource_group_name="<RESOURCE_GROUP>",
 workspace_name="<AML_WORKSPACE_NAME>",
)

pf.runs.create_or_update(
 run=run
)

```

# Next steps

- Try the example [here \(https://github.com/microsoft/promptflow/blob/main/examples/tutorials/run-management/cloud-run-management.ipynb\)](https://github.com/microsoft/promptflow/blob/main/examples/tutorials/run-management/cloud-run-management.ipynb).

filepath: promptflow/docs/reference/flow-yaml-schema-reference.md content: # Flow YAML Schema

This is an experimental feature, and may change at any time. Learn [more \(../how-to-guides/fag.md#stable-vs-experimental\)](#).

The source JSON schema can be found at [Flow.schema.json \(https://azuremschemas.azureedge.net/promptflow/latest/Flow.schema.json\)](#).

## YAML syntax

Key	Type	Description
\$schema	string	The YAML schema. If you use the prompt flow VS Code extension to author the YAML file, including <code>\$schema</code> at the top of your file enables you to invoke schema and resource completions.
inputs	object	Dictionary of flow inputs. The key is a name for the input within the context of the flow and the value is the flow input definition.
inputs.<input_name>	object	The flow input definition. See <a href="#">Flow input</a> for the set of configurable properties.
outputs	object	Dictionary of flow outputs. The key is a name for the output within the context of the flow and the value is the flow output definition.
outputs.<output_name>	object	The component output definition. See <a href="#">Flow output</a> for the set of configurable properties.
nodes	array	Sets of dictionary of individual nodes to run as steps within the flow. Node can use built-in tool or third-party tool. See <a href="#">Nodes</a> for more information.
node_variants	object	Dictionary of nodes with variants. The key is the node name and value contains variants definition and <code>default_variant_id</code> . See <a href="#">Node variants</a> for more information.
environment	object	The environment to use for the flow. The key can be <code>image</code> or <code>python_requirements_txt</code> and the value can be either a image or a python requirements text file.
additional_includes	array	Additional includes is a list of files that can be shared among flows. Users can specify additional files and folders used by flow, and prompt flow will help copy them all to the snapshot during flow creation.

## Flow input

Key	Type	Description	Allowed values
type	string	The type of flow input.	int, double, bool, string, list, object, image
description	string	Description of the input.	
default	int, double, bool, string, list, object, image	The default value for the input.	
is_chat_input	boolean	Whether the input is the chat flow input.	
is_chat_history	boolean	Whether the input is the chat history for chat flow.	

## Flow output

Key	Type	Description	Allowed values
type	string	The type of flow output.	int, double, bool, string, list, object
description	string	Description of the output.	
reference	string	A reference to the node output, e.g. <code>\${&lt;node_name&gt;.output.&lt;node_output_name&gt;}</code>	
is_chat_output	boolean	Whether the output is the chat flow output.	

## Nodes

Nodes is a set of node which is a dictionary with following fields. Below, we only show the common fields of a single node using built-in tool.

Key	Type	Description	Allowed values
name	string	The name of the node.	
type	string	The type of the node.	Type of built-in tool like Python, Prompt, LLM and third-party tool like Vector Search, etc.
inputs	object	Dictionary of node inputs. The key is the input name and the value can be primitive value or a reference to the flow input or the node output, e.g. <code>\${inputs.&lt;flow_input_name&gt;}</code> , <code>\${&lt;node_name&gt;.output}</code> OR <code>\${&lt;node_name&gt;.output.&lt;node_output_name&gt;}</code>	
source	object	Dictionary of tool source used by the node. The key contains <code>type</code> , <code>path</code> and <code>tool</code> . The type can be <code>code</code> , <code>package</code> and <code>package_with_prompt</code> .	
provider	string	It indicates the provider of the tool. Used when the <code>type</code> is LLM.	AzureOpenAI OR OpenAI
connection	string	The connection name which has been created before. Used when the <code>type</code> is LLM.	

Key	Type	Description	Allowed values
api	string	The api name of the provider. Used when the <code>type</code> is LLM.	
module	string	The module name of the tool using by the node. Used when the <code>type</code> is LLM.	
use_variants	bool	Whether the node has variants.	

## Node variants

Node variants is a dictionary containing variants definition for nodes with variants with their respective node names as dictionary keys. Below, we explore the variants for a single node.

Key	Type	Description	Allowed values
<node_name>	string	The name of the node.	
default_variant_id	string	Default variant id.	
variants	object	This dictionary contains all node variations, with the variant id serving as the key and a node definition dictionary as the corresponding value. Within the node definition dictionary, the key labeled 'node' should contain a variant definition similar to <code>Nodes</code> , excluding the 'name' field.	

## Examples

Flow examples are available in the [GitHub repository \(https://github.com/microsoft/promptflow/tree/main/examples/flows\)](https://github.com/microsoft/promptflow/tree/main/examples/flows).

- [basic \(https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/basic\)](https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/basic).
- [web-classification \(https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification\)](https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification).
- [basic-chat \(https://github.com/microsoft/promptflow/tree/main/examples/flows/chat/basic-chat\)](https://github.com/microsoft/promptflow/tree/main/examples/flows/chat/basic-chat).
- [chat-with-pdf \(https://github.com/microsoft/promptflow/tree/main/examples/flows/chat/chat-with-pdf\)](https://github.com/microsoft/promptflow/tree/main/examples/flows/chat/chat-with-pdf).
- [eval-basic \(https://github.com/microsoft/promptflow/tree/main/examples/flows/evaluation/eval-basic\)](https://github.com/microsoft/promptflow/tree/main/examples/flows/evaluation/eval-basic).

filepath: promptflow/docs/reference/index.md content: # Reference

Current stable version:

pypi package

1.5.0



- [promptflow \(https://pypi.org/project/promptflow/\)](https://pypi.org/project/promptflow/): [. \(https://badge.fury.io/py/promptflow\)](https://badge.fury.io/py/promptflow)

pypi package

1.2.0

- [promptflow-tools \(https://pypi.org/project/promptflow-tools/\)](https://pypi.org/project/promptflow-tools/): [. \(https://badge.fury.io/py/promptflow-tools\)](https://badge.fury.io/py/promptflow-tools)



[. \(https://pypi.org/project/promptflow-tools/\)](https://pypi.org/project/promptflow-tools/)

```
:caption: Command Line Interface
:maxdepth: 1

pf-command-reference.md
pfazure-command-reference.md
```

```
:caption: Python Library Reference
:maxdepth: 4

python-library-reference/promptflow
```

```
:caption: Tool Reference
:maxdepth: 1

tools-reference/llm-tool
tools-reference/prompt-tool
tools-reference/python-tool
tools-reference/serp-api-tool
tools-reference/faiss_index_lookup_tool
tools-reference/vector_db_lookup_tool
tools-reference/embedding_tool
tools-reference/open_model_llm_tool
tools-reference/openai-gpt-4v-tool
tools-reference/contentsafety_text_tool
tools-reference/aoai-gpt4-turbo-vision
```

```
:caption: YAML Schema
:maxdepth: 1

flow-yaml-schema-reference.md
run-yaml-schema-reference.md
```

filepath: promptflow/docs/reference/run-yaml-schema-reference.md content: # Run YAML Schema

This is an experimental feature, and may change at any time. Learn [more](#) ([../how-to-guides/faq.md#stable-vs-experimental](#)).

The source JSON schema can be found at [Run.schema.json](#) (<https://azuremlschemas.azureedge.net/promptflow/latest/Run.schema.json>)

## YAML syntax

Key	Type	Description
\$schema	string	The YAML schema. If you use the prompt flow VS Code extension to author the YAML file, including \$schema at the top of your file enables you to invoke schema and resource completions.
name	string	The name of the run.
flow	string	Path of the flow directory.
description	string	Description of the run.
display_name	string	Display name of the run.
runtime	string	The runtime for the run. Only supported for cloud run.
data	string	Input data for the run. Local path or remote uri(starts with azureml: or public URL) are supported. Note: remote uri is only supported for cloud run.
run	string	Referenced flow run name. For example, you can run an evaluation flow against an existing run.
column_mapping	object	Inputs column mapping, use \${data.xx} to refer to data columns, use \${run.inputs.xx} to refer to referenced run's data columns, and \${run.outputs.xx} to refer to run outputs columns.
connections	object	Overwrite node level connections with provided value. Example: --connections node1.connection=test_llm_connection node1.deployment_name=gpt-35-turbo
environment_variables	object/string	Environment variables to set by specifying a property path and value. Example: {"key1"="\${my_connection.api_key}"}. The value reference to connection keys will be resolved to the actual value, and all environment variables specified will be set into os.environ.
properties	object	Dictionary of properties of the run.
tags	object	Dictionary of tags of the run.
resources	object	Dictionary of resources used for automatic runtime. Only supported for cloud run. See <a href="#">Resources Schema</a> for the set of configurable properties.
variant	string	The variant for the run.

Key	Type	Description
status	string	The status of the run. Only available for when getting an existing run. Won't take affect if set when creating a run.

## Resources Schema

Key	Type	Description
instance_type	string	The instance type for automatic runtime of the run.
compute	string	The compute instance for automatic runtime session.

## Examples

Run examples are available in the [GitHub repository \(https://github.com/microsoft/promptflow/tree/main/examples/flows\)](https://github.com/microsoft/promptflow/tree/main/examples/flows).

- [basic \(https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/basic/run.yml\)](https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/basic/run.yml)
- [web-classification \(https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification/run.yml\)](https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification/run.yml)
- [flow-with-additional-includes \(https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/flow-with-additional-includes/run.yml\)](https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/flow-with-additional-includes/run.yml)

filepath: promptflow/docs/reference/pfazure-command-reference.md content: # pfazure

This is an experimental feature, and may change at any time. Learn [more \(../how-to-guides/fag.md#stable-vs-experimental\)](#).

Manage prompt flow resources on Azure with the prompt flow CLI.

Command	Description
---------	-------------

<code>pfazure flow</code>	Manage flows.
---------------------------	---------------

<code>pfazure run</code>	Manage runs.
--------------------------	--------------

## pfazure flow

Manage flows.

Command	Description
<code>pfazure flow create</code>	Create a flow.
<code>pfazure flow list</code>	List flows in a workspace.

## pfazure flow create

Create a flow in Azure AI from a local flow folder.

```
pfazure flow create [--flow]
 [--set]
 [--subscription]
 [--resource-group]
 [--workspace-name]
```

### Parameters

`--flow`

Local path to the flow directory.

`--set`

Update an object by specifying a property path and value to set.

- `display_name`: Flow display name that will be created in remote. Default to be flow folder name + timestamp if not specified.
- `type`: Flow type. Default to be "standard" if not specified. Available types are: "standard", "evaluation", "chat".
- `description`: Flow description. e.g. "`--set description=<description>`".
- `tags`: Flow tags. e.g. "`--set tags.key1=value1 tags.key2=value2`".

`--subscription`

Subscription id, required when there is no default value from `az configure`.

`--resource-group -g`

Resource group name, required when there is no default value from `az configure`.



--workspace-name -w

Workspace name, required when there is no default value from `az configure`.

## pfazure flow list

List remote flows on Azure AI.

```
pfazure flow list [--max-results]
 [--include-others]
 [--type]
 [--output]
 [--archived-only]
 [--include-archived]
 [--subscription]
 [--resource-group]
 [--workspace-name]
 [--output]
```

### Parameters

--max-results -r

Max number of results to return. Default is 50, upper bound is 100.

--include-others

Include flows created by other owners. By default only flows created by the current user are returned.

--type

Filter flows by type. Available types are: "standard", "evaluation", "chat".

--archived-only

List archived flows only.

--include-archived

List archived flows and active flows.

--output -o

Output format. Allowed values: json, table. Default: json.

--subscription

Subscription id, required when there is no default value from `az configure`.

--resource-group -g

Resource group name, required when there is no default value from `az configure`.

--workspace-name -w

Workspace name, required when there is no default value from `az configure`.

## pfazure run

Manage prompt flow runs.

Command	Description
<a href="#">pfazure run create</a>	Create a run.
<a href="#">pfazure run list</a>	List runs in a workspace.
<a href="#">pfazure run show</a>	Show details for a run.
<a href="#">pfazure run stream</a>	Stream run logs to the console.
<a href="#">pfazure run show-details</a>	Show a run details.
<a href="#">pfazure run show-metrics</a>	Show run metrics.
<a href="#">pfazure run visualize</a>	Visualize a run.
<a href="#">pfazure run archive</a>	Archive a run.
<a href="#">pfazure run restore</a>	Restore a run.

Command	Description
<code>pfazure run update</code>	Update a run.
<code>pfazure run download</code>	Download a run.

## pfazure run create

Create a run.

```
pfazure run create [--file]
 [--flow]
 [--data]
 [--column-mapping]
 [--run]
 [--variant]
 [--stream]
 [--environment-variables]
 [--connections]
 [--set]
 [--subscription]
 [--resource-group]
 [--workspace-name]
```

### Parameters

`--file -f`

Local path to the YAML file containing the prompt flow run specification; can be overwritten by other parameters. Reference [here](https://azuremlschemas.azureedge.net/promptflow/latest/Run.schema.json) (<https://azuremlschemas.azureedge.net/promptflow/latest/Run.schema.json>) for YAML schema.

`--flow`

Local path to the flow directory.

`--data`

Local path to the data file or remote data. e.g. azureml:name:version.

`--column-mapping`

Inputs column mapping, use `$(data.xx)` to refer to data columns, use `$(run.inputs.xx)` to refer to referenced run's data columns, and `$(run.outputs.xx)` to refer to run outputs columns.

`--run`

Referenced flow run name. For example, you can run an evaluation flow against an existing run. For example, "pfazure run create --flow evaluation\_flow\_dir --run existing\_bulk\_run --column-mapping url=\$".

`--variant`

Node & variant name in format of `$(node_name.variant_name)`.

`--stream -s`

Indicates whether to stream the run's logs to the console.  
default value: False

`--environment-variables`

Environment variables to set by specifying a property path and value. Example: `--environment-variable key1='${my_connection.api_key}' key2='value2'`. The value reference to connection keys will be resolved to the actual value, and all environment variables specified will be set into `os.environ`.

`--connections`

Overwrite node level connections with provided value. Example: `--connections node1.connection=test_llm_connection node1.deployment_name=gpt-35-turbo`

`--set`

Update an object by specifying a property path and value to set. Example: `--set property1.property2=<value>`.

`--subscription`

Subscription id, required when there is no default value from `az configure`.

`--resource-group -g`

Resource group name, required when there is no default value from `az configure`.

`--workspace-name -w`

Workspace name, required when there is no default value from `az configure`.

## pfazure run list

List runs in a workspace.

```
pfazure run list [--archived-only]
 [--include-archived]
 [--max-results]
 [--subscription]
 [--resource-group]
 [--workspace-name]
```

### Parameters

`--archived-only`

List archived runs only.

default value: False

`--include-archived`

List archived runs and active runs.

default value: False

`--max-results -r`

Max number of results to return. Default is 50, upper bound is 100.

default value: 50

`--subscription`

Subscription id, required when there is no default value from `az configure`.

`--resource-group -g`

Resource group name, required when there is no default value from `az configure`.

`--workspace-name -w`

Workspace name, required when there is no default value from `az configure`.

## pfazure run show

Show details for a run.

```
pfazure run show --name
 [--subscription]
 [--resource-group]
 [--workspace-name]
```

### Parameters

`--name -n`

Name of the run.

`--subscription`

Subscription id, required when there is no default value from `az configure`.

`--resource-group -g`

Resource group name, required when there is no default value from `az configure`.

`--workspace-name -w`

Workspace name, required when there is no default value from `az configure`.

## pfazure run stream

Stream run logs to the console.

```
pfazure run stream --name
 [--subscription]
 [--resource-group]
 [--workspace-name]
```

### Parameters

`--name -n`

Name of the run.

`--subscription`

Subscription id, required when there is no default value from `az configure`.

`--resource-group -g`

Resource group name, required when there is no default value from `az configure`.

`--workspace-name -w`

Workspace name, required when there is no default value from `az configure`.

## pfazure run show-details

Show a run details.

```
pfazure run show-details --name
 [--subscription]
 [--resource-group]
 [--workspace-name]
```

### Parameters

`--name -n`

Name of the run.

`--subscription`

Subscription id, required when there is no default value from `az configure`.

`--resource-group -g`

Resource group name, required when there is no default value from `az configure`.

`--workspace-name -w`

Workspace name, required when there is no default value from `az configure`.

## pfazure run show-metrics

Show run metrics.

```
pfazure run show-metrics --name
 [--subscription]
 [--resource-group]
 [--workspace-name]
```

## Parameters

`--name -n`

Name of the run.

`--subscription`

Subscription id, required when there is no default value from `az configure`.

`--resource-group -g`

Resource group name, required when there is no default value from `az configure`.

`--workspace-name -w`

Workspace name, required when there is no default value from `az configure`.

## pfazure run visualize

Visualize a run.

```
pfazure run visualize --name
 [--subscription]
 [--resource-group]
 [--workspace-name]
```

## Parameters

`--name -n`

Name of the run.

`--subscription`

Subscription id, required when there is no default value from `az configure`.

`--resource-group -g`

Resource group name, required when there is no default value from `az configure`.

`--workspace-name -w`

Workspace name, required when there is no default value from `az configure`.

## pfazure run archive

Archive a run.

```
pfazure run archive --name
 [--subscription]
 [--resource-group]
 [--workspace-name]
```

## Parameters

`--name -n`

Name of the run.

`--subscription`

Subscription id, required when there is no default value from `az configure`.

`--resource-group -g`

Resource group name, required when there is no default value from `az configure`.

`--workspace-name -w`

Workspace name, required when there is no default value from `az configure`.

# pfazure run restore

Restore a run.

```
pfazure run restore --name
 [--subscription]
 [--resource-group]
 [--workspace-name]
```

## Parameters

--name -n

Name of the run.

--subscription

Subscription id, required when there is no default value from az configure.

--resource-group -g

Resource group name, required when there is no default value from az configure.

--workspace-name -w

Workspace name, required when there is no default value from az configure.

# pfazure run update

Update a run's metadata, such as display name, description and tags.

```
pfazure run update --name
 [--set display_name="<value>" description="<value>" tags.key="<value>"]
 [--subscription]
 [--resource-group]
 [--workspace-name]
```

## Examples

Set display name, description and tags:

```
pfazure run update --name <run_name> --set display_name="<value>" description="<value>" tags.key="<value>"
```

## Parameters

--name -n

Name of the run.

--set

Set meta information of the run, like display\_name, description or tags. Example: --set =.

--subscription

Subscription id, required when there is no default value from az configure.

--resource-group -g

Resource group name, required when there is no default value from az configure.

--workspace-name -w

Workspace name, required when there is no default value from az configure.

# pfazure run download

Download a run's metadata, such as input, output, snapshot and artifact. After the download is finished, you can use pf run create --source <run-info-local-folder> to register this run as a local run record, then you can use commands like pf run show/visualize to inspect the run just like a run that was created from local flow.

```
pfazure run download --name
 [--output]
 [--overwrite]
 [--subscription]
 [--resource-group]
 [--workspace-name]
```

Examples

Download a run data to local:

```
pfazure run download --name <name> --output <output-folder-path>
```

Parameters

--name -n

Name of the run.

--output -o

Output folder path to store the downloaded run data. Default to be ~/.promptflow/.runs if not specified

--overwrite

Overwrite the existing run data if the output folder already exists. Default to be False if not specified

--subscription

Subscription id, required when there is no default value from az configure.

--resource-group -g

Resource group name, required when there is no default value from az configure.

--workspace-name -w

Workspace name, required when there is no default value from az configure.

filepath: promptflow/docs/reference/pf-command-reference.md content: # pf

This is an experimental feature, and may change at any time. Learn [more \(../how-to-guides/fag.md#stable-vs-experimental\)](#).

Manage prompt flow resources with the prompt flow CLI.

Command	Description
<a href="#">pf flow</a>	Manage flows.
<a href="#">pf connection</a>	Manage connections.
<a href="#">pf run</a>	Manage runs.
<a href="#">pf tool</a>	Init or list tools.
<a href="#">pf config</a>	Manage config for current user.
<a href="#">pf upgrade</a>	Upgrade prompt flow CLI.

pf flow

Manage promptflow flow flows.

Command	Description
<a href="#">pf flow init</a>	Initialize a prompt flow directory.
<a href="#">pf flow test</a>	Test the prompt flow or flow node.
<a href="#">pf flow validate</a>	Validate a flow and generate flow.tools.json for it.
<a href="#">pf flow build</a>	Build a flow for further sharing or deployment.
<a href="#">pf flow serve</a>	Serve a flow as an endpoint.

pf flow init

Initialize a prompt flow directory.

```
pf flow init [--flow]
 [--entry]
 [--function]
 [--prompt-template]
 [--type]
 [--yes]
```

## Examples

Create a flow folder with code, prompts and YAML specification of the flow.

```
pf flow init --flow <path-to-flow-directory>
```

Create an evaluation prompt flow

```
pf flow init --flow <path-to-flow-directory> --type evaluation
```

Create a flow in existing folder

```
pf flow init --flow <path-to-existing-folder> --entry <entry.py> --function <function-name> --prompt-template <path-to-prompt-template>
```

## Optional Parameters

`--flow`

The flow name to create.

`--entry`

The entry file name.

`--function`

The function name in entry file.

`--prompt-template`

The prompt template parameter and assignment.

`--type`

The initialized flow type.

accepted value: standard, evaluation, chat

`--yes --assume-yes -y`

Automatic yes to all prompts; assume 'yes' as answer to all prompts and run non-interactively.

## pf flow test

Test the prompt flow or flow node.

```
pf flow test --flow
 [--inputs]
 [--node]
 [--variant]
 [--debug]
 [--interactive]
 [--verbose]
```

## Examples

Test the flow.

```
pf flow test --flow <path-to-flow-directory>
```

Test the flow with single line from input file.



```
pf flow test --flow <path-to-flow-directory> --inputs data_key1=data_val1 data_key2=data_val2
```

Test the flow with specified variant node.

```
pf flow test --flow <path-to-flow-directory> --variant '${node_name.variant_name}'
```

Test the single node in the flow.

```
pf flow test --flow <path-to-flow-directory> --node <node_name>
```

Debug the single node in the flow.

```
pf flow test --flow <path-to-flow-directory> --node <node_name> --debug
```

Chat in the flow.

```
pf flow test --flow <path-to-flow-directory> --node <node_name> --interactive
```

## Required Parameter

`--flow`

The flow directory to test.

## Optional Parameters

`--inputs`

Input data for the flow. Example: `--inputs data1=data1_val data2=data2_val`

`--node`

The node name in the flow need to be tested.

`--variant`

Node & variant name in format of \$.

`--debug`

Debug the single node in the flow.

`--interactive`

Start a interactive chat session for chat flow.

`--verbose`

Displays the output for each step in the chat flow.

## pf flow validate

Validate the prompt flow and generate a `flow.tools.json` under `.promptflow`. This file is required when using flow as a component in a Azure ML pipeline.

```
pf flow validate --source
 [--debug]
 [--verbose]
```

## Examples

Validate the flow.

```
pf flow validate --source <path-to-flow>
```

## Required Parameter

`--source`

The flow source to validate.

## pf flow build

Build a flow for further sharing or deployment.

```
pf flow build --source
 --output
 --format
 [--variant]
 [--verbose]
 [--debug]
```

### Examples

Build a flow as docker, which can be built into Docker image via `docker build`.

```
pf flow build --source <path-to-flow> --output <output-path> --format docker
```

Build a flow as docker with specific variant.

```
pf flow build --source <path-to-flow> --output <output-path> --format docker --variant '${node_name.variant_name}'
```

### Required Parameter

`--source`

The flow or run source to be used.

`--output`

The folder to output built flow. Need to be empty or not existed.

`--format`

The format to build flow into

### Optional Parameters

`--variant`

Node & variant name in format of \$.

`--verbose`

Show more details for each step during build.

`--debug`

Show debug information during build.

## pf flow serve

Serving a flow as an endpoint.

```
pf flow serve --source
 [--port]
 [--host]
 [--environment-variables]
 [--verbose]
 [--debug]
 [--skip-open-browser]
```

### Examples

Serve flow as an endpoint.

```
pf flow serve --source <path-to-flow>
```

Serve flow as an endpoint with specific port and host.

```
pf flow serve --source <path-to-flow> --port <port> --host <host> --environment-variables key1="`${my_connection.api_key}`" key2="value2"
```

Required Parameter

--source

The flow or run source to be used.

Optional Parameters

--port

The port on which endpoint to run.

--host

The host of endpoint.

--environment-variables

Environment variables to set by specifying a property path and value. Example: --environment-variable key1="`\${my\_connection.api\_key}`" key2="value2". The value reference to connection keys will be resolved to the actual value, and all environment variables specified will be set into `os.environ`.

--verbose

Show more details for each step during serve.

--debug

Show debug information during serve.

--skip-open-browser

Skip opening browser after serve. Store true parameter.

pf connection

Manage prompt flow connections.

Command	Description
<u>pf connection create</u>	Create a connection.
<u>pf connection update</u>	Update a connection.
<u>pf connection show</u>	Show details of a connection.
<u>pf connection list</u>	List all the connection.
<u>pf connection delete</u>	Delete a connection.

pf connection create

Create a connection.

```
pf connection create --file
 [--name]
 [--set]
```

Examples

Create a connection with YAML file.

```
pf connection create -f <yaml-filename>
```

Create a connection with YAML file with override.

```
pf connection create -f <yaml-filename> --set api_key="<api-key>"
```

Create a custom connection with .env file; note that overrides specified by --set will be ignored.

```
pf connection create -f .env --name <name>
```

#### Required Parameter

--file -f

Local path to the YAML file containing the prompt flow connection specification.

#### Optional Parameters

--name -n

Name of the connection.

--set

Update an object by specifying a property path and value to set. Example: --set property1.property2=.

### pf connection update

Update a connection.

```
pf connection update --name
 [--set]
```

#### Example

Update a connection.

```
pf connection update -n <name> --set api_key="<api-key>"
```

#### Required Parameter

--name -n

Name of the connection.

#### Optional Parameter

--set

Update an object by specifying a property path and value to set. Example: --set property1.property2=.

### pf connection show

Show details of a connection.

```
pf connection show --name
```

#### Required Parameter

--name -n

Name of the connection.

### pf connection list

List all the connection.

```
pf connection list
```

### pf connection delete

Delete a connection.

```
pf connection delete --name
```

## Required Parameter

--name -n

Name of the connection.

# pf run

Manage prompt flow runs.

## Command

## Description

<u>pf run create</u>	Create a run.
<u>pf run update</u>	Update a run metadata, including display name, description and tags.
<u>pf run stream</u>	Stream run logs to the console.
<u>pf run list</u>	List runs.
<u>pf run show</u>	Show details for a run.
<u>pf run show-details</u>	Preview a run's input(s) and output(s).
<u>pf run show-metrics</u>	Print run metrics to the console.
<u>pf run visualize</u>	Visualize a run.
<u>pf run archive</u>	Archive a run.
<u>pf run restore</u>	Restore an archived run.

## pf run create

Create a run.

```
pf run create [--file]
 [--flow]
 [--data]
 [--column-mapping]
 [--run]
 [--variant]
 [--stream]
 [--environment-variables]
 [--connections]
 [--set]
 [--source]
```

## Examples

Create a run with YAML file.

```
pf run create -f <yaml-filename>
```

Create a run with YAML file and replace another data in the YAML file.

```
pf run create -f <yaml-filename> --data <path-to-new-data-file-relative-to-yaml-file>
```

Create a run from flow directory and reference a run.

```
pf run create --flow <path-to-flow-directory> --data <path-to-data-file> --column-mapping groundtruth='${data.answer}' prediction='${
```

Create a run from an existing run record folder.

```
pf run create --source <path-to-run-folder>
```

## Optional Parameters

--file -f

Local path to the YAML file containing the prompt flow run specification; can be overwritten by other parameters. Reference [here](https://azuremlschemas.azureedge.net/promptflow/latest/Run.schema.json) (<https://azuremlschemas.azureedge.net/promptflow/latest/Run.schema.json>) for YAML schema.

`--flow`

Local path to the flow directory. If `--file` is provided, this path should be relative path to the file.

`--data`

Local path to the data file. If `--file` is provided, this path should be relative path to the file.

`--column-mapping`

Inputs column mapping, use `$(data.xx)` to refer to data columns, use `$(run.inputs.xx)` to refer to referenced run's data columns, and `$(run.outputs.xx)` to refer to run outputs columns.

`--run`

Referenced flow run name. For example, you can run an evaluation flow against an existing run. For example, "pf run create --flow evaluation\_flow\_dir --run existing\_bulk\_run".

`--variant`

Node & variant name in format of `$(node_name.variant_name)`.

`--stream -s`

Indicates whether to stream the run's logs to the console.

default value: False

`--environment-variables`

Environment variables to set by specifying a property path and value. Example: `--environment-variable key1='${my_connection.api_key}' key2='value2'`. The value reference to connection keys will be resolved to the actual value, and all environment variables specified will be set into `os.environ`.

`--connections`

Overwrite node level connections with provided value. Example: `--connections node1.connection=test_llm_connection node1.deployment_name=gpt-35-turbo`

`--set`

Update an object by specifying a property path and value to set. Example: `--set property1.property2=<value>`.

`--source`

Local path to the existing run record folder.

## pf run update

Update a run metadata, including display name, description and tags.

```
pf run update --name
 [--set]
```

### Example

Update a run

```
pf run update -n <name> --set display_name="<display-name>" description="<description>" tags.key="value"
```

### Required Parameter

`--name -n`

Name of the run.

### Optional Parameter

`--set`

Update an object by specifying a property path and value to set. Example: `--set property1.property2=`.

## pf run stream

Stream run logs to the console.

```
pf run stream --name
```

### Required Parameter

`--name -n`

Name of the run.

## pf run list

List runs.

```
pf run list [--all-results]
 [--archived-only]
 [--include-archived]
 [--max-results]
```

### Optional Parameters

`--all-results`

Returns all results.

default value: False

`--archived-only`

List archived runs only.

default value: False

`--include-archived`

List archived runs and active runs.

default value: False

`--max-results -r`

Max number of results to return. Default is 50.

default value: 50

## pf run show

Show details for a run.

```
pf run show --name
```

### Required Parameter

`--name -n`

Name of the run.

## pf run show-details

Preview a run's input(s) and output(s).

```
pf run show-details --name
```

### Required Parameter

`--name -n`

Name of the run.

## pf run show-metrics

Print run metrics to the console.

```
pf run show-metrics --name
```

Required Parameter

--name -n

Name of the run.

pf run visualize

Visualize a run in the browser.

```
pf run visualize --names
```

Required Parameter

--names -n

Name of the runs, comma separated.

pf run archive

Archive a run.

```
pf run archive --name
```

Required Parameter

--name -n

Name of the run.

pf run restore

Restore an archived run.

```
pf run restore --name
```

Required Parameter

--name -n

Name of the run.

pf tool

Manage promptflow tools.

Command	Description
<a href="#">pf tool init</a>	Initialize a tool directory.
<a href="#">pf tool list</a>	List all tools in the environment.
<a href="#">pf tool validate</a>	Validate tools.

pf tool init

Initialize a tool directory.

```
pf tool init [--package]
 [--tool]
 [--set]
```

Examples

Creating a package tool from scratch.



```
pf tool init --package <package-name> --tool <tool-name>
```

Creating a package tool with extra info.

```
pf tool init --package <package-name> --tool <tool-name> --set icon=<icon-path> category=<tool-category> tags="{ '<key>': '<value>' }"
```

Creating a package tool from scratch.

```
pf tool init --package <package-name> --tool <tool-name>
```

Creating a python tool from scratch.

```
pf tool init --tool <tool-name>
```

## Optional Parameters

`--package`

The package name to create.

`--tool`

The tool name to create.

`--set`

Set extra information about the tool, like category, icon and tags. Example: `--set =`.

## pf tool list

List all tools in the environment.

```
pf tool list [--flow]
```

## Examples

List all package tool in the environment.

```
pf tool list
```

List all package tool and code tool in the flow.

```
pf tool list --flow <path-to-flow-direcotry>
```

## Optional Parameters

`--flow`

The flow directory.

## pf tool validate

Validate tool.

```
pf tool validate --source
```

## Examples

Validate single function tool.

```
pf tool validate --source <package-name>.<module-name>.<tool-function>
```

Validate all tool in a package tool.

```
pf tool validate --source <package-name>
```

Validate tools in a python script.

```
pf tool validate --source <path-to-tool-script>
```

Required Parameter

--source

The tool source to be used.

# pf config

Manage config for current user.

Command	Description
<a href="#">pf config set</a>	Set prompt flow configs for current user.
<a href="#">pf config show</a>	Show prompt flow configs for current user.

## pf config set

Set prompt flow configs for current user, configs will be stored at ~/.promptflow/pf.yaml.

```
pf config set
```

Examples

Config connection provider to azure workspace for current user.

```
pf config set connection.provider="azureml://subscriptions/<your-subscription>/resourceGroups/<your-resourcegroup>/providers/Microsoft
```

## pf config show

Show prompt flow configs for current user.

```
pf config show
```

Examples

Show prompt flow for current user.

```
pf config show
```

# pf upgrade

Upgrade prompt flow CLI.

Command	Description
<a href="#">pf upgrade</a>	Upgrade prompt flow CLI.

## Examples

Upgrade prompt flow without prompt and run non-interactively.

```
pf upgrade --yes
```

filepath: promptflow/docs/reference/tools-reference/prompt-tool.md content: # Prompt

# Introduction

The Prompt Tool in PromptFlow offers a collection of textual templates that serve as a starting point for creating prompts. These templates, based on the Jinja2 template engine, facilitate the definition of prompts. The tool proves useful when prompt tuning is required prior to feeding the prompts into the Language Model (LLM) model in PromptFlow.

## Inputs

Name	Type	Description	Required
prompt string		The prompt template in Jinja	Yes
Inputs -		List of variables of prompt template and its assignments -	

## Outputs

The prompt text parsed from the prompt + Inputs

## How to write Prompt?

1. Prepare jinja template. Learn more about [Jinja](https://jinja.palletsprojects.com/en/3.1.x/) (<https://jinja.palletsprojects.com/en/3.1.x/>).

*In below example, the prompt incorporates Jinja templating syntax to dynamically generate the welcome message and personalize it based on the user's name. It also presents a menu of options for the user to choose from. Depending on whether the user\_name variable is provided, it either addresses the user by name or uses a generic greeting.*

```
Welcome to {{ website_name }}!
{% if user_name %}
 Hello, {{ user_name }}!
{% else %}
 Hello there!
{% endif %}
Please select an option from the menu below:
1. View your account
2. Update personal information
3. Browse available products
4. Contact customer support
```

2. Assign value for the variables.

*In above example, two variables would be automatically detected and listed in '**Inputs**' section. Please assign values.*

### Sample 1

Inputs

Variable	Type	Sample Value
website_name	string	"Microsoft"
user_name	string	"Jane"

Outputs

```
Welcome to Microsoft! Hello, Jane! Please select an option from the menu below: 1. View your account 2. Update personal information 3. Browse available products 4. Contact customer support
```

### Sample 2

Inputs

Variable	Type	Sample Value
website_name	string	"Bing"
user_name	string	"

Outputs

```
Welcome to Bing! Hello there! Please select an option from the menu below: 1. View your account 2. Update personal information 3. Browse available products 4. Contact customer support
```

filepath: promptflow/docs/reference/tools-reference/openai-gpt-4v-tool.md content: # OpenAI GPT-4V

## Introduction

OpenAI GPT-4V tool enables you to leverage OpenAI's GPT-4 with vision, also referred to as GPT-4V or gpt-4-vision-preview in the API, to take images as input and answer questions about them.

## Prerequisites

- Create OpenAI resources

Sign up account [OpenAI website \(https://openai.com/\)](https://openai.com/) Login and [Find personal API key \(https://platform.openai.com/account/api-keys\)](https://platform.openai.com/account/api-keys)

- Get Access to GPT-4 API

To use GPT-4 with vision, you need access to GPT-4 API. Learn more about [How to get access to GPT-4 API \(https://help.openai.com/en/articles/7102672-how-can-i-access-gpt-4\)](https://help.openai.com/en/articles/7102672-how-can-i-access-gpt-4)

## Connection

Setup connections to provisioned resources in prompt flow.

Type	Name	API KEY
OpenAI	Required	Required

## Inputs

Name	Type	Description	Required
connection	OpenAI	the OpenAI connection to be used in the tool	Yes
model	string	the language model to use, currently only support gpt-4-vision-preview	Yes
prompt	string	The text prompt that the language model will use to generate it's response.	Yes
max_tokens	integer	the maximum number of tokens to generate in the response. Default is 512.	No
temperature	float	the randomness of the generated text. Default is 1.	No
stop	list	the stopping sequence for the generated text. Default is null.	No
top_p	float	the probability of using the top choice from the generated tokens. Default is 1.	No
presence_penalty	float	value that controls the model's behavior with regards to repeating phrases. Default is 0.	No
frequency_penalty	float	value that controls the model's behavior with regards to generating rare phrases. Default is 0.	No

## Outputs

Return Type	Description
-------------	-------------

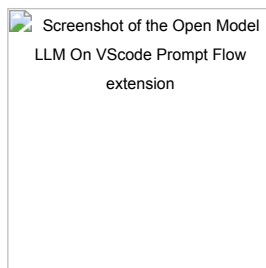
string	The text of one response of conversation
--------	------------------------------------------

filepath: promptflow/docs/reference/tools-reference/open\_model\_llm\_tool.md content: # Open Model LLM

## Introduction

The Open Model LLM tool enables the utilization of a variety of Open Model and Foundational Models, such as [Falcon \(https://ml.azure.com/models/tiiuae-falcon-7b/version/4/catalog/registry/azureml\)](https://ml.azure.com/models/tiiuae-falcon-7b/version/4/catalog/registry/azureml), and [Llama 2 \(https://ml.azure.com/models/Llama-2-7b-chat/version/14/catalog/registry/azureml-meta\)](https://ml.azure.com/models/Llama-2-7b-chat/version/14/catalog/registry/azureml-meta), for natural language processing in Azure ML Prompt Flow.

Here's how it looks in action on the Visual Studio Code prompt flow extension. In this example, the tool is being used to call a LLaMa-2 chat endpoint and asking "What is CI?".



This prompt flow tool supports two different LLM API types:

- **Chat:** Shown in the example above. The chat API type facilitates interactive conversations with text-based inputs and responses.
- **Completion:** The Completion API type is used to generate single response text completions based on provided prompt input.

# Quick Overview: How do I use Open Model LLM Tool?

1. Choose a Model from the AzureML Model Catalog and get it deployed.
2. Connect to the model deployment.
3. Configure the open model llm tool settings.
4. Prepare the Prompt with [guidance](#) ([./prompt-tool.md#how-to-write-prompt](#)).
5. Run the flow.

## Prerequisites: Model Deployment

1. Pick the model which matched your scenario from the [Azure Machine Learning model catalog](https://ml.azure.com/model/catalog) (<https://ml.azure.com/model/catalog>).
2. Use the "Deploy" button to deploy the model to a AzureML Online Inference endpoint. 2.1. Use one of the Pay as you go deployment options.

More detailed instructions can be found here [Deploying foundation models to endpoints for inferencing](https://learn.microsoft.com/en-us/azure/machine-learning/how-to-use-foundation-models?view=azureml-api-2#deploying-foundation-models-to-endpoints-for-inferencing), (<https://learn.microsoft.com/en-us/azure/machine-learning/how-to-use-foundation-models?view=azureml-api-2#deploying-foundation-models-to-endpoints-for-inferencing>).

## Prerequisites: Connect to the Model

In order for prompt flow to use your deployed model, you will need to connect to it. There are several ways to connect.

### 1. Endpoint Connections

Once associated to a AzureML or Azure AI Studio workspace, the Open Model LLM tool can use the endpoints on that workspace.

1. **Using AzureML or Azure AI Studio workspaces:** If you are using prompt flow in one of the web page based browsers workspaces, the online endpoints available on that workspace will automatically show up.
2. **Using VScode or Code First:** If you are using prompt flow in VScode or one of the Code First offerings, you will need to connect to the workspace. The Open Model LLM tool uses the azure.identity DefaultAzureCredential client for authorization. One way is through [setting environment credential values](https://learn.microsoft.com/en-us/python/api/azure-identity/azure.identity.environmentcredential?view=azure-python) (<https://learn.microsoft.com/en-us/python/api/azure-identity/azure.identity.environmentcredential?view=azure-python>).

### 2. Custom Connections

The Open Model LLM tool uses the CustomConnection. Prompt flow supports two types of connections:

1. **Workspace Connections** - These are connections which are stored as secrets on an Azure Machine Learning workspace. While these can be used, in many places, they are commonly created and maintained in the Studio UI.
2. **Local Connections** - These are connections which are stored locally on your machine. These connections are not available in the Studio UX's, but can be used with the VScode extension.

Instructions on how to create a workspace or local Custom Connection [can be found here](#). ([./../how-to-guides/manage-connections.md#create-a-connection](#))

The required keys to set are:

1. **endpoint\_url**
  - o This value can be found at the previously created Inferencing endpoint.
2. **endpoint\_api\_key**
  - o Ensure to set this as a secret value.
  - o This value can be found at the previously created Inferencing endpoint.
3. **model\_family**
  - o Supported values: LLAMA, DOLLY, GPT2, or FALCON
  - o This value is dependent on the type of deployment you are targeting.

## Running the Tool: Inputs

The Open Model LLM tool has a number of parameters, some of which are required. Please see the below table for details, you can match these to the screen shot above for visual clarity.

Name	Type	Description	Required
api	string	This is the API mode and will depend on the model used and the scenario selected. <i>Supported values: (Completion   Chat)</i>	Yes
endpoint_name	string	Name of an Online Inferencing Endpoint with a supported model deployed on it. Takes priority over connection.	No
temperature	float	The randomness of the generated text. Default is 1.	No
max_new_tokens	integer	The maximum number of tokens to generate in the completion. Default is 500.	No
top_p	float	The probability of using the top choice from the generated tokens. Default is 1.	No

Name	Type	Description	Required
model_kwargs	dictionary	This input is used to provide configuration specific to the model used. For example, the Llama-02 model may use {"temperature":0.4}. <i>Default:</i>	No
deployment_name	string	The name of the deployment to target on the Online Inferencing endpoint. If no value is passed, the Inferencing load balancer traffic settings will be used.	No
prompt	string	The text prompt that the language model will use to generate it's response.	Yes

## Outputs

API	Return Type	Description
Completion	string	The text of one predicted completion
Chat	string	The text of one response in the conversation

## Deploying to an Online Endpoint

When deploying a flow containing the Open Model LLM tool to an online endpoint, there is an additional step to setup permissions. During deployment through the web pages, there is a choice between System-assigned and User-assigned Identity types. Either way, using the Azure Portal (or a similar functionality), add the "Reader" Job function role to the identity on the Azure Machine Learning workspace or Ai Studio project which is hosting the endpoint. The prompt flow deployment may need to be refreshed.

filepath: promptflow/docs/reference/tools-reference/serp-api-tool.md content: # SerpAPI

## Introduction

The SerpAPI API is a Python tool that provides a wrapper to the [SerpAPI Google Search Engine Results API \(https://serpapi.com/search-api\)](https://serpapi.com/search-api) and [SerpAPI Bing Search Engine Results API \(https://serpapi.com/bing-search-api\)](https://serpapi.com/bing-search-api). We could use the tool to retrieve search results from a number of different search engines, including Google and Bing, and you can specify a range of search parameters, such as the search query, location, device type, and more.

## Prerequisite

Sign up at [SERP API homepage \(https://serpapi.com/\)](https://serpapi.com/)

## Connection

Connection is the model used to establish connections with Serp API.

Type	Name	API KEY
Serp	Required	Required

*API Key is on SerpAPI account dashboard*

## Inputs

The **serp api** tool supports following parameters:

Name	Type	Description	Required
query	string	The search query to be executed.	Yes
engine	string	The search engine to use for the search. Default is 'google'.	Yes
num	integer	The number of search results to return. Default is 10.	No
location	string	The geographic location to execute the search from.	No
safe	string	The safe search mode to use for the search. Default is 'off'.	No

## Outputs

The json representation from serpapi query.

Engine	Return Type	Output
google	json	<a href="https://serpapi.com/search-api#api-examples">Sample (https://serpapi.com/search-api#api-examples)</a>
bing	json	<a href="https://serpapi.com/bing-search-api">Sample (https://serpapi.com/bing-search-api)</a>

filepath: promptflow/docs/reference/tools-reference/contentssafety\_text\_tool.md content: # Content Safety (Text)

Azure Content Safety is a content moderation service developed by Microsoft that help users detect harmful content from different modalities and languages. This tool is a wrapper for the Azure Content Safety Text API, which allows you to detect text content and get moderation results. See the [Azure Content Safety \(https://aka.ms/acs-doc\)](https://aka.ms/acs-doc) for more information.

# Requirements

- For AzureML users, the tool is installed in default image, you can use the tool without extra installation.
- For local users, `pip install promptflow-tools`

[!NOTE] Content Safety (Text) tool is now incorporated into the latest `promptflow-tools` package. If you have previously installed the package `promptflow-contentsafety`, please uninstall it to avoid the duplication in your local tool list.

## Prerequisites

- Create an [Azure Content Safety](https://aka.ms/acs-create) (<https://aka.ms/acs-create>) resource.
- Add "Azure Content Safety" connection in prompt flow. Fill "API key" field with "Primary key" from "Keys and Endpoint" section of created resource.

## Inputs

You can use the following parameters as inputs for this tool:

Name	Type	Description	Required
text	string	The text that need to be moderated.	Yes
hate_category	string	The moderation sensitivity for Hate category. You can choose from four options: <i>disable</i> , <i>low_sensitivity</i> , <i>medium_sensitivity</i> , or <i>high_sensitivity</i> . The <i>disable</i> option means no moderation for hate category. The other three options mean different degrees of strictness in filtering out hate content. The default option is <i>medium_sensitivity</i> .	Yes
sexual_category	string	The moderation sensitivity for Sexual category. You can choose from four options: <i>disable</i> , <i>low_sensitivity</i> , <i>medium_sensitivity</i> , or <i>high_sensitivity</i> . The <i>disable</i> option means no moderation for sexual category. The other three options mean different degrees of strictness in filtering out sexual content. The default option is <i>medium_sensitivity</i> .	Yes
self_harm_category	string	The moderation sensitivity for Self-harm category. You can choose from four options: <i>disable</i> , <i>low_sensitivity</i> , <i>medium_sensitivity</i> , or <i>high_sensitivity</i> . The <i>disable</i> option means no moderation for self-harm category. The other three options mean different degrees of strictness in filtering out self-harm content. The default option is <i>medium_sensitivity</i> .	Yes
violence_category	string	The moderation sensitivity for Violence category. You can choose from four options: <i>disable</i> , <i>low_sensitivity</i> , <i>medium_sensitivity</i> , or <i>high_sensitivity</i> . The <i>disable</i> option means no moderation for violence category. The other three options mean different degrees of strictness in filtering out violence content. The default option is <i>medium_sensitivity</i> .	Yes

For more information, please refer to [Azure Content Safety](https://aka.ms/acs-doc) (<https://aka.ms/acs-doc>).

## Outputs

The following is an example JSON format response returned by the tool:

► Output

The `action_by_category` field gives you a binary value for each category: *Accept* or *Reject*. This value shows if the text meets the sensitivity level that you set in the request parameters for that category.

The `suggested_action` field gives you an overall recommendation based on the four categories. If any category has a *Reject* value, the `suggested_action` will be *Reject* as well.

filepath: promptflow/docs/reference/tools-reference/llm-tool.md content: # LLM

## Introduction

Prompt flow LLM tool enables you to leverage widely used large language models like [OpenAI](https://platform.openai.com/) (<https://platform.openai.com/>) or [Azure OpenAI \(AOAI\)](https://learn.microsoft.com/en-us/azure/cognitive-services/openai/overview) (<https://learn.microsoft.com/en-us/azure/cognitive-services/openai/overview>) for natural language processing.

Prompt flow provides a few different LLM APIs:

- **Completion** (<https://platform.openai.com/docs/api-reference/completions>): OpenAI's completion models generate text based on provided prompts.
- **Chat** (<https://platform.openai.com/docs/api-reference/chat>): OpenAI's chat models facilitate interactive conversations with text-based inputs and responses.

[!NOTE] We now remove the *embedding* option from LLM tool api setting. You can use embedding api with [Embedding tool](https://github.com/microsoft/promptflow/blob/main/docs/reference/tools-reference/embedding_tool.md) ([https://github.com/microsoft/promptflow/blob/main/docs/reference/tools-reference/embedding\\_tool.md](https://github.com/microsoft/promptflow/blob/main/docs/reference/tools-reference/embedding_tool.md)).

## Prerequisite

Create OpenAI resources:

- **OpenAI**

Sign up account [OpenAI website \(https://openai.com/\)](https://openai.com/). Login and [Find personal API key \(https://platform.openai.com/account/api-keys\)](https://platform.openai.com/account/api-keys).

- **Azure OpenAI (AOAI)**

Create Azure OpenAI resources with [instruction \(https://learn.microsoft.com/en-us/azure/cognitive-services/openai/how-to/create-resource?pivots=web-portal\)](https://learn.microsoft.com/en-us/azure/cognitive-services/openai/how-to/create-resource?pivots=web-portal).

## Connections

Setup connections to provisioned resources in prompt flow.

Type	Name	API KEY	API Type	API Version
OpenAI	Required	Required	-	-
AzureOpenAI	Required	Required	Required	Required

## Inputs

### Text Completion

Name	Type	Description	Required
prompt	string	text prompt that the language model will complete	Yes
model, deployment_name	string	the language model to use	Yes
max_tokens	integer	the maximum number of tokens to generate in the completion. Default is 16.	No
temperature	float	the randomness of the generated text. Default is 1.	No
stop	list	the stopping sequence for the generated text. Default is null.	No
suffix	string	text appended to the end of the completion	No
top_p	float	the probability of using the top choice from the generated tokens. Default is 1.	No
logprobs	integer	the number of log probabilities to generate. Default is null.	No
echo	boolean	value that indicates whether to echo back the prompt in the response. Default is false.	No
presence_penalty	float	value that controls the model's behavior with regards to repeating phrases. Default is 0.	No
frequency_penalty	float	value that controls the model's behavior with regards to generating rare phrases. Default is 0.	No
best_of	integer	the number of best completions to generate. Default is 1.	No
logit_bias	dictionary	the logit bias for the language model. Default is empty dictionary.	No

### Chat

Name	Type	Description	Required
prompt	string	text prompt that the language model will response	Yes
model, deployment_name	string	the language model to use	Yes
max_tokens	integer	the maximum number of tokens to generate in the response. Default is inf.	No
temperature	float	the randomness of the generated text. Default is 1.	No
stop	list	the stopping sequence for the generated text. Default is null.	No
top_p	float	the probability of using the top choice from the generated tokens. Default is 1.	No
presence_penalty	float	value that controls the model's behavior with regards to repeating phrases. Default is 0.	No
frequency_penalty	float	value that controls the model's behavior with regards to generating rare phrases. Default is 0.	No
logit_bias	dictionary	the logit bias for the language model. Default is empty dictionary.	No
function_call	object	value that controls which function is called by the model. Default is null.	No
functions	list	a list of functions the model may generate JSON inputs for. Default is null.	No
response_format	object	an object specifying the format that the model must output. Default is null.	No

## Outputs



API	Return Type	Description
Completion	string	The text of one predicted completion
Chat	string	The text of one response of conversation

## How to use LLM Tool?

1. Setup and select the connections to OpenAI resources
2. Configure LLM model api and its parameters
3. Prepare the Prompt with [guidance](#) ([./prompt-tool.md#how-to-write-prompt](#)).

filepath: promptflow/docs/reference/tools-reference/python-tool.md content: # Python

## Introduction

Users are empowered by the Python Tool to offer customized code snippets as self-contained executable nodes in PromptFlow. Users can effortlessly create Python tools, edit code, and verify results with ease.

## Inputs

Name	Type	Description	Required
Code	string	Python code snippet	Yes
Inputs -		List of tool function parameters and its assignments -	

## Types

Type	Python example	Description
int	param: int	Integer type
bool	param: bool	Boolean type
string	param: str	String type
double	param: float	Double type
list	param: list or param: List[T]	List type
object	param: dict or param: Dict[K, V]	Object type
<a href="#">Connection</a> ( <a href="#">./../concepts/concept-connections.md</a> )	param: CustomConnection	Connection type, will be handled specially

Parameters with `Connection` type annotation will be treated as connection inputs, which means:

- Promptflow extension will show a selector to select the connection.
- During execution time, promptflow will try to find the connection with the name same from parameter value passed in.

**Note** that `Union[...]` type annotation is supported **ONLY** for connection type, for example, `param: Union[CustomConnection, OpenAIConnection]`.

## Outputs

The return of the python tool function.

## How to write Python Tool?

### Guidelines

1. Python Tool Code should consist of a complete Python code, including any necessary module imports.
2. Python Tool Code must contain a function decorated with `@tool` (tool function), serving as the entry point for execution. The `@tool` decorator should be applied only once within the snippet.

*Below sample defines python tool "my\_python\_tool", decorated with `@tool`*

3. Python tool function parameters must be assigned in 'Inputs' section

*Below sample defines inputs "message" and assign with "world"*

4. Python tool function shall have return

*Below sample returns a concatenated string*

### Code

The snippet below shows the basic structure of a tool function. Promptflow will read the function and extract inputs from function parameters and type annotations.

```

from promptflow import tool
from promptflow.connections import CustomConnection

The inputs section will change based on the arguments of the tool function, after you save the code
Adding type to arguments and return value will help the system show the types properly
Please update the function name/signature per need
@tool
def my_python_tool(message: str, my_conn: CustomConnection) -> str:
 my_conn_dict = dict(my_conn)
 # Do some function call with my_conn_dict...
 return 'hello ' + message

```

## Inputs

Name	Type	Sample Value in Flow Yaml	Value passed to function
message	string	"world"	"world"
my_conn	CustomConnection	"my_conn"	CustomConnection object

Promptflow will try to find the connection named 'my\_conn' during execution time.

## outputs

```
"hello world"
```

## Keyword Arguments Support

Starting from version 1.0.0 of PromptFlow and version 1.4.0 of [Prompt flow for VS Code](https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow) (<https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow>), we have introduced support for keyword arguments (kwargs) in the Python tool.

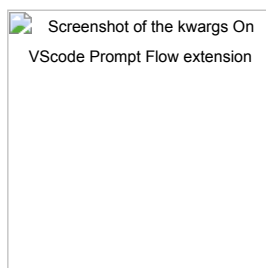
```

from promptflow import tool

@tool
def print_test(normal_input: str, **kwargs):
 for key, value in kwargs.items():
 print(f"Key {key}'s value is {value}")
 return len(kwargs)

```

When you add kwargs in your python tool like above code, you can insert variable number of inputs by the +Add input button.



filepath: promptflow/docs/reference/tools-reference/aoai-gpt4-turbo-vision.md content: # Azure OpenAI GPT-4 Turbo with Vision

## Introduction

Azure OpenAI GPT-4 Turbo with Vision tool enables you to leverage your AzureOpenAI GPT-4 Turbo with Vision model deployment to analyze images and provide textual responses to questions about them.

## Prerequisites

- Create AzureOpenAI resources

Create Azure OpenAI resources with [instruction](https://learn.microsoft.com/en-us/azure/cognitive-services/openai/how-to/create-resource?pivots=web-portal) (<https://learn.microsoft.com/en-us/azure/cognitive-services/openai/how-to/create-resource?pivots=web-portal>).

- Create a GPT-4 Turbo with Vision deployment

Browse to [Azure OpenAI Studio \(https://oai.azure.com/\)](https://oai.azure.com/) and sign in with the credentials associated with your Azure OpenAI resource. During or after the sign-in workflow, select the appropriate directory, Azure subscription, and Azure OpenAI resource.

Under Management select Deployments and Create a GPT-4 Turbo with Vision deployment by selecting model name: `gpt-4` and model version `vision-preview`.

# Connection

Setup connections to provisioned resources in prompt flow.

Type	Name	API KEY	API Type	API Version
AzureOpenAI	Required	Required	Required	Required

# Inputs

Name	Type	Description	Required
connection	AzureOpenAI	the AzureOpenAI connection to be used in the tool	Yes
deployment_name	string	the language model to use	Yes
prompt	string	The text prompt that the language model will use to generate it's response.	Yes
max_tokens	integer	the maximum number of tokens to generate in the response. Default is 512.	No
temperature	float	the randomness of the generated text. Default is 1.	No
stop	list	the stopping sequence for the generated text. Default is null.	No
top_p	float	the probability of using the top choice from the generated tokens. Default is 1.	No
presence_penalty	float	value that controls the model's behavior with regards to repeating phrases. Default is 0.	No
frequency_penalty	float	value that controls the model's behavior with regards to generating rare phrases. Default is 0.	No

# Outputs

Return Type	Description
string	The text of one response of conversation

filepath: `promptflow/docs/reference/tools-reference/faiss_index_lookup_tool.md` content: `# Faiss Index Lookup`

Faiss Index Lookup is a tool tailored for querying within a user-provided Faiss-based vector store. In combination with our Large Language Model (LLM) tool, it empowers users to extract contextually relevant information from a domain knowledge base.

# Requirements

- For AzureML users, the tool is installed in default image, you can use the tool without extra installation.
- For local users, if your index is stored in local path,

```
pip install promptflow-vectorordb
```

if your index is stored in Azure storage,

```
pip install promptflow-vectorordb[azure]
```

# Prerequisites

For AzureML users,

- step 1. Prepare an accessible path on Azure Blob Storage. Here's the guide if a new storage account needs to be created: [Azure Storage Account \(https://learn.microsoft.com/en-us/azure/storage/common/storage-account-create?tabs=azure-portal\)](https://learn.microsoft.com/en-us/azure/storage/common/storage-account-create?tabs=azure-portal).
- step 2. Create related Faiss-based index files on Azure Blob Storage. We support the LangChain format (index.faiss + index.pkl) for the index files, which can be prepared either by employing our promptflow-vectorordb SDK or following the quick guide from [LangChain documentation \(https://python.langchain.com/docs/modules/data\\_connection/vectorstores/integrations/faiss\)](https://python.langchain.com/docs/modules/data_connection/vectorstores/integrations/faiss). Please refer to the instructions of [An example code for creating Faiss index \(https://aka.ms/pf-sample-build-faiss-index\)](https://aka.ms/pf-sample-build-faiss-index) for building index using promptflow-vectorordb SDK.
- step 3. Based on where you put your own index files, the identity used by the promptflow runtime should be granted with certain roles. Please refer to [Steps to assign an Azure role \(https://learn.microsoft.com/en-us/azure/role-based-access-control/role-assignments-steps\)](https://learn.microsoft.com/en-us/azure/role-based-access-control/role-assignments-steps):

Location	Role
workspace datastores or workspace default blob	AzureML Data Scientist

Location	Role
other blobs	Storage Blob Data Reader

For local users,

- Create Faiss-based index files in local path by only doing step 2 above.

## Inputs

The tool accepts the following inputs:

Name	Type	Description
		URL or path for the vector store.
		local path (for local users): <local_path_to_the_index_folder>
path	string	Azure blob URL format (with [azure] extra installed): https://<account_name>.blob.core.windows.net/<container_name>/<path_and_folder_name>.
		AML datastore URL format (with [azure] extra installed): azureml://subscriptions/<your_subscription>/resourcegroups/<your_resource_group>/workspaces/<your_workspace>/data
		public http/https URL (for public demonstration): http(s)://<path_and_folder_name>
vector	list[float]	The target vector to be queried, which can be generated by the LLM tool.
top_k	integer	The count of top-scored entities to return. Default value is 3.

## Outputs

The following is an example for JSON format response returned by the tool, which includes the top-k scored entities. The entity follows a generic schema of vector search result provided by our promptflow-vectoradb SDK. For the Faiss Index Search, the following fields are populated:

Field Name	Type	Description
text	string	Text of the entity
score	float	Distance between the entity and the query vector
metadata	dict	Customized key-value pairs provided by user when create the index

► Output

- **Qdrant:**

For Qdrant, the following fields are populated: | Field Name | Type | Description | | --- | --- | ----- | | original\_entity | dict | the original response json from search REST API| | metadata | dict | payload from the original entity| | score | float | score from the original entity, which evaluates the similarity between the entity and the query vector| | text | string | text of the payload| | vector | list | vector of the entity|

► Output

- **Weaviate:**

For Weaviate, the following fields are populated: | Field Name | Type | Description | | --- | --- | ----- | | original\_entity | dict | the original response json from search REST API| | score | float | certainty from the original entity, which evaluates the similarity between the entity and the query vector| | text | string | text in the original entity| | vector | list | vector of the entity|

► Output

filepath: promptflow/docs/reference/python-library-reference/promptflow.md content: # PLACEHOLDER

filepath: promptflow/docs/tutorials/index.md content: # Tutorials

This section contains a collection of flow samples and step-by-step tutorials.

Area	Sample	Description
SDK	<a href="https://github.com/microsoft/promptflow/blob/main/examples/tutorials/get-started/quickstart.ipynb">Getting started with prompt flow</a> ( <a href="https://github.com/microsoft/promptflow/blob/main/examples/tutorials/get-started/quickstart.ipynb">https://github.com/microsoft/promptflow/blob/main/examples/tutorials/get-started/quickstart.ipynb</a> )	A step by step guidance to invoke your first flow run.
CLI	<a href="https://github.com/microsoft/promptflow/blob/main/examples/tutorials/e2e-development/chat-with-pdf.md">Chat with PDF</a> ( <a href="https://github.com/microsoft/promptflow/blob/main/examples/tutorials/e2e-development/chat-with-pdf.md">https://github.com/microsoft/promptflow/blob/main/examples/tutorials/e2e-development/chat-with-pdf.md</a> )	An end-to-end tutorial on how to build a high quality chat application with prompt flow, including flow development and evaluation with metrics.

Area	Sample	Description
SDK	<a href="https://github.com/microsoft/promptflow/blob/main/examples/flows/chat/chat-with-pdf/chat-with-pdf.ipynb">Chat with PDF - test, evaluation and experimentation</a> ( <a href="https://github.com/microsoft/promptflow/blob/main/examples/flows/chat/chat-with-pdf/chat-with-pdf.ipynb">https://github.com/microsoft/promptflow/blob/main/examples/flows/chat/chat-with-pdf/chat-with-pdf.ipynb</a> )	We will walk you through how to use prompt flow Python SDK to test, evaluate and experiment with the "Chat with PDF" flow.
SDK	<a href="https://github.com/microsoft/promptflow/blob/main/examples/connections/connection.ipynb">Connection management</a> ( <a href="https://github.com/microsoft/promptflow/blob/main/examples/connections/connection.ipynb">https://github.com/microsoft/promptflow/blob/main/examples/connections/connection.ipynb</a> )	Manage various types of connections using sdk
CLI	<a href="https://github.com/microsoft/promptflow/blob/main/examples/connections/README.md">Working with connection</a> ( <a href="https://github.com/microsoft/promptflow/blob/main/examples/connections/README.md">https://github.com/microsoft/promptflow/blob/main/examples/connections/README.md</a> )	Manage various types of connections using cli
SDK	<a href="https://github.com/microsoft/promptflow/blob/main/examples/tutorials/get-started/quickstart-azure.ipynb">Run prompt flow in Azure AI</a> ( <a href="https://github.com/microsoft/promptflow/blob/main/examples/tutorials/get-started/quickstart-azure.ipynb">https://github.com/microsoft/promptflow/blob/main/examples/tutorials/get-started/quickstart-azure.ipynb</a> )	A quick start tutorial to run a flow in Azure AI and evaluate it.
SDK	<a href="https://github.com/microsoft/promptflow/blob/main/examples/tutorials/run-management/cloud-run-management.ipynb">Flow run management in Azure AI</a> ( <a href="https://github.com/microsoft/promptflow/blob/main/examples/tutorials/run-management/cloud-run-management.ipynb">https://github.com/microsoft/promptflow/blob/main/examples/tutorials/run-management/cloud-run-management.ipynb</a> )	Flow run management in azure AI

## Samples

Area	Sample	Description
Standard Flow	<a href="https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/basic">basic</a> ( <a href="https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/basic">https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/basic</a> )	a basic flow with prompt and python tool.
Standard Flow	<a href="https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/basic-with-connection">basic-with-connection</a> ( <a href="https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/basic-with-connection">https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/basic-with-connection</a> )	a basic flow using custom connection with prompt and python tool
Standard Flow	<a href="https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/basic-with-builtin-llm">basic-with-builtin-llm</a> ( <a href="https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/basic-with-builtin-llm">https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/basic-with-builtin-llm</a> )	a basic flow using builtin llm tool
Standard Flow	<a href="https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/customer-intent-extraction">customer-intent-extraction</a> ( <a href="https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/customer-intent-extraction">https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/customer-intent-extraction</a> )	a flow created from existing langchain python code
Standard Flow	<a href="https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification">web-classification</a> ( <a href="https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification">https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/web-classification</a> )	a flow demonstrating multi-class classification with LLM. Given an url, it will classify the url into one web category with just a few shots, simple summarization and classification prompts.
Standard Flow	<a href="https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/autonomous-agent">autonomous-agent</a> ( <a href="https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/autonomous-agent">https://github.com/microsoft/promptflow/tree/main/examples/flows/standard/autonomous-agent</a> )	a flow showcasing how to construct a AutoGPT flow to autonomously figures out how to apply the given functions to solve the goal, which is film trivia that provides accurate and up-to-date information about movies, directors, actors, and more.
Chat Flow	<a href="https://github.com/microsoft/promptflow/tree/main/examples/flows/chat/chat-with-wikipedia">chat-with-wikipedia</a> ( <a href="https://github.com/microsoft/promptflow/tree/main/examples/flows/chat/chat-with-wikipedia">https://github.com/microsoft/promptflow/tree/main/examples/flows/chat/chat-with-wikipedia</a> )	a flow demonstrating Q&A with GPT3.5 using information from Wikipedia to make the answer more grounded.
Chat Flow	<a href="https://github.com/microsoft/promptflow/tree/main/examples/flows/chat/chat-with-pdf">chat-with-pdf</a> ( <a href="https://github.com/microsoft/promptflow/tree/main/examples/flows/chat/chat-with-pdf">https://github.com/microsoft/promptflow/tree/main/examples/flows/chat/chat-with-pdf</a> )	a flow that allow you to ask questions about the content of a PDF file and get answers.
Evaluation Flow	<a href="https://github.com/microsoft/promptflow/tree/main/examples/flows/evaluation/eval-classification-accuracy">eval-classification-accuracy</a> ( <a href="https://github.com/microsoft/promptflow/tree/main/examples/flows/evaluation/eval-classification-accuracy">https://github.com/microsoft/promptflow/tree/main/examples/flows/evaluation/eval-classification-accuracy</a> )	a flow illustrating how to evaluate the performance of a classification system.

Learn more: [Try out more promptflow examples](https://github.com/microsoft/promptflow/tree/main/examples) (<https://github.com/microsoft/promptflow/tree/main/examples>)

filepath: promptflow/docs/concepts/concept-variants.md content: With prompt flow, you can use variants to tune your prompt. In this article, you'll learn the prompt flow variants concept.

## Variants

A variant refers to a specific version of a tool node that has distinct settings. Currently, variants are supported only in the LLM tool. For example, in the LLM tool, a new variant can represent either a different prompt content or different connection settings.

Suppose you want to generate a summary of a news article. You can set different variants of prompts and settings like this:

Variants	Prompt	Connection settings
Variant 0	Summary: {{input sentences}}	Temperature = 1
Variant 1	Summary: {{input sentences}}	Temperature = 0.7
Variant 2	What is the main point of this article? {{input sentences}}	Temperature = 1
Variant 3	What is the main point of this article? {{input sentences}}	Temperature = 0.7

By utilizing different variants of prompts and settings, you can explore how the model responds to various inputs and outputs, enabling you to discover the most suitable combination for your requirements.

## Benefits of using variants

- **Enhance the quality of your LLM generation:** By creating multiple variants of the same LLM node with diverse prompts and configurations, you can identify the optimal combination that produces high-quality content aligned with your needs.
- **Save time and effort:** Even slight modifications to a prompt can yield significantly different results. It's crucial to track and compare the performance of each prompt version. With variants, you can easily manage the historical versions of your LLM nodes, facilitating updates based on any variant without the risk of forgetting previous iterations. This saves you time and effort in managing prompt tuning history.
- **Boost productivity:** Variants streamline the optimization process for LLM nodes, making it simpler to create and manage multiple variations. You can achieve improved results in less time, thereby increasing your overall productivity.
- **Facilitate easy comparison:** You can effortlessly compare the results obtained from different variants side by side, enabling you to make data-driven decisions regarding the variant that generates the best outcomes.

## Next steps

- [Tune prompts with variants \(../how-to-guides/tune-prompts-with-variants.md\)](#)

filepath: promptflow/docs/concepts/index.md content: # Concepts

In this section, you will learn the basic concepts of prompt flow.

```
:maxdepth: 1

concept-flows
concept-tools
concept-connections
concept-variants
design-principles
```

filepath: promptflow/docs/concepts/concept-flows.md content: While how LLMs work may be elusive to many developers, how LLM apps work is not - they essentially involve a series of calls to external services such as LLMs/databases/search engines, or intermediate data processing, all glued together. Thus LLM apps are merely Directed Acyclic Graphs (DAGs) of function calls. These DAGs are flows in prompt flow.

## Flows

A flow in prompt flow is a DAG of functions (we call them [tools \(./concept-tools.md\)](#)). These functions/tools connected via input/output dependencies and executed based on the topology by prompt flow executor.

A flow is represented as a YAML file and can be visualized with our [Prompt flow for VS Code extension \(https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow\)](https://marketplace.visualstudio.com/items?itemName=prompt-flow.prompt-flow). Here is an example:



## Flow types

Prompt flow has three flow types:

- **Standard flow** and **Chat flow:** these two are for you to develop your LLM application. The primary difference between the two lies in the additional support provided by the "Chat Flow" for chat applications. For instance, you can define chat\_history, chat\_input, and chat\_output for your flow. The prompt flow, in turn, will offer a chat-like experience (including conversation history) during the development of the flow. Moreover, it also provides a sample chat application for deployment purposes.
- **Evaluation flow** is for you to test/evaluate the quality of your LLM application (standard/chat flow). It usually run on the outputs of standard/chat flow, and compute some metrics that can be used to determine whether the standard/chat flow performs well. E.g. is the answer accurate? is the answer fact-based?

# When to use standard flow vs. chat flow?

As a general guideline, if you are building a chatbot that needs to maintain conversation history, try chat flow. In most other cases, standard flow should serve your needs.

Our examples should also give you an idea when to use what:

- [examples/flows/standard](https://github.com/microsoft/promptflow/tree/main/examples/flows/standard) (<https://github.com/microsoft/promptflow/tree/main/examples/flows/standard>)
- [examples/flows/chat](https://github.com/microsoft/promptflow/tree/main/examples/flows/chat) (<https://github.com/microsoft/promptflow/tree/main/examples/flows/chat>)

## Next steps

- [Quick start](#) ([./how-to-guides/quick-start.md](#))
- [Initialize and test a flow](#) ([./how-to-guides/init-and-test-a-flow.md](#))
- [Run and evaluate a flow](#) ([./how-to-guides/run-and-evaluate-a-flow/index.md](#))
- [Tune prompts using variants](#) ([./how-to-guides/tune-prompts-with-variants.md](#))

filepath: promptflow/docs/concepts/concept-tools.md content: Tools are the fundamental building blocks of a [flow](#) ([./concept-flows.md](#)).

Each tool is an executable unit, basically a function to performs various tasks including but not limited to:

- Accessing LLMs for various purposes
- Querying databases
- Getting information from search engines
- Pre/post processing of data

## Tools

Prompt flow provides 3 basic tools:

- [LLM](#) ([./reference/tools-reference/llm-tool.md](#)): The LLM tool allows you to write custom prompts and leverage large language models to achieve specific goals, such as summarizing articles, generating customer support responses, and more.
- [Python](#) ([./reference/tools-reference/python-tool.md](#)): The Python tool enables you to write custom Python functions to perform various tasks, such as fetching web pages, processing intermediate data, calling third-party APIs, and more.
- [Prompt](#) ([./reference/tools-reference/prompt-tool.md](#)): The Prompt tool allows you to prepare a prompt as a string for more complex use cases or for use in conjunction with other prompt tools or python tools.

## More tools

Our partners also contributes other useful tools for advanced scenarios, here are some links:

- [Vector DB Lookup](#) ([./reference/tools-reference/vector\\_db\\_lookup\\_tool.md](#)): vector search tool that allows users to search top k similar vectors from vector database.
- [Faiss Index Lookup](#) ([./reference/tools-reference/faiss\\_index\\_lookup\\_tool.md](#)): querying within a user-provided Faiss-based vector store.

## Custom tools

You can create your own tools that can be shared with your team or anyone in the world. Learn more on [Create and Use Tool Package](#) ([./how-to-guides/develop-a-tool/create-and-use-tool-package.md](#)).

## Next steps

For more information on the available tools and their usage, visit the our [reference doc](#) ([./reference/index.md](#)).

filepath: promptflow/docs/concepts/design-principles.md content: # Design principles

When we started this project, [LangChain](https://www.langchain.com/) (<https://www.langchain.com/>) already became popular esp. after the ChatGPT launch. One of the questions we've been asked is what's the difference between prompt flow and LangChain. This article is to elucidate the reasons for building prompt flow and the deliberate design choices we have made. To put it succinctly, prompt flow is a suite of development tools for you to build LLM apps with a strong emphasis of quality through experimentations, not a framework - which LangChain is.

While LLM apps are mostly in exploration stage, Microsoft started in this area a bit earlier and we've had the opportunity to observe how developers are integrating LLMs into existing systems or build new applications. These invaluable insights have shaped the fundamental design principles of prompt flow.

## 1. Expose the prompts vs. hiding them

The core essence of LLM applications lies in the prompts themselves, at least for today. When developing a reasonably complex LLM application, the majority of development work should be “tuning” the prompts (note the intentional use of the term “tuning,” which we will delve into further later on). Any framework or tool trying to help in this space should focus on making prompt tuning easier and more straightforward. On the other hand, prompts are very volatile, it’s unlikely to write a single prompt that can work across different models or even different version of same models. Building a successful LLM-based application, you have to understand every prompt introduced, so that you can tune it when necessary. LLM is simply not powerful or deterministic enough that you can use a prompt written by others like you use libraries in traditional programming languages.

In this context, any design that tries to provide a smart function or agent by encapsulating a few prompts in a library is unlikely to yield favorable results in real-world scenarios. And hiding prompts inside a library’s code base only makes it’s hard for people to improve or tailor the prompts to suit their specific needs.

Prompt flow, being positioned as a tool, refrains from wrapping any prompts within its core codebase. The only place you will see prompts are our sample flows, which are, of course, available for adoption and utilization. Every prompt should be authored and controlled by the developers themselves, rather than relying on us.

## 2. A new way of work

LLMs possess remarkable capabilities that enable developers to enhance their applications without delving deep into the intricacies of machine learning. In the meantime, LLMs make these apps more stochastic, which pose new challenges to application development. Merely asserting “no exception” or “result == x” in gated tests is no longer sufficient. Adopting a new methodology and employing new tools becomes imperative to ensure the quality of LLM applications — an entirely novel way of working is required.

At the center of this paradigm shift is evaluation, a term frequently used in machine learning space, refers to the process of assessing the performance and quality of a trained model.

It involves measuring how well the model performs on a given task or dataset, which plays a pivotal role in understanding the model’s strengths, weaknesses, and overall effectiveness. Evaluation metrics and techniques vary depending on the specific task and problem domain. Some common metrics include accuracy, precision and recall, you probably already familiar with. Now the LLM apps share similarities with machine learning models, they requires an evaluation-centric approach integrated into the development workflow, with a robust set of metrics and evaluation forming the foundation for ensuring the quality of LLM applications.

Prompt flow offers a range of tools to streamline the new way of work:

- Develop your evaluation program as Evaluation flow to calculate metrics for your app/flow, learn from our sample evaluation flows.
- Iterate on your application flow and run evaluation flows via the SDK/CLI, allowing you to compare metrics and choose the optimal candidate for release. These iterations include trying different prompts, different LLM parameters like temperature etc. - this is referred as “tuning” process earlier, or sometime referred as experimentation.
- Integrate the evaluation into your CI/CD pipeline, aligning the assertions in your gated tests with the selected metrics.

Prompt flow introduces two conceptual components to facilitate this workflow:

- Evaluation flow: a flow type that indicates this flow is not for deploy or integrate into your app, it’s for evaluating an app/flow performance.
- Run: every time you run your flow with data, or run an evaluation on the output of a flow, a Run object is created to manage the history and allow for comparison and additional analysis.

While new concepts introduce additional cognitive load, we firmly believe they hold greater importance compared to abstracting different LLM APIs or vector database APIs.

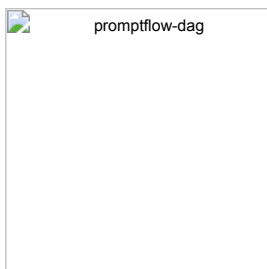
## 3. Optimize for “visibility”

There are quite some interesting application patterns emerging because of LLMs, like Retrieval Augmented Generation (RAG), ReAct and more. Though how LLMs work may remain enigmatic to many developers, how LLM apps work is not - they essentially involve a series of calls to external services such as LLMs, databases, and search engines, all glued together. Architecturally there isn’t much new, patterns like RAG and ReAct are both straightforward to implement once a developer understands what they are - plain Python programs with API calls to external services can totally serve the purpose effectively.

By observing many internal use cases, we learned that deeper insight into the detail of the execution is critical. Establishing a systematic method for tracking interactions with external systems is one of design priority. Consequently, We adopted an unconventional approach - prompt flow has a YAML file describing how function calls (we call them Tools ([./concepts/concept-tools.md](#))) are executed and connected into a Directed Acyclic Graph (DAG).

This approach offers several key benefits, primarily centered around **enhanced visibility**:

1. During development, your flow can be visualized in an intelligible manner, enabling clear identification of any faulty components. As a byproduct, you obtain an architecturally descriptive diagram that can be shared with others.
2. Each node in the flow has it’s internal detail visualized in a consistent way.
3. Single nodes can be individually run or debugged without the need to rerun previous nodes.



The emphasis on visibility in prompt flow’s design helps developers to gain a comprehensive understanding of the intricate details of their applications. This, in turn, empowers developers to engage in effective troubleshooting and optimization.



Despite there're some control flow features like "activate-when" to serve the needs of branches/switch-case, we do not intend to make Flow itself Turing-complete. If you want to develop an agent which is fully dynamic and guided by LLM, leveraging [Semantic Kernel \(https://github.com/microsoft/semantic-kernel\)](https://github.com/microsoft/semantic-kernel) together with prompt flow would be a favorable option.

filepath: promptflow/docs/concepts/concept-connections.md content: In prompt flow, you can utilize connections to securely manage credentials or secrets for external services.

# Connections

Connections are for storing information about how to access external services like LLMs: endpoint, api keys etc.

- In your local development environment, the connections are persisted in your local machine with keys encrypted.
- In Azure AI, connections can be configured to be shared across the entire workspace. Secrets associated with connections are securely persisted in the corresponding Azure Key Vault, adhering to robust security and compliance standards.

Prompt flow provides a variety of pre-built connections, including Azure Open AI, Open AI, etc. These pre-built connections enable seamless integration with these resources within the built-in tools. Additionally, you have the flexibility to create custom connection types using key-value pairs, empowering them to tailor the connections to their specific requirements, particularly in Python tools.

Connection type	Built-in tools
<a href="https://azure.microsoft.com/en-us/products/cognitive-services/openai-service">Azure Open AI (https://azure.microsoft.com/en-us/products/cognitive-services/openai-service)</a>	LLM or Python
<a href="https://openai.com/">Open AI (https://openai.com/)</a>	LLM or Python
<a href="https://azure.microsoft.com/en-us/products/search">Cognitive Search (https://azure.microsoft.com/en-us/products/search)</a>	Vector DB Lookup or Python
<a href="https://serpapi.com/">Serp (https://serpapi.com/)</a>	Serp API or Python
Custom	Python

By leveraging connections in prompt flow, you can easily establish and manage connections to external APIs and data sources, facilitating efficient data exchange and interaction within their AI applications.

## Next steps

- [Create connections \(./how-to-guides/manage-connections.md\)](#)