PyGPT - pygpt.net

Release: 2.6.65 (2025-09-28), Last update: 2025-09-28 08:00
Project Website: https://pygpt.net
GitHub: https://github.com/szczyglis-dev/py-gpt
Snap Store: https://snapcraft.io/pygpt
PyPI: https://pypi.org/project/pygpt-net
Microsoft Store: https://apps.microsoft.com/detail/XP99R4MX3X65VQ

Contents

PyGPT - pygpt.net

Release: 2.6.65 (2025-09-28), Last update: 2025-09-28 08:00
Project Website: https://pygpt.net
GitHub: https://github.com/szczyglis-dev/py-gpt
Snap Store: https://snapcraft.io/pygpt
PyPI: https://pypi.org/project/pygpt-net
Microsoft Store:
https://apps.microsoft.com/detail/XP99R4MX3X65VQ

Contents

Introduction

Overview

PyGPT is **all-in-one** Desktop AI Assistant that provides direct interaction with OpenAI language models, including `GPT-5`, `GPT-4`, `o1`, `o3` and more, through the `OpenAI API`. By utilizing other SDKs and `LlamaIndex`, the application also supports alternative LLMs, like those available on `HuggingFace`, locally available models via `Ollama` (like `gpt-oss`, `Llama 3`, `Mistral`, `DeepSeek V3/R1` or `Bielik`), and other models like `Google Gemini`, `Anthropic Claude`, `Perplexity / Sonar`, and `xAI Grok`.

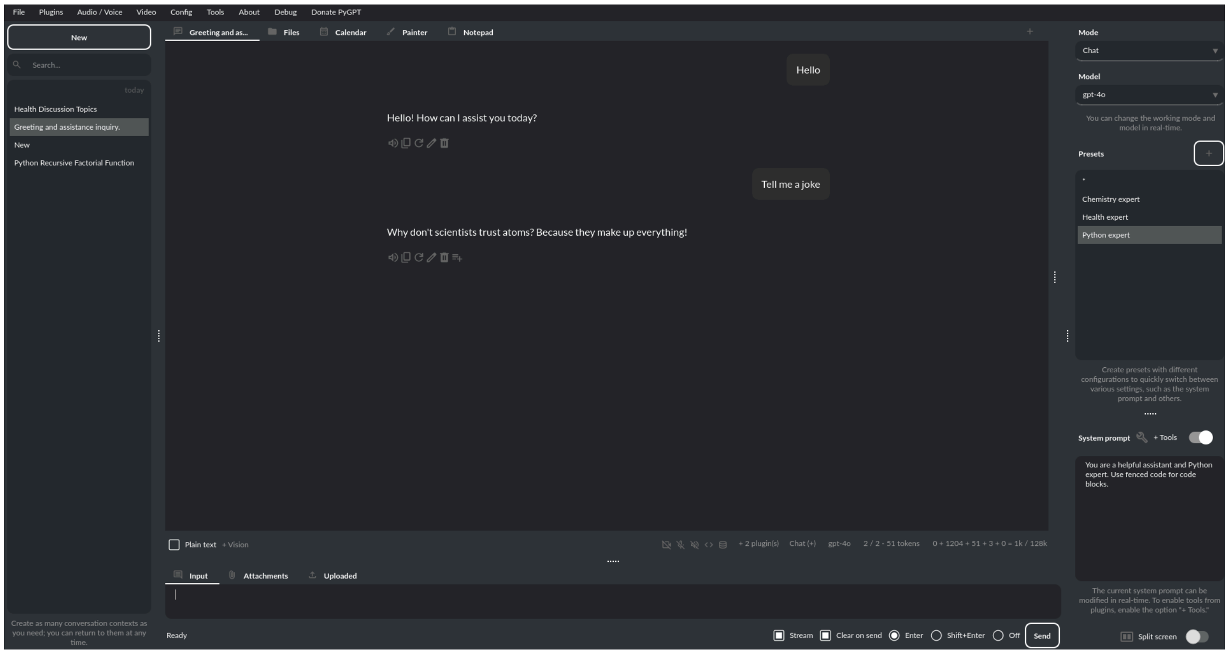
This assistant offers multiple modes of operation such as chat, assistants, agents, completions, and image-related tasks like image generation and image analysis. **PyGPT** has filesystem capabilities for file I/O, can generate and run Python code, execute system commands, execute custom commands and manage file transfers. It also allows models to perform web searches with the `DuckDuckGo`, `Google` and `Microsoft Bing`.

For audio interactions, **PyGPT** includes speech synthesis using the `Microsoft Azure`, `Google`, `Eleven Labs` and `OpenAI` Text-To-Speech services. Additionally, it features speech recognition capabilities provided by `OpenAI Whisper`, `Google` and `Bing` enabling the application to understand spoken commands and transcribe audio inputs into text. It features context memory with save and load functionality, enabling users to resume interactions from predefined points in the conversation. Prompt creation and management are streamlined through an intuitive preset system.

PyGPT's functionality extends through plugin support, allowing for custom enhancements (with multiple plugins included). Its multi-modal capabilities make it an adaptable tool for a range of AI-assisted operations, such as text-based interactions, system automation, daily assisting, vision applications, natural language processing, code generation and image creation.

Multiple operation modes are included, such as chat, text completion, assistant, agents, vision, Chat with Files (via `LlamaIndex`), commands execution, external API calls and image generation, making **PyGPT** a multi-tool for many AI-driven tasks.

Dark theme



today
Health Discussion Topics
Greeting and assistance inquiry.
New
Python Recursive Factorial Function

Create as many conversation contexts as you need; you can return to them at any time.

Hello

Hello! How can I assist you today?

Tell me a joke

Why don't scientists trust atoms? Because they make up everything!

Plain text + Vision

+ 2 plugin(s) Chat (+) gpt-4o 2 / 2 - 51 tokens 0 + 1204 + 51 + 3 + 0 = 1k / 128k

Ready

Stream Clear on send Enter Shift+Enter Off

Mode: Chat

Model: gpt-4o

You can change the working mode and model in real-time.

Presets
*
Chemistry expert
Health expert
Python expert

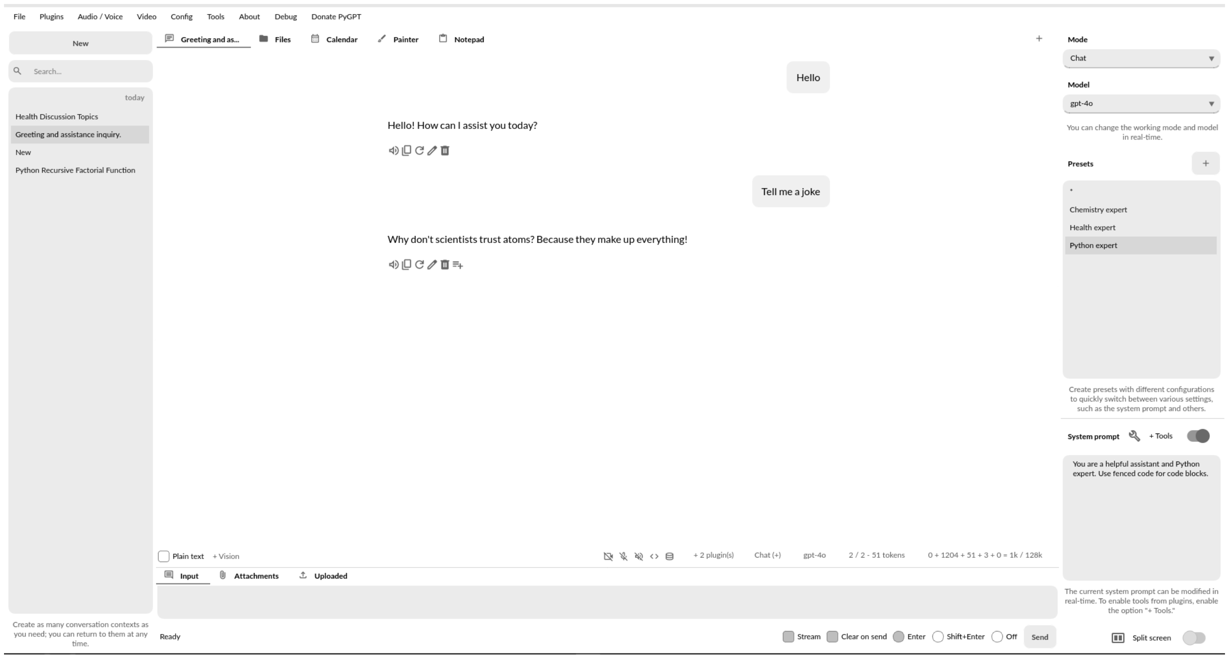
Create presets with different configurations to quickly switch between various settings, such as the system prompt and others.

System prompt + Tools

You are a helpful assistant and Python expert. Use fenced code for code blocks.

The current system prompt can be modified in real-time. To enable tools from plugins, enable the option "+ Tools."

Light theme



Health Discussion Topics
Greeting and assistance inquiry.
New
Python Recursive Factorial Function

Hello

Hello! How can I assist you today?

Tell me a joke

Why don't scientists trust atoms? Because they make up everything!

Mode: Chat

Model: gpt-4o

You can change the working mode and model in real-time.

Presets: *, Chemistry expert, Health expert, Python expert

Create presets with different configurations to quickly switch between various settings, such as the system prompt and others.

System prompt: You are a helpful assistant and Python expert. Use fenced code for code blocks.

The current system prompt can be modified in real-time. To enable tools from plugins, enable the option "+ Tools."

Create as many conversation contexts as you need; you can return to them at any time.

Features

- Desktop AI Assistant for `Linux`, `Windows` and `Mac`, written in Python.
- Works similarly to `ChatGPT`, but locally (on a desktop computer).
- 11 modes of operation: Chat, Chat with Files, Realtime + audio, Research (Perplexity), Completion, Image and video generation, Assistants, Experts, Computer use, Agents and Autonomous Mode.
- Supports multiple models like `OpenAI GPT-5`, `GPT-4`, `o1`, `o3`, `o4`, `Google Gemini`, `Anthropic Claude`, `xAI Grok`, `DeepSeek V3/R1`, `Perplexity / Sonar`, and any model accessible through `LlamaIndex` and `Ollama` such as `DeepSeek`, `gpt-oss`, `Llama 3`, `Mistral`, `Bielik`, etc.
- Chat with your own Files: integrated `LlamaIndex` support: chat with data such as: `txt`, `pdf`, `csv`, `html`, `md`, `docx`, `json`, `epub`, `xlsx`, `xml`, webpages, `Google`, `GitHub`, video/audio, images and other data types, or use conversation history as additional context provided to the model.

- Built-in vector databases support and automated files and data embedding.
- Included support features for individuals with disabilities: customizable keyboard shortcuts, voice control, and translation of on-screen actions into audio via speech synthesis.
- Handles and stores the full context of conversations (short and long-term memory).
- Internet access via `DuckDuckGo`, `Google` and `Microsoft Bing`.
- Speech synthesis via `Microsoft Azure`, `Google`, `Eleven Labs` and `OpenAI` Text-To-Speech services.
- Speech recognition via `OpenAI Whisper`, `Google` and `Microsoft Speech Recognition`.
- MCP support.
- Real-time video camera capture in Vision mode.
- Image analysis via `GPT-5` and `GPT-4o`.
- Integrated calendar, day notes and search in contexts by selected date.
- Tools and commands execution (via plugins: access to the local filesystem, Python Code Interpreter, system commands execution, and more).
- Custom commands creation and execution.
- Crontab / Task scheduler included.
- Built-in real-time Python Code Interpreter.
- Manages files and attachments with options to upload, download, and organize.
- Context history with the capability to revert to previous contexts (long-term memory).
- Allows you to easily manage prompts with handy editable presets.
- Provides an intuitive operation and interface.
- Includes a notepad.
- Includes simple painter / drawing tool.
- Includes an node-based Agents Builder.
- Supports multiple languages.
- Requires no previous knowledge of using AI models.

- Simplifies image generation using image models like *DALL-E* and *Imagen*.
- Fully configurable.
- Themes support.
- Real-time code syntax highlighting.
- Plugins support with built-in plugins like `Files I/O`, `Code Interpreter`, `Web Search`, `Google`, `Facebook`, `X/Twitter`, `Slack`, `Telegram`, `GitHub`, `MCP`, and many more.
- Built-in token usage calculation.
- Possesses the potential to support future OpenAI models.
- **Open source**; source code is available on `GitHub`.
- Utilizes the user's own API key.
- and many more.

The application is free, open-source, and runs on PCs with `Linux`, `Windows 10`, `Windows 11` and `Mac`. Full Python source code is available on `GitHub`.

PyGPT uses the user's API key - to use the GPT models, you must have a registered OpenAI account and your own API key. Local models do not require any API keys.

Note

This application is not officially associated with OpenAI. The author shall not be held liable for any damages resulting from the use of this application. It is provided "as is," without any form of warranty. Users are reminded to be mindful of token usage - always verify the number of tokens utilized by the model on the API website and engage with the application responsibly. Activating plugins, such as Web Search, may consume additional tokens that are not displayed in the main window. **Always monitor your actual token usage on the OpenAI, Google, Anthropic, etc. websites.**

How to Install

Binaries

You can download compiled binary versions for `Linux` and `Windows` (10/11).

PyGPT binaries require a PC with Windows 10, 11, or Linux. Simply download the installer or the archive with the appropriate version from the download page at https://pygpt.net, extract it, or install it, and then run the application. A binary version for Mac is not available, so you must run PyGPT from PyPi or from the source code on Mac. Currently, only 64-bit binaries are available.

Windows 10 and 11

The application is available for 64-bit Windows 10, 11 in the form of an MSI installer. The installer will automatically install all required dependencies and create a shortcut on the desktop. Just download the installer from the download page and run it

Linux

The application is available for 64-bit Linux in the form of an archive with all required dependencies. Just download the archive from the download page and extract it. Then run the application by running the `pygpt` binary file in the root directory.

Linux version requires `GLIBC` >= `2.35`.

Snap Store

You can install **PyGPT** directly from Snap Store:

```
$ sudo snap install pygpt
```

To manage future updates just use:

```
$ sudo snap refresh pygpt
```

Using camera: to use camera in Snap version you must connect the camera with interface:

```
$ snap connect pygpt:camera
```

Using microphone: to use microphone in Snap version you must connect the microphone with:

```
$ sudo snap connect pygpt:audio-record :audio-record
$ sudo snap connect pygpt:alsa
```

Using audio output: to use audio output in Snap version you must connect the audio with:

```
$ sudo snap connect pygpt:audio-playback
$ sudo snap connect pygpt:alsa
```

Connecting IPython in Docker in Snap version:

To use IPython in the Snap version, you must connect PyGPT to the Docker daemon:

```
$ sudo snap connect pygpt:docker-executables docker:docker-
executables

$ sudo snap connect pygpt:docker docker:docker-daemon
```

Snap Store: https://snapcraft.io/pygpt

Python version

The second way to run is to download the source code from GitHub and run the application using the Python interpreter (`>=3.10`, `<3.14`). You can also install application from PyPi (using `pip install`) and we recommend this type of installation.

PyPi (pip)

1. Create a new virtual environment:

```
$ python3 -m venv venv
$ source venv/bin/activate
```

2. Install from PyPi:

```
$ pip install pygpt-net
```

3. Once installed run the command to start the application:

```
$ pygpt
```

Running from source code

Install with pip

1. Clone git repository or download .zip file:

```
$ git clone https://github.com/szczyglis-dev/py-gpt.git
$ cd py-gpt
```

2. Create a new virtual environment:

```
$ python3 -m venv venv
$ source venv/bin/activate
```

3. Install requirements:

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

4. Run the application:

```
$ python3 run.py
```

Install with Poetry

1. Clone git repository or download .zip file:

```
$ git clone https://github.com/szczyglis-dev/py-gpt.git
$ cd py-gpt
```

2. Install Poetry (if not installed):

```
$ pip install poetry
```

3. Create a new virtual environment that uses Python 3.10:

```
$ poetry env use python3.10
$ poetry shell
```

or (Poetry >= 2.0):

```
$ poetry env use python3.10
$ poetry env activate
```

4. Install requirements:

```
$ poetry install
```

5. Run the application:

```
$ poetry run python3 run.py
```

Tip

You can use `PyInstaller` to create a compiled version of the application for your system (required version >= `6.4.0`).

Troubleshooting

If you have a problems with `xcb` plugin with newer versions of PySide on Linux, e.g. like this:

```
qt.qpa.plugin: Could not load the Qt platform plugin "xcb" in
"" even though it was found.
This application failed to start because no Qt platform plugin
could be initialized. Reinstalling the application may fix this
problem.
```

…then install libxcb on linux:

```
$ sudo apt install libxcb-cursor0
```

If you have a problems with audio on Linux, then try to install `portaudio19-dev` and/or `libasound2`:

```
$ sudo apt install portaudio19-dev
```

```
$ sudo apt install libasound2
$ sudo apt install libasound2-data
$ sudo apt install libasound2-plugins
```

Access to camera in Snap version:

To use camera in Vision mode in Snap version you must connect the camera with:

```
$ sudo snap connect pygpt:camera
```

Access to microphone in Snap version:

To use microphone in Snap version you must connect the microphone with:

```
$ sudo snap connect pygpt:audio-record :audio-record
```

Snap and AppArmor permission denied

Snap installs AppArmor profiles for each application by default. The profile for PyGPT is created at:

```
/var/lib/snapd/apparmor/profiles/snap.pygpt.pygpt
```

The application should work with the default profile; however, if you encounter errors like:

```
PermissionError: [Errno 13] Permission denied:
'/etc/httpd/conf/mime.types'
```

add the appropriate access rules to the profile file, for example:

```
# /var/lib/snapd/apparmor/profiles/snap.pygpt.pygpt

...

/etc/httpd/conf/mime.types r
```

and reload the profiles.

Alternatively, you can try removing snap and reinstalling it:

```
$ sudo snap remove --purge pygpt

$ sudo snap install pygpt
```

Problems with GLIBC on Linux

If you encounter error:

```
Error loading Python lib libpython3.10.so.1.0: dlopen:
/lib/x86_64-linux-gnu/libm.so.6: version GLIBC_2.35 not found
```

```
(required by libpython3.10.so.1.0)
```

when trying to run the compiled version for Linux, try updating GLIBC to version `2.35`, or use a newer operating system that has at least version `2.35` of GLIBC.

Access to microphone and audio in Windows version:

If you have a problems with audio or microphone in the non-binary PIP/Python version on Windows, check to see if FFmpeg is installed. If it's not, install it and add it to the PATH. You can find a tutorial on how to do this here: https://phoenixnap.com/kb/ffmpeg-windows. The binary version already includes FFmpeg.

Windows and VC++ Redistributable

On Windows, the proper functioning requires the installation of the `VC++ Redistributable`, which can be found on the Microsoft website:

https://learn.microsoft.com/en-us/cpp/windows/latest-supported-vc-redist

The libraries from this environment are used by `PySide6` - one of the base packages used by PyGPT. The absence of the installed libraries may cause display errors or completely prevent the application from running.

It may also be necessary to add the path `C:\path\to\venv\Lib\python3.x\site-packages\PySide6` to the `PATH` variable.

WebEngine/Chromium renderer and OpenGL problems

If you have a problems with `WebEngine / Chromium` renderer you can force the legacy mode by launching the app with command line arguments:

```
$ python3 run.py --legacy=1
```

and to force disable OpenGL hardware acceleration:

```
$ python3 run.py --disable-gpu=1
```

You can also manualy enable legacy mode by editing config file - open the `%WORKDIR%/config.json` config file in editor and set the following options:

```
"render.engine": "legacy",
"render.open_gl": false,
```

Other requirements

For operation, an internet connection is needed (for API connectivity), a registered OpenAI account, and an active API key that must be input into the program. Local models, such as `Llama3` do not require OpenAI account and any API keys.

Debugging and logging

Please go to `Debugging and Logging` section for instructions on how to log and diagnose issues in a more detailed manner.

Quick Start

Setting-up API Key(s)

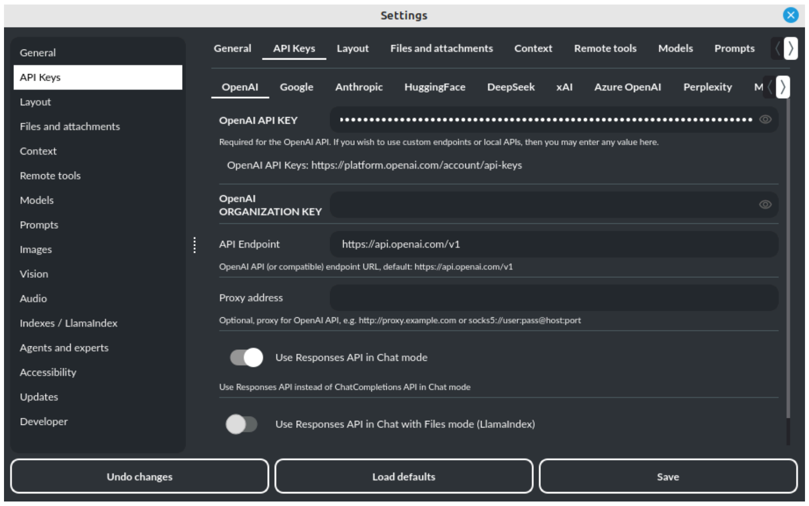
You can configure API keys for various providers, such as `OpenAI`, `Anthropic`, `Google`, `xAI`, `Perplexity`, `OpenRouter`, and more. This flexibility allows you to use different providers based on your needs.

During the initial setup, configure your API keys within the application.

To do so, navigate to the menu:

```
Config -> Settings -> API Keys
```

Here, you can add or manage API keys for any supported provider.



OpenAI API KEY ••

Required for the OpenAI API. If you wish to use custom endpoints or local APIs, then you may enter any value here.

OpenAI API Keys: https://platform.openai.com/account/api-keys

OpenAI ORGANIZATION KEY

API Endpoint https://api.openai.com/v1

OpenAI API (or compatible) endpoint URL, default: https://api.openai.com/v1

Proxy address

Optional, proxy for OpenAI API, e.g. http://proxy.example.com or socks5://user:pass@host:port

Use Responses API in Chat mode

Use Responses API instead of ChatCompletions API in Chat mode

Use Responses API in Chat with Files mode (LlamaIndex)

Configuring Provider

1. **Select the Provider:** Choose a tab with provider.
2. **Enter the API Key:** Paste the corresponding API key for the selected provider.

Example

- **OpenAI:** Obtain your API key by registering on the OpenAI website: https://platform.openai.com and navigating to https://platform.openai.com/account/api-keys.
- **Anthropic, Google, etc.:** Follow similar steps on their respective platforms.

Note

The ability to use models or services depends on your access level with the respective provider. If you wish to use custom API endpoints or local APIs that do not require API keys, simply enter any value into the API key field to bypass prompts about an empty key.

Work modes

Chat

+ Inline Vision and Image generation

In **PyGPT**, this mode mirrors `ChatGPT`, allowing you to chat with models like `GPT-5`, `GPT-4`, `o1`, `o3`, `Claude`, `Gemini`, `Grok`, `Perplexity (Sonar)`, `Deepseek`, and more. It works with the OpenAI SDK using the `Responses API` and `ChatCompletions API`. You can also use SDKs from Google GenAI, Anthropic, or xAI if the native SDK is enabled. You can set the endpoint for `ChatCompletions` in `Config -> Settings -> API Keys`.

Note

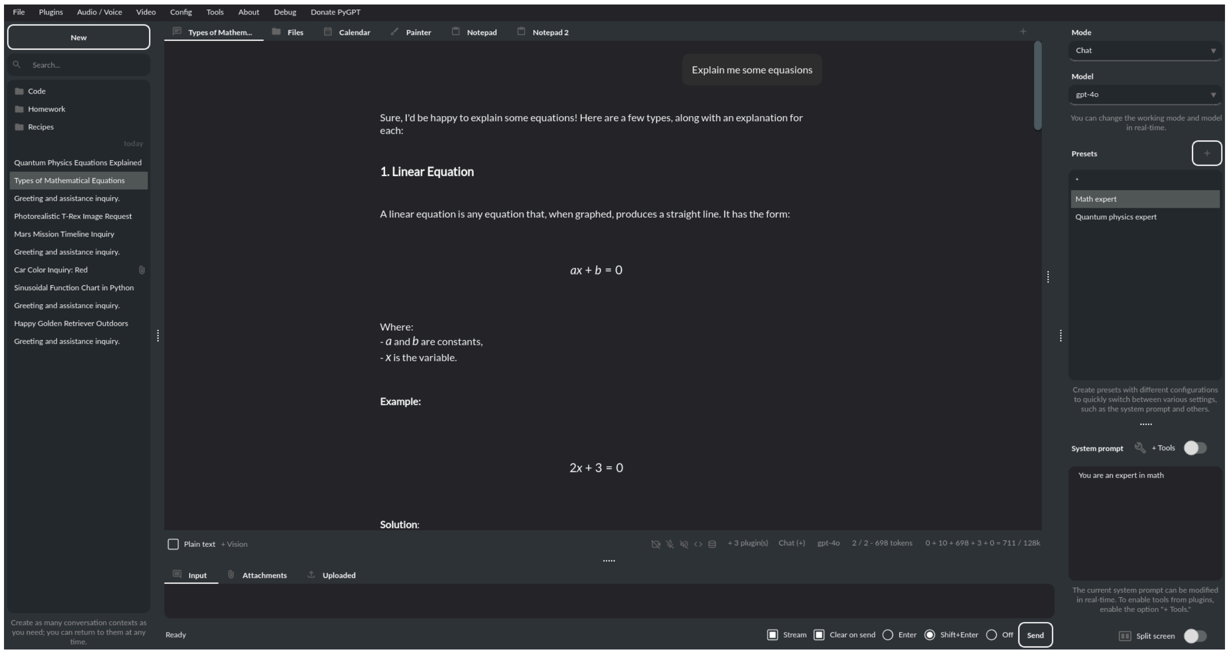
This mode uses the provider SDK directly. If there’s no native client built into the app, models like Sonar, or Llama3 are supported in Chat mode via LlamaIndex or OpenAI-compatible API endpoints. The app automatically switches to these endpoints when using non-OpenAI models. You can enable or disable the use of the native API SDK (per provider) in `Settings -> API Keys`. If the native SDK is disabled, the OpenAI SDK will be used via the compatible ChatCompletions API endpoint.

Currently built-in native clients:

- Anthropic SDK
- OpenAI SDK
- Google GenAI SDK
- xAI SDK

The main part of the interface is a chat window where you see your conversations. Below it is a message box for typing. On the right side, you can set up or change the model and system prompt. You can also save these settings as presets to easily switch between models or tasks.

Above where you type your messages, the interface shows you the number of tokens your message will use up as you type it – this helps to keep track of usage. There is also a feature to attach and upload files in this area. Go to the `Files and Attachments` section for more information on how to use attachments.



Quantum Physics Equations Explained
Types of Mathematical Equations
Greeting and assistance inquiry.
Photorealistic T-Rex Image Request
Mars Mission Timeline Inquiry
Greeting and assistance inquiry.
Car Color Inquiry: Red
Sinusoidal Function Chart in Python
Greeting and assistance inquiry.
Happy Golden Retriever Outdoors
Greeting and assistance inquiry.

Create as many conversation contexts as you need; you can return to them at any time.

Explain me some equasions

Sure, I'd be happy to explain some equations! Here are a few types, along with an explanation for each:

1. Linear Equation

A linear equation is any equation that, when graphed, produces a straight line. It has the form:

$$ax + b = 0$$

Where:
- a and b are constants,
- x is the variable.

Example:

$$2x + 3 = 0$$

Solution:

+ 3 plugin(s)   Chat (+)   gpt-4o   2 / 2 - 698 tokens   0 + 10 + 698 + 3 + 0 = 711 / 128k

Mode: Chat
Model: gpt-4o
You can change the working mode and model in real-time.

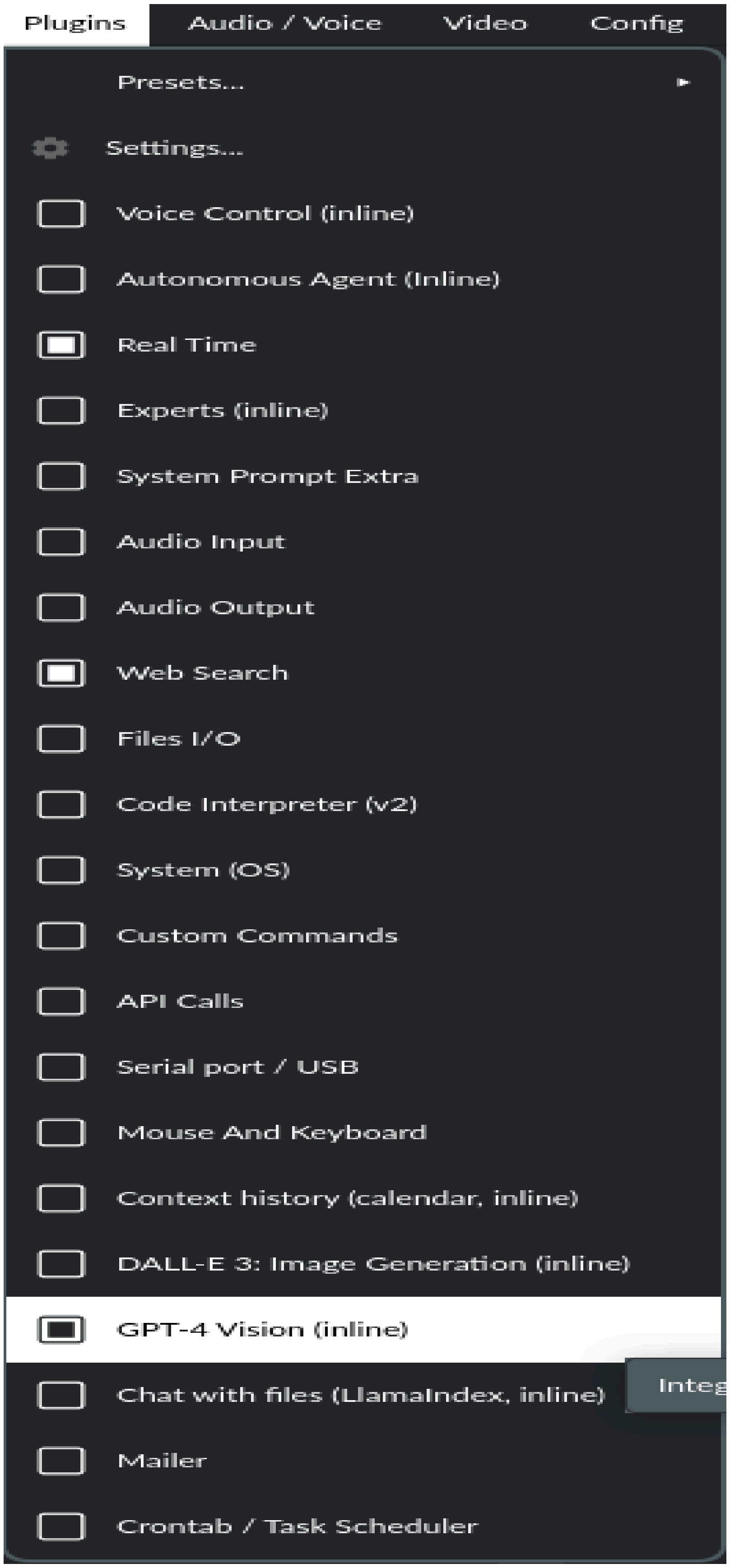
Presets
*
Math expert
Quantum physics expert

Create presets with different configurations to quickly switch between various settings, such as the system prompt and others.

System prompt
You are an expert in math

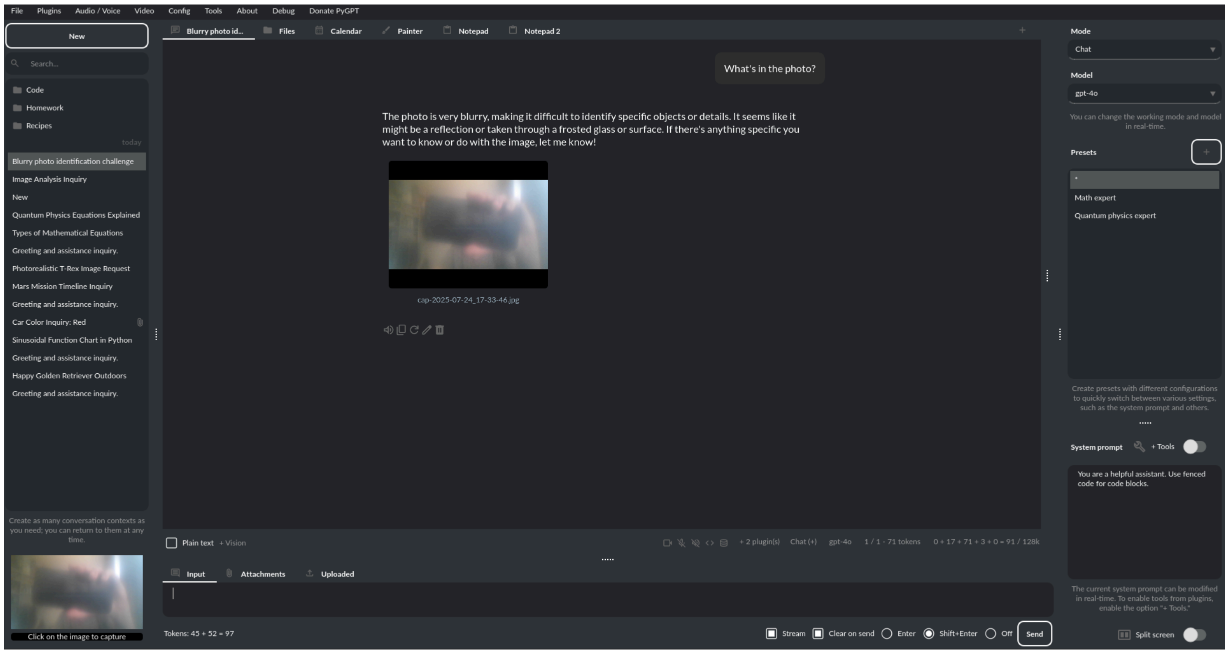
The current system prompt can be modified in real-time. To enable tools from plugins, enable the option "+ Tools."

Vision: If you want to send photos from your disk or images from your camera for analysis, and the selected model does not support Vision, you must enable the `Vision (inline)` plugin in the Plugins menu. This plugin allows you to send photos or images from your camera for analysis in any Chat mode.



- [ ] Voice Control (inline)
- [ ] Autonomous Agent (Inline)
- [x] Real Time
- [ ] Experts (inline)
- [ ] System Prompt Extra
- [ ] Audio Input
- [ ] Audio Output
- [x] Web Search
- [ ] Files I/O
- [ ] Code Interpreter (v2)
- [ ] System (OS)
- [ ] Custom Commands
- [ ] API Calls
- [ ] Serial port / USB
- [ ] Mouse And Keyboard
- [ ] Context history (calendar, inline)
- [ ] DALL-E 3: Image Generation (inline)
- [x] GPT-4 Vision (inline)
- [ ] Chat with files (LlamaIndex, inline)
- [ ] Mailer
- [ ] Crontab / Task Scheduler

With this plugin, you can capture an image with your camera or attach an image and send it for analysis to discuss the photograph:


What's in the photo?

The photo is very blurry, making it difficult to identify specific objects or details. It seems like it might be a reflection or taken through a frosted glass or surface. If there's anything specific you want to know or do with the image, let me know!

cap-2025-07-24_17-33-46.jpg

Blurry photo identification challenge; Image Analysis Inquiry; New; Quantum Physics Equations Explained; Types of Mathematical Equations; Greeting and assistance inquiry.; Photorealistic T-Rex Image Request; Mars Mission Timeline Inquiry; Greeting and assistance inquiry.; Car Color Inquiry: Red; Sinusoidal Function Chart in Python; Greeting and assistance inquiry.; Happy Golden Retriever Outdoors; Greeting and assistance inquiry.

Mode: Chat — Model: gpt-4o

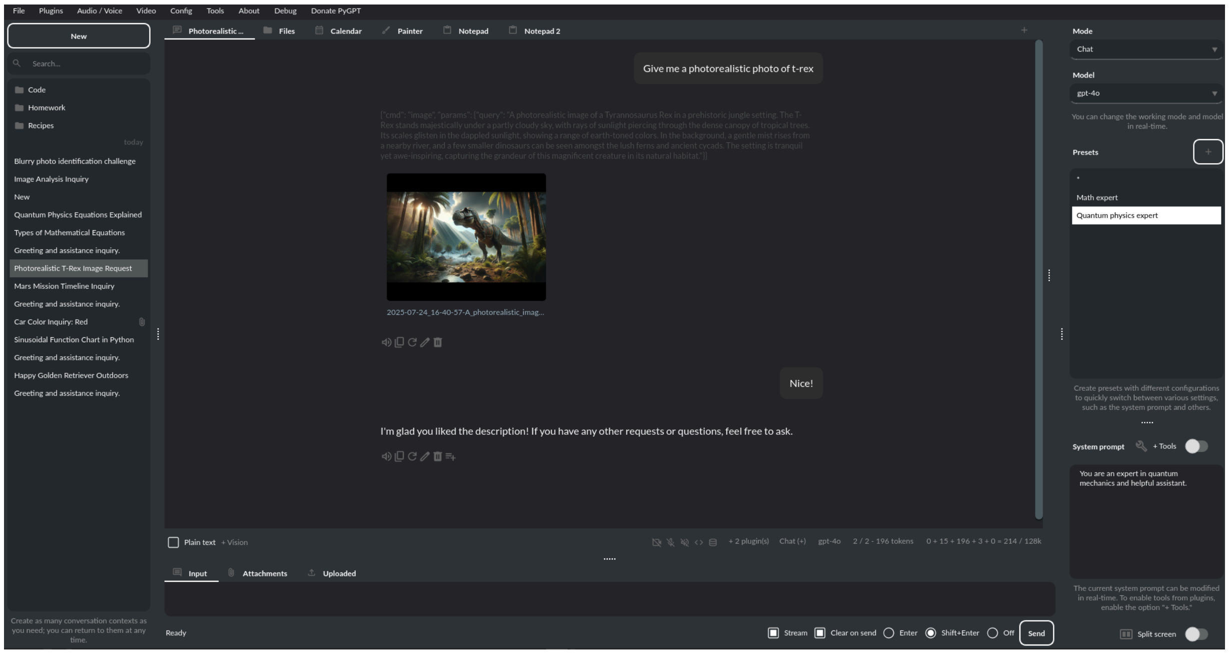
System prompt: You are a helpful assistant. Use fenced code for code blocks.

+ 2 plugin(s)   Chat (+)   gpt-4o   1 / 1 - 71 tokens   0 + 17 + 71 + 3 + 0 = 91 / 128k

Tokens: 45 + 52 = 97

Click on the image to capture

Image generation: If you want to generate images directly in chat you must enable plugin `Image generation (inline)` in the Plugins menu. Plugin allows you to generate images in Chat mode:


Give me a photorealistic photo of t-rex

{"cmd": "image", "params": {"query": "A photorealistic image of a Tyrannosaurus Rex in a prehistoric jungle setting. The T-Rex stands majestically under a partly cloudy sky, with rays of sunlight piercing through the dense canopy of tropical trees. Its scales glisten in the dappled sunlight, showing a range of earth-toned colors. In the background, a gentle mist rises from a nearby river, and a few smaller dinosaurs can be seen amongst the lush ferns and ancient cycads. The setting is tranquil yet awe-inspiring, capturing the grandeur of this magnificent creature in its natural habitat."}}

2025-07-24_16-40-57-A_photorealistic_imag...

Nice!

I'm glad you liked the description! If you have any other requests or questions, feel free to ask.

Mode: Chat — Model: gpt-4o

System prompt: You are an expert in quantum mechanics and helpful assistant.

+ 2 plugin(s)   Chat (+)   gpt-4o   2 / 2 - 196 tokens   0 + 15 + 196 + 3 + 0 = 214 / 128k

Ready

Photorealistic ...
You can change the working mode and model in real-time.
Create presets with different configurations to quickly switch between various settings, such as the system prompt and others.
The current system prompt can be modified in real-time. To enable tools from plugins, enable the option "+ Tools."
Create as many conversation contexts as you need; you can return to them at any time.

Chat with Files (LlamaIndex)

This mode enables chat interaction with your documents and entire context history through conversation. It seamlessly incorporates `LlamaIndex` into the chat interface, allowing for immediate querying of your indexed documents.

Tip

If you do not want to call tools/commands, disable the checkbox `+Tools`. It will speed up the response time when using local models. You can also enable the ReAct agent for tool calls in: `Settings -> Indexes / LlamaIndex -> Chat -> Use ReAct agent for Tool calls in Chat with Files mode`. Stream mode is disabled if the ReAct agent and `+Tools` checkbox are active.

Querying single files

You can also query individual files "on the fly" using the `query_file` command from the `Files I/O` plugin. This allows you to query any file by simply asking a question about that file. A temporary index will be created in memory for the file being queried, and an answer will be returned from it. From version `2.1.9` similar command is available for querying web and external content: `Directly query web content with LlamaIndex`.

For example:

If you have a file: `data/my_cars.txt` with content `My car is red.`

You can ask for: `Query the file my_cars.txt about what color my car is.`

And you will receive the response: `Red`.

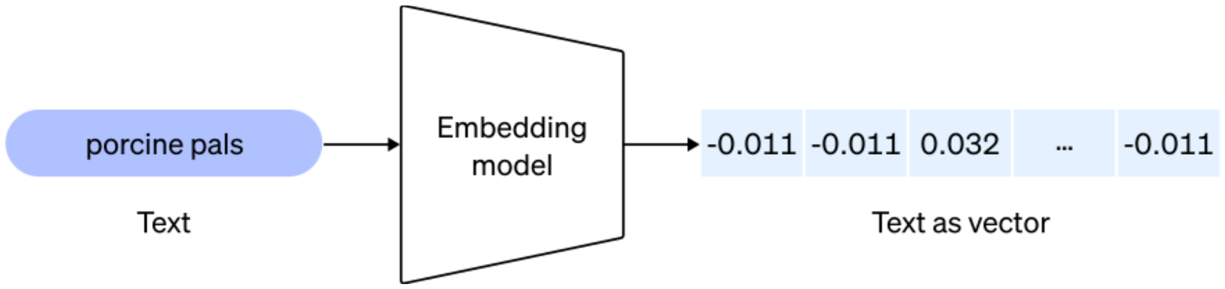
Note: this command indexes the file only for the current query and does not persist it in the database. To store queried files also in the standard index you must enable the option `Auto-index readed files` in plugin settings. Remember to enable `+ Tools` checkbox to allow usage of tools and commands from plugins.

Using Chat with Files mode

In this mode, you are querying the whole index, stored in a vector store database. To start, you need to index (embed) the files you want to use as additional context. Embedding transforms your text data into vectors. If you're unfamiliar with embeddings and how they work, check out this article:

https://stackoverflow.blog/2023/11/09/an-intuitive-introduction-to-text-embeddings/

For a visualization from OpenAI's page, see this picture:



Source: https://cdn.openai.com/new-and-improved-embedding-model/draft-20221214a/vectors-3.svg

To index your files, simply copy or upload them into the `data` directory and initiate indexing (embedding) by clicking the `Index all` button, or right-click on a file and select `Index...`. Additionally, you have the option to utilize data from indexed files in any Chat mode by activating the `Chat with Files (LlamaIndex, inline)` plugin.

Built-in file loaders:

Files:

- CSV files (csv)
- Epub files (epub)
- Excel .xlsx spreadsheets (xlsx)
- HTML files (html, htm)
- IPYNB Notebook files (ipynb)
- Image (vision) (jpg, jpeg, png, gif, bmp, tiff, webp)
- JSON files (json)
- Markdown files (md)
- PDF documents (pdf)
- Plain-text files (txt)
- Video/audio (mp4, avi, mov, mkv, webm, mp3, mpeg, mpga, m4a, wav)
- Word .docx documents (docx)
- XML files (xml)

Web/external content:

- Bitbucket
- ChatGPT Retrieval Plugin
- GitHub Issues
- GitHub Repository
- Google Calendar
- Google Docs
- Google Drive
- Google Gmail
- Google Keep
- Google Sheets
- Microsoft OneDrive
- RSS
- SQL Database
- Sitemap (XML)
- Twitter/X posts
- Webpages (crawling any webpage content)
- YouTube (transcriptions)

You can configure data loaders in `Settings / LlamaIndex / Data Loaders` by providing list of keyword arguments for specified loaders. You can also develop and provide your own custom loader and register it within the application.

LlamaIndex is also integrated with context database - you can use data from database (your context history) as additional context in discussion. Options for indexing existing context history or enabling real-time indexing new ones (from database) are available in `Settings / LlamaIndex` section.

Warning

Remember that when indexing content, API calls to the embedding model are used. Each indexing consumes additional tokens. Always control the number of tokens used on the provider's page.

Tip

Using the Chat with Files mode, you have default access to files manually indexed from the /data directory. However, you can use additional context by attaching a file - such additional context from the attachment does not land in the main index, but only in a temporary one, available only for the given conversation.

Token limit: When you use `Chat with Files` in non-query mode, LlamaIndex adds extra context to the system prompt. If you use a plugins (which also adds more instructions to system prompt), you might go over the maximum number of tokens allowed. If you get a warning that says you've used too many tokens, turn off plugins you're not using or turn off the "+ Tools" option to reduce the number of tokens used by the system prompt.

Available vector stores (provided by `LlamaIndex`):

- ChromaVectorStore
- ElasticsearchStore
- PinecodeVectorStore
- RedisVectorStore
- SimpleVectorStore

You can configure selected vector store by providing config options like `api_key`, etc. in `Settings -> LlamaIndex` window. See the section: `Configuration / Vector stores` for configuration reference.

Configuring data loaders

In the `Settings -> LlamaIndex -> Data loaders` section you can define the additional keyword arguments to pass into data loader instance. See the section: `Configuration / Data Loaders` for configuration reference.

Chat with Audio

This mode works like the Chat mode but with native support for audio input and output using a Realtime and Live APIs. In this mode, audio input and output are directed to and from the model directly, without the use of external plugins. This enables faster and better audio communication.

Currently, in beta.

At this moment, only OpenAI real-time models (via the Realtime API) and Google Gemini real-time models (via the Live API) are supported.

Research

This mode (when using Sonar and R1 models) operates using the Perplexity API: https://perplexity.ai.

It allows for deep web searching and utilizes Sonar models, available in `Perplexity AI`.

It requires a Perplexity API key, which can be generated at: https://perplexity.ai.

From version `2.5.27` also OpenAI deep-research models are available in this mode.

Completion

An older mode of operation that allows working in the standard text completion mode. However, it allows for a bit more flexibility with the text by enabling you to initiate the entire discussion in any way you like.

Similar to chat mode, on the right-hand side of the interface, there are convenient presets. These allow you to fine-tune instructions and swiftly transition between varied configurations and pre-made prompt templates.

Additionally, this mode offers options for labeling the AI and the user, making it possible to simulate dialogues between specific characters - for example, you could create a conversation between Batman and the Joker, as predefined in the prompt. This feature presents a range of creative possibilities for setting up different conversational scenarios in an engaging and exploratory manner.

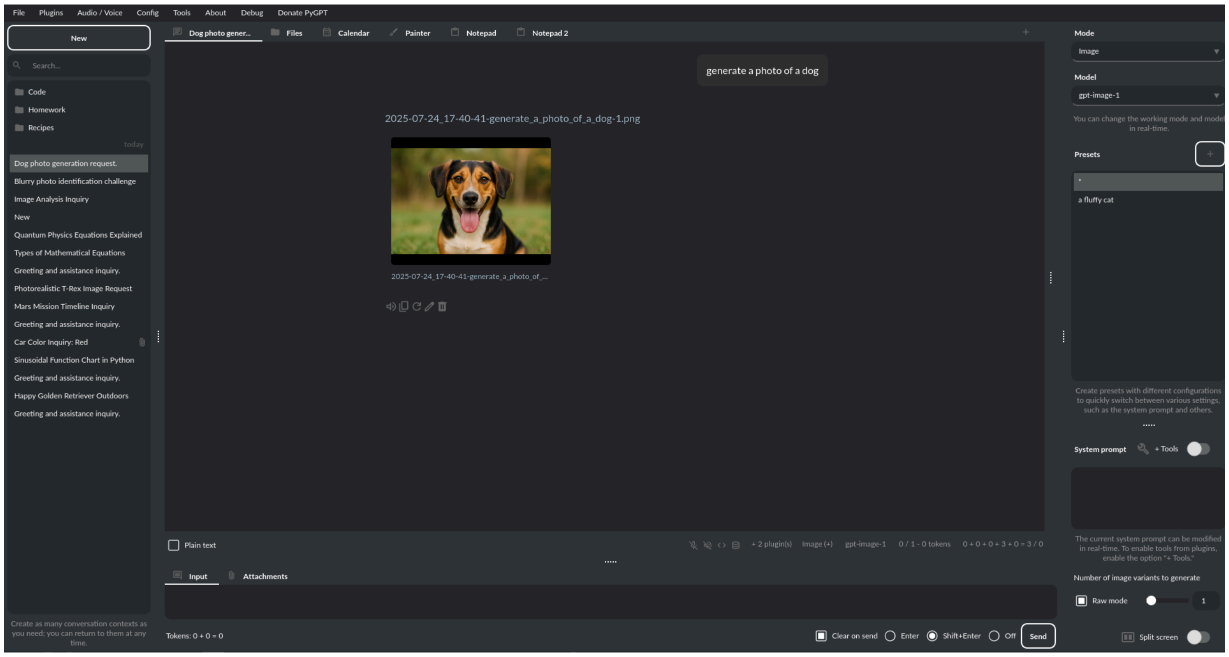
Note

From version `2.0.107` the davinci models are deprecated and has been replaced with `gpt-3.5-turbo-instruct` model.

Image and video generation

OpenAI DALL-E 3 / Google Imagen 3 and 4

PyGPT enables quick and easy image creation with image models like `DALL-E 3`, `gpt-image-1` or `Google Imagen`. Generating images is akin to a chat conversation - a user's prompt triggers the generation, followed by downloading, saving to the computer, and displaying the image onscreen. You can send raw prompt to the model in `Image generation` mode or ask the model for the best prompt.


File
Plugins
Audio / Voice
Config
Tools
About
Debug
Donate PyGPT
New
Search...
Code
Homework
Recipes
today
Dog photo generation request.
Blurry photo identification challenge
Image Analysis Inquiry
New
Quantum Physics Equations Explained
Types of Mathematical Equations
Greeting and assistance inquiry.
Photorealistic T-Rex Image Request
Mars Mission Timeline Inquiry
Greeting and assistance inquiry.
Car Color Inquiry: Red
Sinusoidal Function Chart in Python
Greeting and assistance inquiry.
Happy Golden Retriever Outdoors
Greeting and assistance inquiry.
Create as many conversation contexts as you need; you can return to them at any time.
Dog photo gener...
Files
Calendar
Painter
Notepad
Notepad 2
Plain text
Input
Attachments
Tokens: 0 + 0 = 0
Clear on send
Enter
Shift+Enter
Off
Send
Mode
Model
You can change the working mode and model in real-time.
Presets
a fluffy cat
Create presets with different configurations to quickly switch between various settings, such as the system prompt and others.
System prompt
+ Tools
The current system prompt can be modified in real-time. To enable tools from plugins, enable the option "+ Tools."
Number of image variants to generate
Raw mode
Split screen

generate a photo of a dog

2025-07-24_17-40-41-generate_a_photo_of_a_dog-1.png

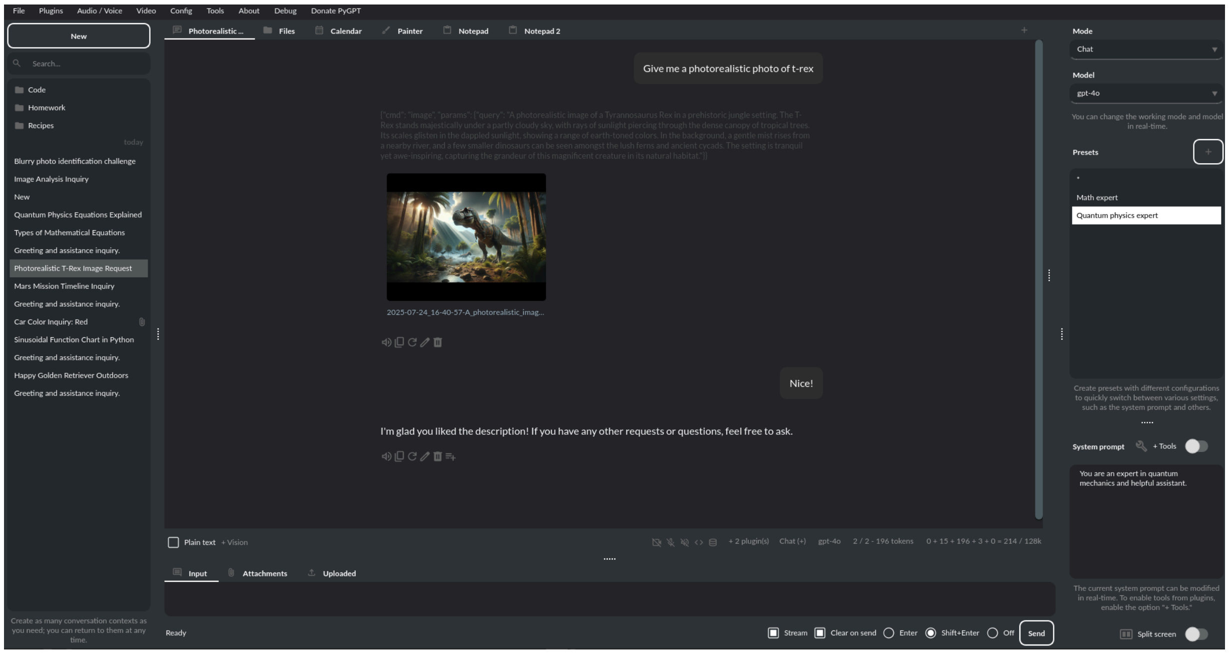
Mode: Image

Model: gpt-image-1

+ 2 plugin(s) Image (+) gpt-image-1 0 / 1 - 0 tokens 0 + 0 + 0 + 3 + 0 = 3 / 0

Image generation using image models is also available in every mode via plugin `Image Generation (inline)`. Just ask any model, in any mode, like e.g. GPT or Gemini to generate an image and it will do it inline, without need to mode change.

If you want to generate images directly in chat you must enable plugin **Image generation (inline)** in the Plugins menu. Plugin allows you to generate images in Chat mode:



Blurry photo identification challenge
Image Analysis Inquiry
New
Quantum Physics Equations Explained
Types of Mathematical Equations
Greeting and assistance inquiry.
Photorealistic T-Rex Image Request
Mars Mission Timeline Inquiry
Greeting and assistance inquiry.
Car Color Inquiry: Red
Sinusoidal Function Chart in Python
Greeting and assistance inquiry.
Happy Golden Retriever Outdoors
Greeting and assistance inquiry.

Create as many conversation contexts as you need; you can return to them at any time.

Give me a photorealistic photo of t-rex

```
{"cmd": "image", "params": {"query": "A photorealistic image of a Tyrannosaurus Rex in a prehistoric jungle setting. The T-Rex stands majestically under a partly cloudy sky, with rays of sunlight piercing through the dense canopy of tropical trees. Its scales glisten in the dappled sunlight, showing a range of earth-toned colors. In the background, a gentle mist rises from a nearby river, and a few smaller dinosaurs can be seen amongst the lush ferns and ancient cycads. The setting is tranquil yet awe-inspiring, capturing the grandeur of this magnificent creature in its natural habitat."}}
```

2025-07-24_16-40-57-A_photorealistic_imag...

Nice!

I'm glad you liked the description! If you have any other requests or questions, feel free to ask.

Plain text + Vision | + 2 plugin(s) Chat (+) gpt-4o 2 / 2 - 196 tokens 0 + 15 + 196 + 3 + 0 = 214 / 128k

Ready | Stream [✓] Clear on send [✓] Enter [✓ Shift+Enter] Off

Mode: Chat

Model: gpt-4o

You can change the working mode and model in real-time.

Presets: *, Math expert, Quantum physics expert

Create presets with different configurations to quickly switch between various settings, such as the system prompt and others.

System prompt + Tools

You are an expert in quantum mechanics and helpful assistant.

The current system prompt can be modified in real-time. To enable tools from plugins, enable the option "+ Tools."

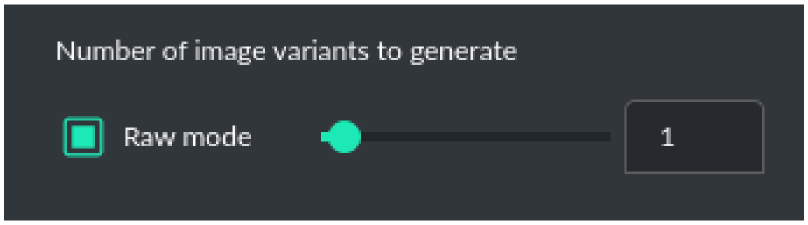
Video generation: From version `2.6.32`, video generation (using `Google Veo 3`) is also available.

Multiple variants

You can generate up to **4 different variants** (DALL-E 2) for a given prompt in one session. DALL-E 3 allows one image. To select the desired number of variants to create, use the slider located in the right-hand corner at the bottom of the screen. This replaces the conversation temperature slider when you switch to image generation mode.

Raw mode

There is an option for switching prompt generation mode.



If **Raw Mode** is enabled, a model will receive the prompt exactly as you have provided it. If **Raw Mode** is disabled, a model will generate the best prompt for you based on your instructions.

Image storage

Once you've generated an image, you can easily save it anywhere on your disk by right-clicking on it. You also have the options to delete it or view it in full size in your web browser.

Tip

Use presets to save your prepared prompts. This lets you quickly use them again for generating new images later on.

The app keeps a history of all your prompts, allowing you to revisit any session and reuse previous prompts for creating new images.

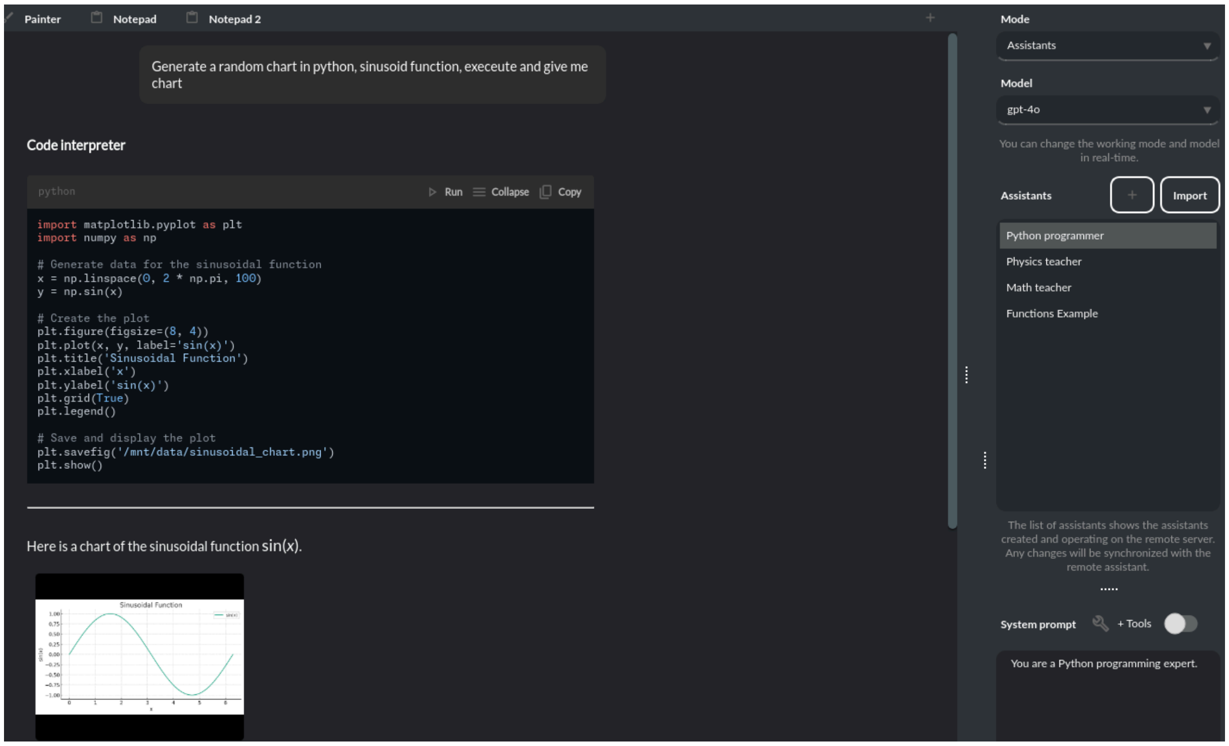
Images are stored in `img` directory in PyGPT's user data folder.

Assistants

This mode uses the OpenAI's **Assistants API**.

This mode expands on the basic chat functionality by including additional external tools like a `Code Interpreter` for executing code, `Retrieval Files` for accessing files, and custom `Functions` for enhanced interaction and integration with other APIs or services. In this mode, you can easily upload and download files. **PyGPT** streamlines file management, enabling you to quickly upload documents and manage files created by the model.

Setting up new assistants is simple - a single click is all it takes, and they instantly sync with the `OpenAI API`. Importing assistants you've previously created with OpenAI into **PyGPT** is also a seamless process.



Generate a random chart in python, sinusoid function, exeecute and give me chart

Code interpreter

```python
import matplotlib.pyplot as plt
import numpy as np

# Generate data for the sinusoidal function
x = np.linspace(0, 2 * np.pi, 100)
y = np.sin(x)

# Create the plot
plt.figure(figsize=(8, 4))
plt.plot(x, y, label='sin(x)')
plt.title('Sinusoidal Function')
plt.xlabel('x')
plt.ylabel('sin(x)')
plt.grid(True)
plt.legend()

# Save and display the plot
plt.savefig('/mnt/data/sinusoidal_chart.png')
plt.show()
```

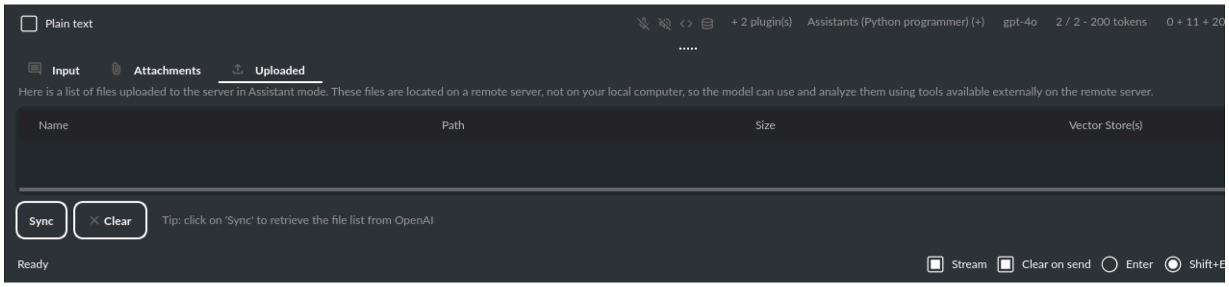
Here is a chart of the sinusoidal function sin(x).

You can change the working mode and model in real-time.

The list of assistants shows the assistants created and operating on the remote server. Any changes will be synchronized with the remote assistant.

You are a Python programming expert.

In Assistant mode you are allowed to storage your files in remote vector store (per Assistant) and manage them easily from app:



Here is a list of files uploaded to the server in Assistant mode. These files are located on a remote server, not on your local computer, so the model can use and analyze them using tools available externally on the remote server.

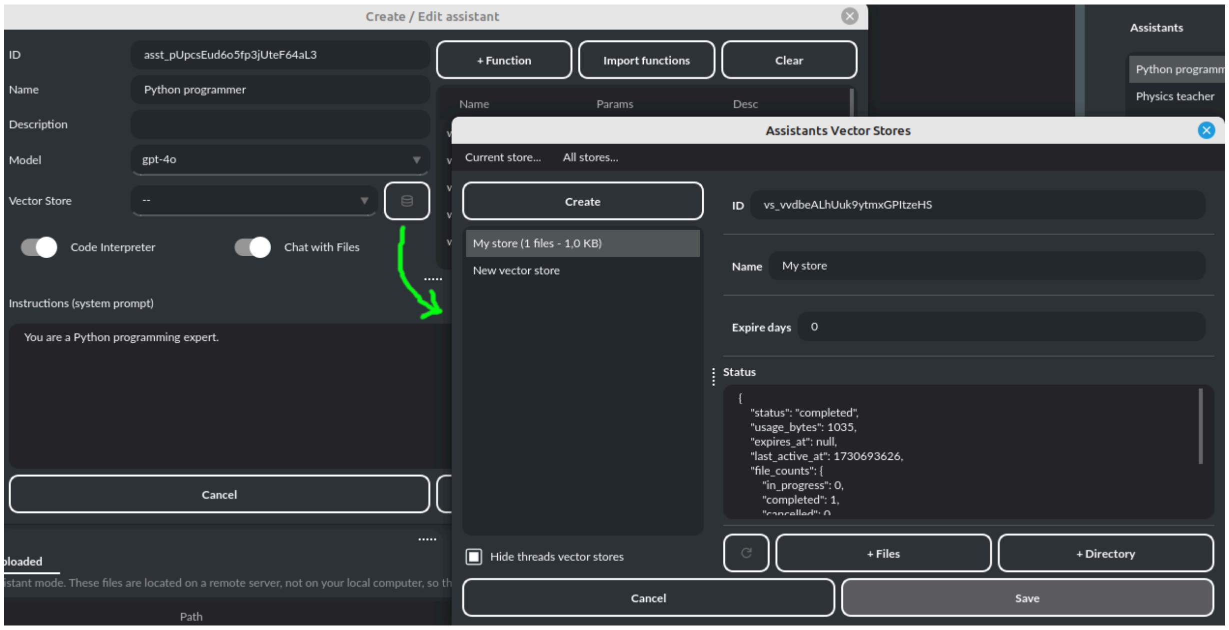
Tip: click on 'Sync' to retrieve the file list from OpenAI

Vector stores (via Assistants API)

Assistant mode supports the use of external vector databases offered by the OpenAI API. This feature allows you to store your files in a database and then search them using the Assistant's API. Each assistant can be linked to one vector database—if a database is linked, all files uploaded in this mode will be stored in the linked vector database. If an assistant does not have a linked vector database, a temporary database is automatically created during the file upload, which is accessible only in the current thread. Files from temporary databases are automatically deleted after 7 days.

To enable the use of vector stores, enable the `Chat with Files` checkbox in the Assistant settings. This enables the `File search` tool in Assistants API.

To manage external vector databases, click the DB icon next to the vector database selection list in the Assistant creation and editing window (screen below). In this management window, you can create a new vector database, edit an existing one, or import a list of all existing databases from the OpenAI server:



ID: asst_pUpcsEud6o5fp3jUteF64aL3
Name: Python programmer
Description:
Model: gpt-4o
Vector Store: --
Code Interpreter
Chat with Files
Instructions (system prompt): You are a Python programming expert.

ploaded
istant mode. These files are located on a remote server, not on your local computer, so th

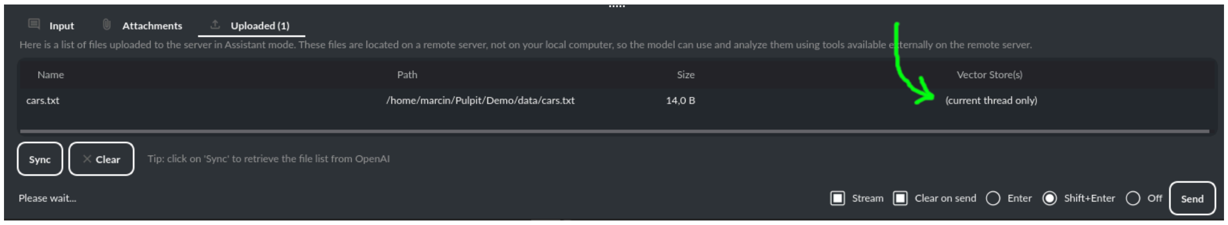
My store (1 files - 1,0 KB)
New vector store

ID: vs_vvdbeALhUuk9ytmxGPltzeHS
Name: My store
Expire days: 0
Status:

```
{
    "status": "completed",
    "usage_bytes": 1035,
    "expires_at": null,
    "last_active_at": 1730693626,
    "file_counts": {
        "in_progress": 0,
        "completed": 1,
        "cancelled": 0
```

You can define, using `Expire days`, how long files should be automatically kept in the database before deletion (as storing files on OpenAI incurs costs). If the value is set to 0, files will not be automatically deleted.

The vector database in use will be displayed in the list of uploaded files, on the field to the right—if a file is stored in a database, the name of the database will be displayed there; if not, information will be shown indicating that the file is only accessible within the thread.



Here is a list of files uploaded to the server in Assistant mode. These files are located on a remote server, not on your local computer, so the model can use and analyze them using tools available externally on the remote server.

Name	Path	Size	Vector Store(s)
cars.txt	/home/marcin/Pulpit/Demo/data/cars.txt	14,0 B	(current thread only)

Tip: click on 'Sync' to retrieve the file list from OpenAI

Please wait...

[✓ Stream] [✓ Clear on send] Enter [✓ Shift+Enter] Off

Agent (LlamaIndex)

Mode that allows the use of agents offered by `LlamaIndex`.

Includes built-in agents (Workflow):

- FunctionAgent
- ReAct
- Structured Planner (sub-tasks)
- CodeAct (connected to Code Interpreter plugin)
- Supervisor + worker

Includes built-in agents (Legacy):

- OpenAI Assistants

In the future, the list of built-in agents will be expanded.

You can create your own types (workflows/patterns) using the built-in visual node-based editor found in the `Tools -> Agents Builder`.

You can also create your own agent by creating a new provider that inherits from `pygpt_net.provider.agents.base`.

Tools and Plugins

In this mode, all commands from active plugins are available (commands from plugins are automatically converted into tools for the agent on-the-fly).

RAG - using indexes

If an index is selected in the agent preset, a tool for reading data from the index is automatically added to the agent, creating a RAG automatically.

Multimodality is currently unavailable, only text is supported. Vision support will be added in the future.

Loop / Evaluate Mode

You can run the agent in autonomous mode, in a loop, and with evaluation of the current output. When you enable the `Loop / Evaluate` checkbox, after the final response is given, the quality of the answer will be rated on a percentage scale of `0% to 100%` by another agent. If the response receives a score lower than the one expected (set using a slider at the bottom right corner of the screen, with a default value `75%`), a prompt will be sent to the agent requesting improvements and enhancements to the response.

Setting the expected (required) score to `0%` means that the response will be evaluated every time the agent produces a result, and it will always be prompted to self-improve its answer. This way, you can put the agent in an autonomous loop, where it will continue to operate until it succeeds.

You can choose between two methods of evaluation:

- By the percentage of tasks completed
- By the accuracy (score) of the final response

You can set the limit of steps in such a loop by going to `Settings -> Agents and experts -> LlamaIndex agents -> Max evaluation steps`. The default value is `3`, meaning the agent will only make three attempts to improve or correct its answer. If you set the limit to zero, there will be no limit, and the agent can operate in this mode indefinitely (watch out for tokens!).

You can change the prompts used for evaluating the response in `Settings -> Prompts -> Agent: evaluation prompt in loop`. Here, you can adjust it to suit your needs, for example, by defining more or less critical feedback for the responses received.

Agent (OpenAI)

The mode operates on the `openai-agents` library integrated into the application:

https://github.com/openai/openai-agents-python

It allows running agents for OpenAI models and models compatible with the OpenAI.

In this mode, you can use pre-configured Experts in Expert mode presets - they will be launched as agents (in the `openai_agents_experts` type, which allows launching one main agent and subordinate agents to which queries will be appropriately directed).

Agent types (workflows/patterns):

- `Agent with experts` - uses attached experts as sub-agents

- `Agent with experts + feedback` - uses attached experts as sub-agents + feedback agent in a loop
- `Agent with feedback` - single agent + feedback agent in a loop
- `Planner` - planner agent, 3 sub-agents inside: planner, base agent + feedback
- `Research bot` - researcher, 3 sub-agents inside: planner, searcher and writer as base agent
- `Simple agent` - a single agent.
- `Evolve` - in each generation (cycle), the best response from a given parent agent is selected; in the next generation, the cycle repeats.
- `B2B` - bot-to-bot communication, involving two bots interacting with each other while keeping a human in the loop.
- `Supervisor + Worker` - one agent (supervisor) acts as a bridge between the user and the second agent (worker). The user provides a query to the supervisor, who then sends instructions to the worker until the task is completed by the worker.

You can create your own types (workflows/patterns) using the built-in visual node-based editor found in the `Tools -> Agents Builder`.

There are also predefined presets added as examples:

- `Coder`
- `Experts agent`
- `Planner`
- `Researcher`
- `Simple agent`
- `Writer with Feedback`
- `2 bots`
- `Supervisor + worker`

In the Agents (OpenAI) mode, all remote tools are available for the base agent according to the configuration in the Config -> Settings -> Remote tools menu.

Remote tools for experts can be selected separately for each expert in the preset configuration.

Local tools (from plugins) are available for agents and experts according to the enabled plugins, as in other modes.

In agents with feedback and plans, tools can be allowed in a preset configuration for each agent. They also have separate prompts that can be configured in presets.

Description of how different types of agents work:

Below is a pattern for how different types of agents work. You can use these patterns to create agents for different tasks by modifying the appropriate prompts in the preset for the specific task.

Simple Agent

- The agent completes its task and then stops working.

Agent with Feedback

- The first agent answers a question.
- The second agent (feedback) evaluates the answer and, if necessary, goes back to the first agent to enforce corrections.
- The cycle repeats until the feedback agent is satisfied with the evaluation.

Agent with Experts

- The agent completes the assigned task on its own or delegates it to the most suitable expert (another agent).

Agent with Experts + Feedback

- The first agent answers a question or delegates it to the most suitable expert.

- The second agent (feedback) evaluates and, if necessary, goes back to the first agent to enforce corrections.
- The cycle repeats until the feedback agent is satisfied with the evaluation.

Research Bot

- The first agent (planner) prepares a list of phrases to search.
- The second agent (search) finds information based on the phrases and creates a summary.
- The third agent (writer) prepares a report based on the summary.

Planner

- The first agent (planner) breaks down a task into sub-tasks and sends the list to the second agent.
- The second agent performs the task based on the prepared task list.
- The third agent, responsible for feedback, evaluates, requests corrections if needed, and sends the request back to the first agent. The cycle repeats.

Evolve

- You select the number of agents (parents) to operate in each generation (iteration).
- Each agent prepares a separate answer to a question.
- The best agent (producing the best answer) in a generation is selected by the next agent (chooser).
- Another agent (feedback) verifies the best answer and suggests improvements.
- A request for improving the best answer is sent to a new pair of agents (new parents).
- From this new pair, the best answer is selected again in the next generation, and the cycle repeats.

B2B

- A human provides a topic for discussion.
- Bot 1 generates a response and sends it to Bot 2.
- Bot 2 receives the response from Bot 1 as input, provides an answer, and sends the response back to Bot 1 as its input. This cycle repeats.
- The human can interrupt the loop at any time and update the entire discussion.

Supervisor + Worker

- A human provides a query to the Supervisor.
- The Supervisor prepares instructions for the Worker and sends them to the Worker.
- The Worker completes the task and returns the result to the Supervisor.
- If the task is completed, the Supervisor returns the result to the user. If not, the Supervisor sends another instruction to the Worker to complete the task or asks the user if there are any questions.
- The cycle repeats until the task is completed.

Tip

Starting from version `2.5.97`, you can assign and use Experts in all of the agent types.

Limitations:

- When the *Computer use* tool is selected for an expert or when the *computer-use* model is chosen, all other tools will not be available for that model.

Agent (Autonomous)

This is an older version of the Agent mode, still available as legacy. However, it is recommended to use the newer mode: `Agent (LlamaIndex)`.

Warning

Please use this mode with caution! - autonomous mode, when connected with other plugins, may produce unexpected results!

The mode activates autonomous mode, where AI begins a conversation with itself. You can set this loop to run for any number of iterations. Throughout this sequence, the model will engage in self-dialogue, answering his own questions and comments, in order to find the best possible solution, subjecting previously generated steps to criticism.

Warning

Setting the number of run steps (iterations) to `0` activates an infinite loop which can generate a large number of requests and cause very high token consumption, so use this option with caution! Confirmation will be displayed every time you run the infinite loop.

This mode is similar to `Auto-GPT` - it can be used to create more advanced inferences and to solve problems by breaking them down into subtasks that the model will autonomously perform one after another until the goal is achieved.

You can create presets with custom instructions for multiple agents, incorporating various workflows, instructions, and goals to achieve.

All plugins are available for agents, so you can enable features such as file access, command execution, web searching, image generation, vision analysis, etc., for your agents. Connecting agents with plugins can create a fully autonomous, self-sufficient system. All currently enabled plugins are automatically available to the Agent.

When the `Auto-stop` option is enabled, the agent will attempt to stop once the goal has been reached.

In opposition to `Auto-stop`, when the `Always continue...` option is enabled, the agent will use the "always continue" prompt to generate additional reasoning and automatically proceed to the next step, even if it appears that the task has been completed.

Options

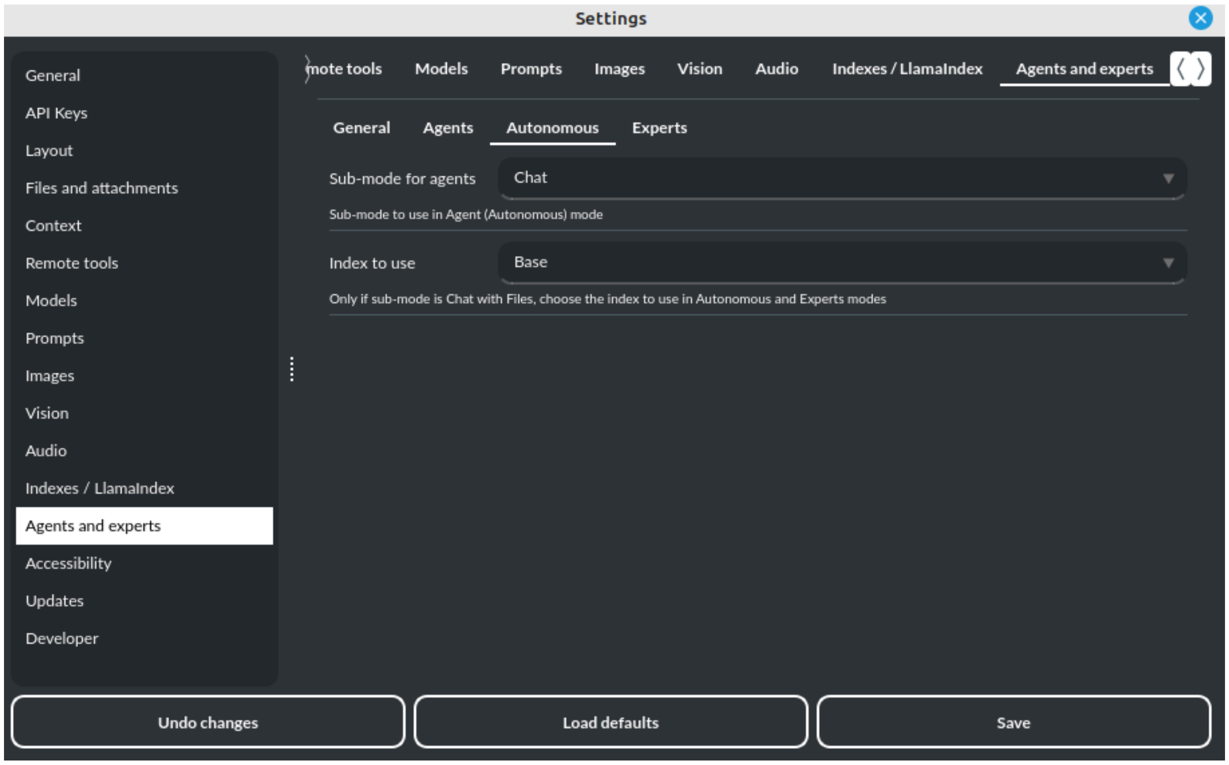
The agent is essentially a **virtual** mode that internally sequences the execution of a selected underlying mode. You can choose which internal mode the agent should use in the settings:

```
Settings / Agent (autonomous) / Sub-mode to use
```

Default mode is: `Chat`.

If you want to use the LlamaIndex mode when running the agent, you can also specify which index `LlamaIndex` should use with the option:

```
Settings / Agent (autonomous) / Index to use
```


Sub-mode for agents: Chat

Sub-mode to use in Agent (Autonomous) mode

Index to use: Base

Only if sub-mode is Chat with Files, choose the index to use in Autonomous and Experts modes

Experts (Co-op, co-operation mode)

Expert mode allows for the creation of experts (using presets) and then consulting them during a conversation. In this mode, a primary base context is created for conducting the conversation. From within this context, the model can make requests to an expert to perform a task and return the results to the main thread. When an expert is called in the background, a separate context is created for them with their own memory. This means that each expert, during the life of one main context, also has access to their own memory via their separate, isolated context.

In simple terms - you can imagine an expert as a separate, additional instance of the model running in the background, which can be called at any moment for assistance, with its own context and memory, as well as its own specialized instructions in a given subject.

Experts do not share contexts with one another, and the only point of contact between them is the main conversation thread. In this main thread, the model acts as a manager of experts, who can exchange data between them as needed.

An expert is selected based on the name in the presets; for example, naming your expert as: ID = python_expert, name = “Python programmer” will create an expert whom the model will attempt to invoke for matters related to Python programming. You can also manually request to refer to a given expert:

```
Call the Python expert to generate some code.
```

Experts can be activated or deactivated - to enable or disable use RMB context menu to select the `Enable/Disable` options from the presets list. Only enabled experts are available to use in the thread.

Experts can also be used in `Agent (autonomous)` mode - by creating a new agent using a preset. Simply move the appropriate experts to the active list to automatically make them available for use by the agent.

You can also use experts in “inline” mode - by activating the `Experts (inline)` plugin. This allows for the use of experts in any mode, such as normal chat.

Expert mode, like agent mode, is a “virtual” mode - you need to select a target mode of operation for it, which can be done in the settings at `Settings / Agent (autonomous) / Sub-mode for experts`.

You can also ask for a list of active experts at any time:

```
Give me a list of active experts.
```

Computer use

This mode allows for autonomous computer control.

In this mode, the model takes control of the mouse and keyboard and can navigate within the user's environment.

The `Computer use` remote tool is used here: https://platform.openai.com/docs/guides/tools-computer-use, combined with the `Mouse and Keyboard` plugin.

Example of use:

```
Click on the Start Menu to open it, search for the Notepad in
the list, and run it.
```

You can change the environment in which the navigation mode operates by using the list at the bottom of the toolbox.

Available Environments:

- Browser
- Linux
- Windows
- Mac

Tip

DO NOT enable the `Mouse and Keyboard` plugin in `Computer use` mode — it is already connected to `Computer use` mode in the background.

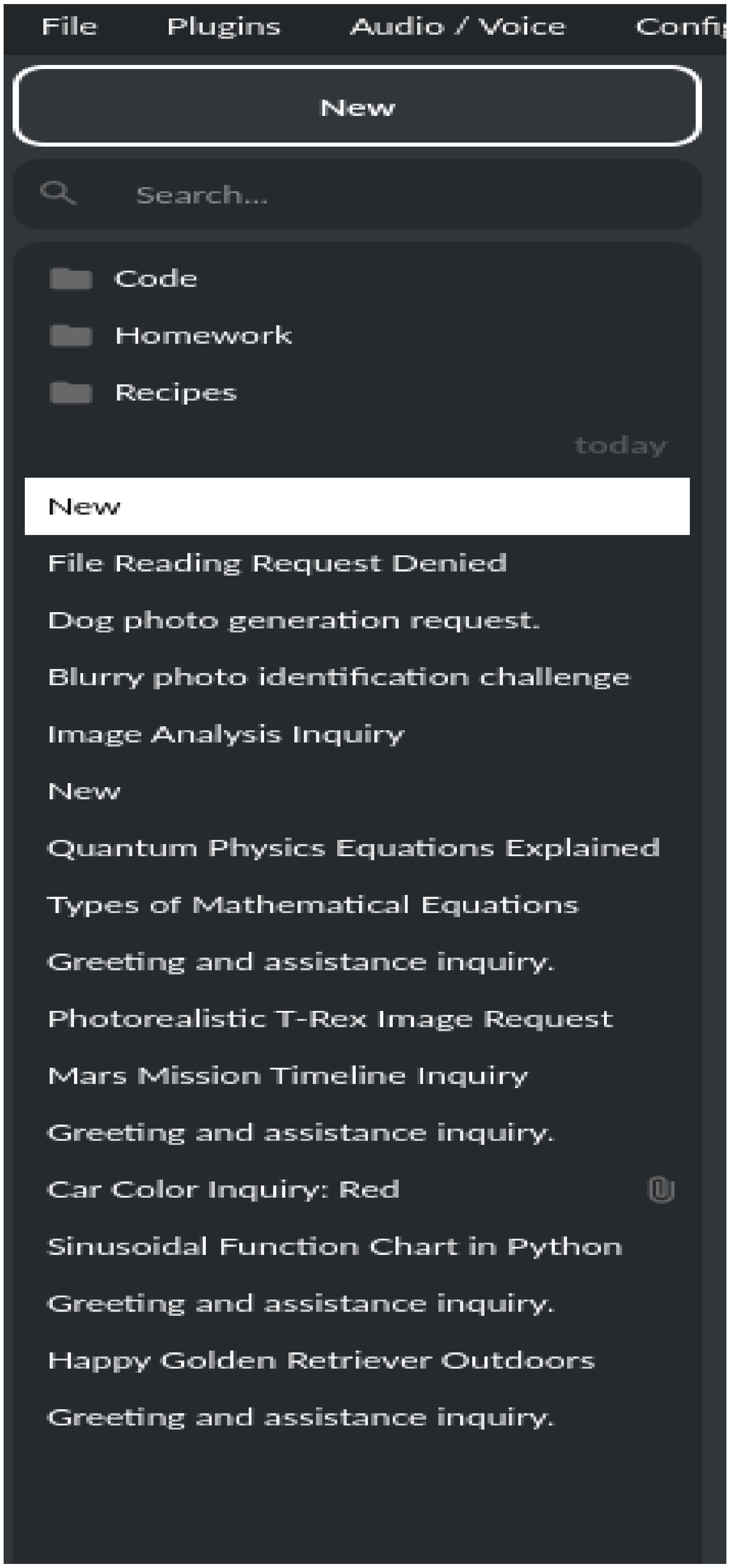
Context and memory

Short and long-term memory

PyGPT features a continuous chat mode that maintains a long context of the ongoing dialogue. It preserves the entire conversation history and automatically appends it to each new message (prompt) you send to the AI. Additionally, you have the flexibility to revisit past conversations whenever you choose. The application keeps a record of your chat history, allowing you to resume discussions from the exact point you stopped.

Handling multiple contexts

On the left side of the application interface, there is a panel that displays a list of saved conversations. You can save numerous contexts and switch between them with ease. This feature allows you to revisit and continue from any point in a previous conversation. **PyGPT** automatically generates a summary for each context, akin to the way `ChatGPT` operates and gives you the option to modify these titles itself.



Code
Homework
Recipes

today

New
File Reading Request Denied
Dog photo generation request.
Blurry photo identification challenge
Image Analysis Inquiry
New
Quantum Physics Equations Explained
Types of Mathematical Equations
Greeting and assistance inquiry.
Photorealistic T-Rex Image Request
Mars Mission Timeline Inquiry
Greeting and assistance inquiry.
Car Color Inquiry: Red
Sinusoidal Function Chart in Python
Greeting and assistance inquiry.
Happy Golden Retriever Outdoors
Greeting and assistance inquiry.

You can disable context support in the settings by using the following option:

```
Config -> Settings -> Use context
```

Clearing history

You can clear the entire memory (all contexts) by selecting the menu option:

```
File -> Clear history...
```

Context storage

On the application side, the context is stored in the `SQLite` database located in the working directory (`db.sqlite`). In addition, all history is also saved to `.txt` files for easy reading.

Files and Attachments

Uploading attachments

Using Your Own Files as Additional Context in Conversations

You can use your own files (for example, to analyze them) during any conversation. You can do this in two ways: by indexing (embedding) your files in a vector database, which makes them available all the time during a "Chat with Files" session, or by adding a file attachment (the attachment file will only be available during the conversation in which it was uploaded).

Attachments

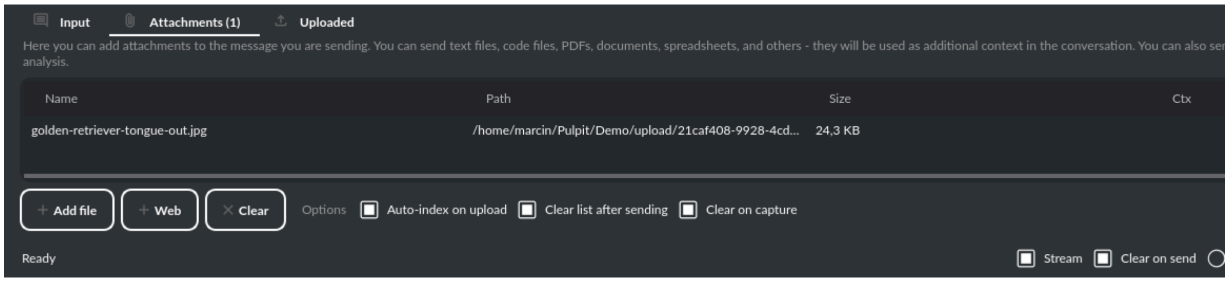
Warning

Important: When using `Full context` mode, the entire content of the file is included in the prompt, which can result in high token usage each time. If you want to reduce the number of tokens used, instead use the `RAG` option, which will only query the indexed attachment in the vector database to provide additional context.

PyGPT makes it simple for users to upload files and send them to the model for tasks like analysis, similar to attaching files in `ChatGPT`. There's a separate `Attachments` tab next to the text input area specifically for managing file uploads.

Tip

Attachments uploaded in group are available in all contexts in group.



Here you can add attachments to the message you are sending. You can send text files, code files, PDFs, documents, spreadsheets, and others - they will be used as additional context in the conversation. You can also se
analysis.

Name	Path	Size	Ctx
golden-retriever-tongue-out.jpg	/home/marcin/Pulpit/Demo/upload/21caf408-9928-4cd...	24,3 KB	

Options ☐ Auto-index on upload ☐ Clear list after sending ☐ Clear on capture

Ready ☐ Stream ☐ Clear on send

You can use attachments to provide additional context to the conversation. Uploaded files will be converted into text using loaders from LlamaIndex. You can upload any file format supported by the application through LlamaIndex. Supported formats include:

Text-based types:

- CSV files (csv)
- Epub files (epub)
- Excel .xlsx spreadsheets (xlsx)
- HTML files (html, htm)
- IPYNB Notebook files (ipynb)
- JSON files (json)
- Markdown files (md)
- PDF documents (pdf)
- Plain-text files (txt and etc.)
- Word .docx documents (docx)
- XML files (xml)

Media-types:

- Image (using vision) (jpg, jpeg, png, gif, bmp, tiff, webp)
- Video/audio (mp4, avi, mov, mkv, webm, mp3, mpeg, mpga, m4a, wav)

Archives:

- zip
- tar, tar.gz, tar.bz2

The content from the uploaded attachments will be used in the current conversation and will be available throughout (per context). There are 3 modes available for working with additional context from attachments:

- `Full context`: Provides best results. This mode attaches the entire content of the read file to the user's prompt. This process happens in the background and may require a large number of tokens if you uploaded extensive content.
- `RAG`: The indexed attachment will only be queried in real-time using LlamaIndex. This operation does not require any additional tokens, but it may not provide access to the full content of the file 1:1.
- `Summary`: When queried, an additional query will be generated in the background and executed by a separate model to summarize the content of the attachment and return the required information to the main model. You can change the model used for summarization in the settings under the `Files and attachments` section.

In the `RAG` and `Summary` mode, you can enable an additional setting by going to `Settings -> Files and attachments -> Use history in RAG query`. This allows for better preparation of queries for RAG. When this option is turned on, the entire conversation context is considered, rather than just the user's last query. This allows for better searching of the index for additional context. In the `RAG limit` option, you can set a limit on how many recent entries in a discussion should be considered (`0 = no limit, default: 3`).

Images as Additional Context

Files such as jpg, png, and similar images are a special case. By default, images are not used as additional context; they are analyzed in real-time using a vision model. If you want to use them as additional context instead, you must enable the "Allow images as additional context" option in the settings: `Files and attachments -> Allow images as additional context`.

Uploading larger files and auto-index

To use the `RAG` mode, the file must be indexed in the vector database. This occurs automatically at the time of upload if the `Auto-index on upload` option in the `Attachments` tab is enabled. When uploading large files, such indexing might take a while - therefore, if you are using the `Full context` option, which does not use the index, you can disable the `Auto-index` option to speed up the upload of the attachment. In this case, it will only be indexed when the `RAG` option is called for the first time, and until then, attachment will be available in the form of `Full context` and `Summary`.

Embeddings

When using RAG to query attachments, the documents are indexed into a temporary vector store. With multiple providers and models available, you can select the model used for querying attachments in: `Config -> Settings -> Files and Attachments`. You can also choose the embedding models for specified providers in `Config -> Settings -> Indexes / LlamaIndex -> Embeddings -> Default embedding models` list. By default, when querying an attachment using RAG, the default embedding model and provider corresponding to the RAG query model will be used. If no default configuration is provided for a specific provider, the global embedding configuration will be used.

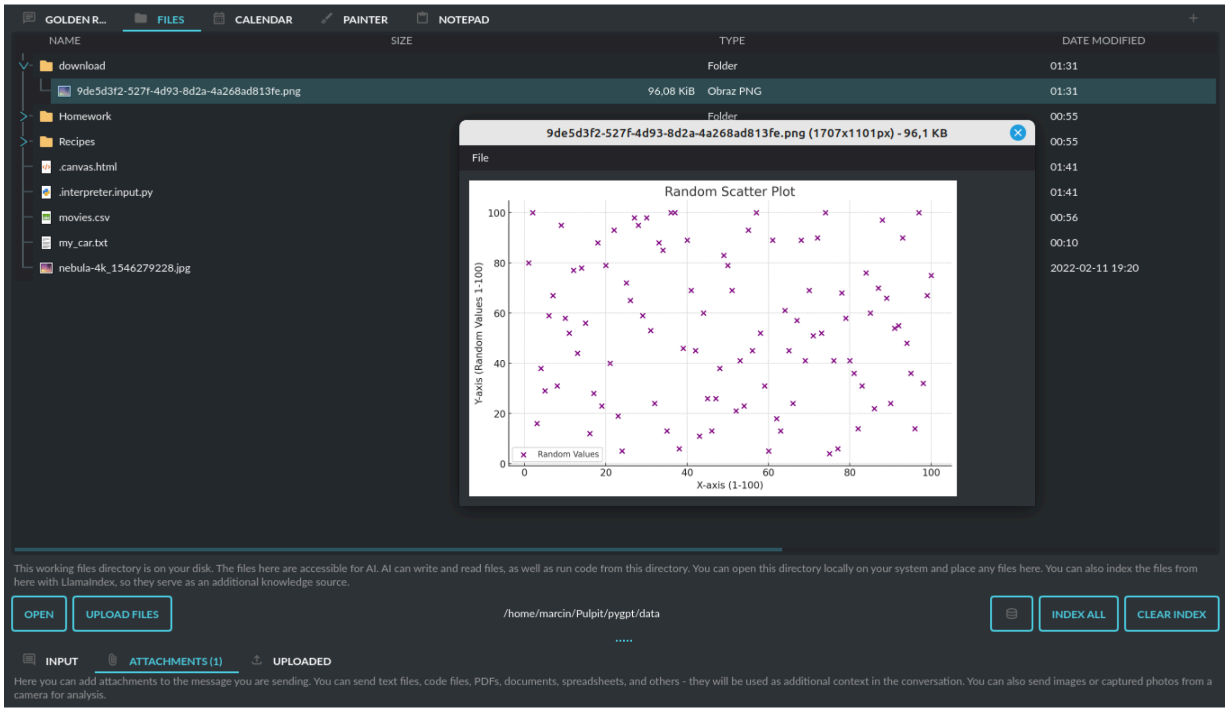
For example, if the RAG query model is `gpt-4o-mini`, then the default model for the provider `OpenAI` will be used. If the default model for `OpenAI` is not specified on the list, the global provider and model will be used.

Downloading files

PyGPT enables the automatic download and saving of files created by the model. This is carried out in the background, with the files being saved to an `data` folder located within the user's working directory. To view or manage these files, users can navigate to the `Files` tab which features a file browser for this specific directory. Here, users have the interface to handle all files sent by the AI.

This `data` directory is also where the application stores files that are generated locally by the AI, such as code files or any other outputs requested from the model. Users have the option to execute code directly from the stored files and read their contents, with the results fed back to the AI. This hands-off process is managed by the built-in plugin system and model-triggered commands. You can also indexing files from this directory (using integrated `LlamaIndex`) and use it's contents as additional context provided to discussion.

The `Files I/O` plugin takes care of file operations in the `data` directory, while the `Code Interpreter` plugin allows for the execution of code from these files.



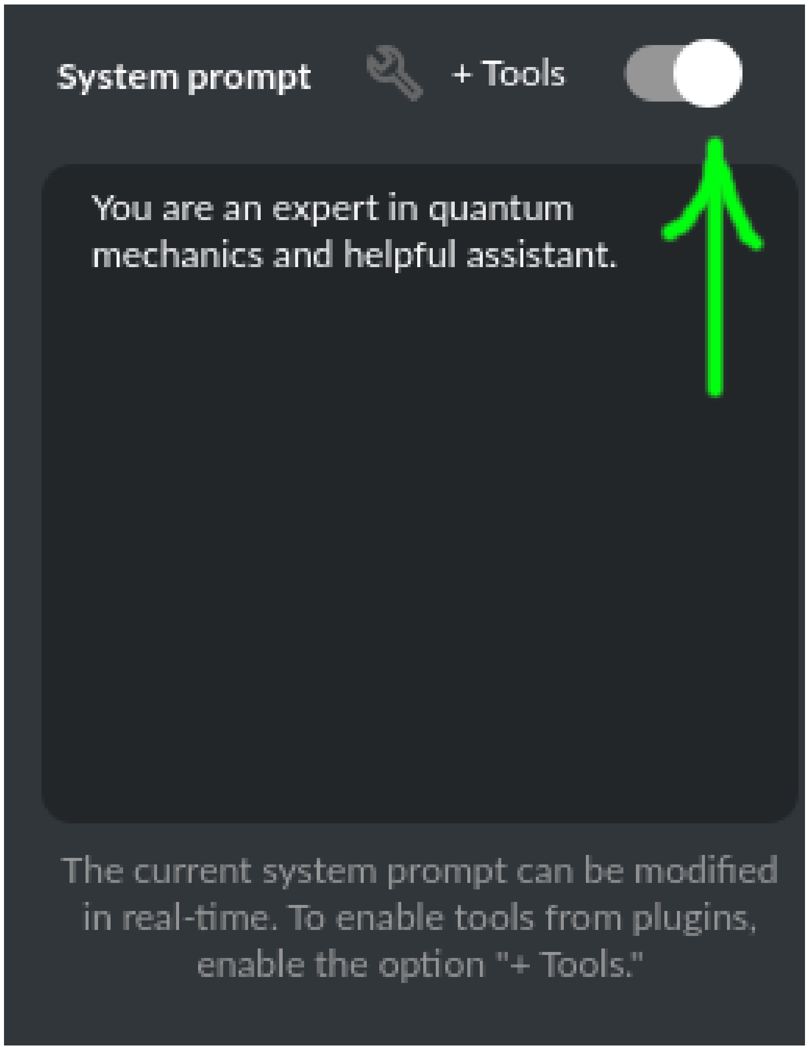
NAME	SIZE	TYPE	DATE MODIFIED
download		Folder	01:31
9de5d3f2-527f-4d93-8d2a-4a268ad813fe.png	96,08 KiB	Obraz PNG	01:31
Homework		Folder	00:55
Recipes			00:55
.canvas.html			01:41
.interpreter.input.py			01:41
movies.csv			00:56
my_car.txt			00:10
nebula-4k_1546279228.jpg			2022-02-11 19:20

This working files directory is on your disk. The files here are accessible for AI. AI can write and read files, as well as run code from this directory. You can open this directory locally on your system and place any files here. You can also index the files from here with LlamaIndex, so they serve as an additional knowledge source.

/home/marcin/Pulpit/pygpt/data

Here you can add attachments to the message you are sending. You can send text files, code files, PDFs, documents, spreadsheets, and others - they will be used as additional context in the conversation. You can also send images or captured photos from a camera for analysis.

To allow the model to manage files or python code execution, the `+ Tools` option must be active, along with the above-mentioned plugins:


System prompt

+ Tools

You are an expert in quantum mechanics and helpful assistant.

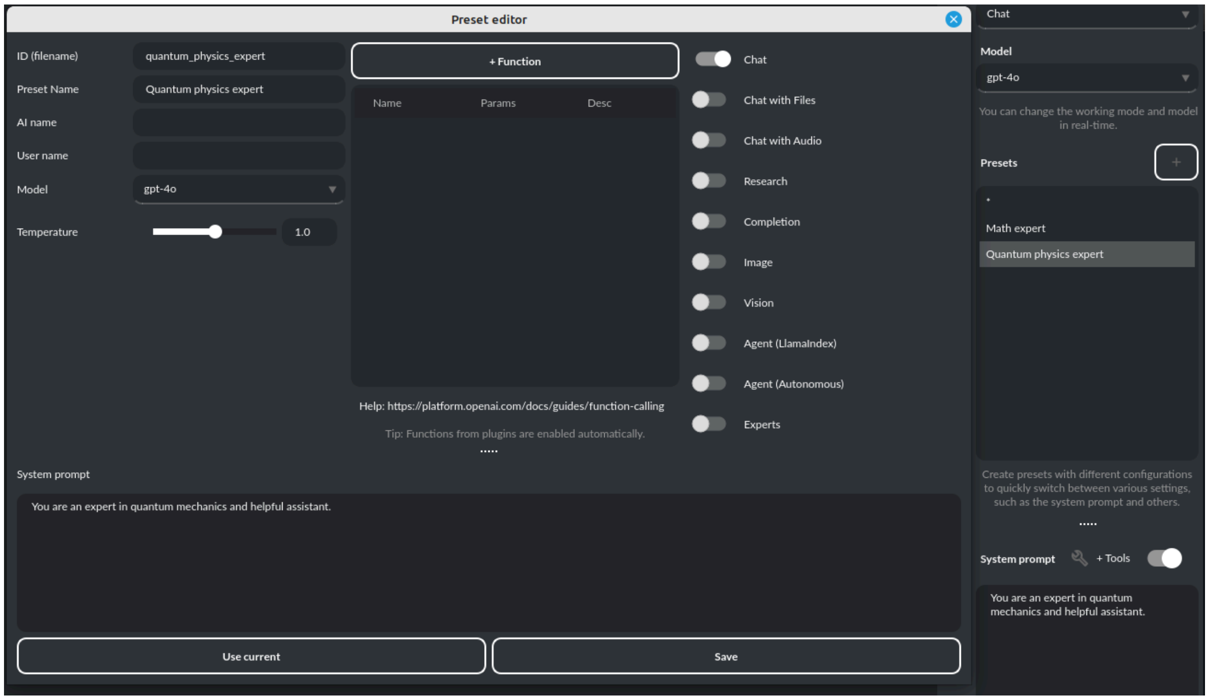
The current system prompt can be modified in real-time. To enable tools from plugins, enable the option "+ Tools."

Presets

What is preset?

Presets in **PyGPT** are essentially templates used to store and quickly apply different configurations. Each preset includes settings for the mode you want to use (such as chat, completion, or image generation), an initial system prompt, an assigned name for the AI, a username for the session, and the desired "temperature" for the conversation. A warmer "temperature" setting allows the AI to provide more creative responses, while a cooler setting encourages more predictable replies. These presets can be used across various modes and with models accessed via the `OpenAI API` or `LlamaIndex`.

The application lets you create as many presets as needed and easily switch among them. Additionally, you can clone an existing preset, which is useful for creating variations based on previously set configurations and experimentation.



You are an expert in quantum mechanics and helpful assistant.

Example usage

The application includes several sample presets that help you become acquainted with the mechanism of their use.

Profiles

You can create multiple profiles for an app and switch between them. Each profile uses its own configuration, settings, context history, and a separate folder for user files. This allows you to set up different environments and quickly switch between them, changing the entire setup with just one click.

The app lets you create new profiles, edit existing ones, and duplicate current ones.

To create a new profile, select the option from the menu: `Config -> Profile -> New Profile...`

To edit saved profiles, choose the option from the menu: `Config -> Profile -> Edit Profiles...`

To switch to a created profile, pick the profile from the menu: `Config -> Profile -> [Profile Name]`

Each profile uses its own user directory (workdir). You can link a newly created or edited profile to an existing workdir with its configuration.

The name of the currently active profile is shown as (Profile Name) in the window title.

Models

Built-in models

PyGPT has a preconfigured list of models (as of 2025-08-31):

- `bielik-11b-v2.3-instruct:Q4_K_M` (Ollama)
- `chatgpt-4o-latest` (OpenAI)
- `claude-3-5-sonnet-20240620` (Anthropic)
- `claude-3-7-sonnet` (Anthropic)
- `claude-3-opus` (Anthropic)
- `claude-3-opus` (Anthropic)
- `claude-opus-4-0` (Anthropic)
- `claude-sonnet-4-0` (Anthropic)
- `codellama` (Ollama)
- `codex-mini` (OpenAI)
- `dall-e-2` (OpenAI)
- `dall-e-3` (OpenAI)
- `deepseek-chat` (DeepSeek)
- `deepseek-r1:1.5b` (Ollama)
- `deepseek-r1:14b` (Ollama)
- `deepseek-r1:32b` (Ollama)
- `deepseek-r1:7b` (Ollama)
- `deepseek-reasoner` (DeepSeek)
- `gemini-1.5-flash` (Google)
- `gemini-1.5-pro` (Google)
- `gemini-2.0-flash-exp` (Google)
- `gemini-2.5-flash` (Google)
- `gemini-2.5-flash-preview-native-audio-dialog` (Google, real-time)
- `gemini-2.5-pro` (Google)
- `gpt-3.5-turbo` (OpenAI)

- `gpt-3.5-turbo-16k` (OpenAI)
- `gpt-3.5-turbo-instruct` (OpenAI)
- `gpt-4` (OpenAI)
- `gpt-4-32k` (OpenAI)
- `gpt-4-turbo` (OpenAI)
- `gpt-4-vision-preview` (OpenAI)
- `gpt-4.1` (OpenAI)
- `gpt-4.1-mini` (OpenAI)
- `gpt-4.1-nano` (OpenAI)
- `gpt-4o` (OpenAI)
- `gpt-4o-realtime-preview` (OpenAI, real-time)
- `gpt-4o-mini` (OpenAI)
- `gpt-5` (OpenAI)
- `gpt-5-mini` (OpenAI)
- `gpt-5-nano` (OpenAI)
- `gpt-image-1` (OpenAI)
- `gpt-oss:20b` (OpenAI - via Ollama and HuggingFace Router)
- `gpt-oss:120b` (OpenAI - via Ollama and HuggingFace Router)
- `gpt-realtime` (OpenAI, real-time)
- `grok-2-vision` (xAI)
- `grok-3` (xAI)
- `grok-3-fast` (xAI)
- `grok-3-mini` (xAI)
- `grok-3-mini-fast` (xAI)
- `grok-4` (xAI)
- `llama2-uncensored` (Ollama)
- `llama3.1` (Ollama)
- `llama3.1:70b` (Ollama)
- `mistral` (Ollama)
- `mistral-large` (Ollama)
- `mistral-small3.1` (Ollama)
- `o1` (OpenAI)
- `o1-mini` (OpenAI)
- `o1-pro` (OpenAI)

- `o3` (OpenAI)
- `o3-deep-research` (OpenAI)
- `o3-mini` (OpenAI)
- `o3-pro` (OpenAI)
- `o4-mini` (OpenAI)
- `o4-mini-deep-research` (OpenAI)
- `qwen2:7b` (Ollama)
- `qwen2.5-coder:7b` (Ollama)
- `qwen3:8b` (Ollama)
- `qwen3:30b-a3b` (Ollama)
- `r1` (Perplexity)
- `sonar` (Perplexity)
- `sonar-deep-research` (Perplexity)
- `sonar-pro` (Perplexity)
- `sonar-reasoning` (Perplexity)
- `sonar-reasoning-pro` (Perplexity)
- `veo-3.0-generate-preview` (Google)
- `veo-3.0-fast-generate-preview` (Google)

All models are specified in the configuration file `models.json`, which you can customize. This file is located in your working directory. You can add new models provided directly by `OpenAI API` (or compatible) and those supported by `LlamaIndex` or `Ollama` to this file. Configuration for LlamaIndex in placed in `llama_index` key.

You can import new models by manually editing `models.json` or by using the model importer in the `Config -> Models -> Import` menu.

Tip

The models on the list are sorted by provider, not by manufacturer. A model from a particular manufacturer may be available through different providers (e.g., OpenAI models can be provided by the `OpenAI API` or by `OpenRouter`). If you want to use a specific model through a particular provider, you need to configure the provider in

`Config -> Models -> Edit`, or import it directly via `Config -> Models -> Import`.

Tip

Anthropic and Deepseek API providers use VoyageAI for embeddings (Chat with Files and attachments RAG), so you must also configure the Voyage API key if you want to use embeddings from these providers.

Adding a custom model

You can add your own models. See the section `Extending PyGPT / Adding a new model` for more info.

There is built-in support for those LLM providers:

- `Anthropic`
- `Azure OpenAI` (native SDK)
- `Deepseek API`
- `Google` (native SDK)
- `HuggingFace API`
- `HuggingFace Router` (wrapper for OpenAI compatible ChatCompletions)
- `Local models` (OpenAI API compatible)
- `Mistral AI`
- `Ollama`
- `OpenAI` (native SDK)
- `OpenRouter`
- `Perplexity`
- `xAI` (native SDK)

How to use local or non-GPT models

Llama 3, Mistral, DeepSeek, Qwen, gpt-oss, and other local models

How to use locally installed Llama 3, DeepSeek, Mistral, etc. models:

1. Choose a working mode: `Chat` or `Chat with Files`.
2. On the models list - select, edit, or add a new model (with `ollama` provider). You can edit the model settings through the menu `Config -> Models -> Edit`, then configure the model parameters in the `advanced` section.
3. Download and install Ollama from here: https://github.com/ollama/ollama

For example, on Linux:

```
$ curl -fsSL https://ollama.com/install.sh | sh
```

4. Run the model (e.g. Llama 3) locally on your machine. For example, on Linux:

```
$ ollama run llama3.1
```

5. Return to PyGPT and select the correct model from models list to chat with selected model using Ollama running locally.

Example available models:

- `llama3.1`
- `codellama`
- `mistral`
- `llama2-uncensored`
- `deepseek-r1`

etc.

You can add more models by editing the models list.

Real-time importer

You can also import models in real-time from a running Ollama instance using the `Config -> Models -> Import...` tool.

Custom Ollama endpoint

The default endpoint for Ollama is: http://localhost:11434

You can change it globally by setting the environment variable `OLLAMA_API_BASE` in `Settings -> General -> Advanced -> Application environment`.

You can also change the "base_url" for a specific model in its configuration:

`Config -> Models -> Edit`, then in the `Advanced -> [LlamaIndex] ENV Vars` section add the variable:

NAME: `OLLAMA_API_BASE` VALUE: `http://my_endpoint.com:11434`

List of all models supported by Ollama:

https://ollama.com/library

https://github.com/ollama/ollama

IMPORTANT: Remember to define the correct model name in the **kwargs list in the model settings.

Using local embeddings

Refer to: https://docs.llamaindex.ai/en/stable/examples/embeddings/ollama_embedding/

You can use an Ollama instance for embeddings. Simply select the `ollama` provider in:

```
Config -> Settings -> Indexes / LlamaIndex -> Embeddings ->
Embeddings provider
```

Define parameters like model name and Ollama base URL in the Embeddings provider **kwargs list, e.g.:

- name: `model_name`, value: `llama3.1`, type: `str`
- name: `base_url`, value: `http://localhost:11434`, type: `str`

Google Gemini, Anthropic Claude, xAI Grok, etc.

If you want to use non-OpenAI models in `Chat with Files` and `Agents (LlamaIndex)` modes, then remember to configure the required parameters like API keys in the model config fields. `Chat` mode works via OpenAI SDK (compatible API), `Chat with Files` and `Agents (LlamaIndex)` modes works via LlamaIndex.

Google Gemini

Required ENV:

- GOOGLE_API_KEY = {api_key_google}

Required **kwargs:

- model

Anthropic Claude

Required ENV:

- ANTHROPIC_API_KEY = {api_key_anthropic}

Required **kwargs:

- model

xAI Grok (Chat mode only)

Required ENV:

- OPENAI_API_KEY = {api_key_xai}
- OPENAI_API_BASE = {api_endpoint_xai}

Required **kwargs:

- model

Mistral AI

Required ENV:

- MISTRAL_API_KEY = {api_key_mistral}

Required **kwargs:

- model

Perplexity

Required ENV:

- PPLX_API_KEY = {api_key_perplexity}

Required **kwargs:

- model

HuggingFace API (Chat with Files mode only)

Required ENV:

- HUGGING_FACE_TOKEN = {api_key_hugging_face}

Required **kwargs:

- model_name | model
- token
- provider = auto

Plugins

Overview

PyGPT can be enhanced with plugins to add new features.

The following plugins are currently available, and model can use them instantly:

- `API calls` - plugin lets you connect the model to the external services using custom defined API calls.
- `Audio Input` - provides speech recognition.
- `Audio Output` - provides voice synthesis.
- `Autonomous Agent (inline)` - enables autonomous conversation (AI to AI), manages loop, and connects output back to input. This is the inline Agent mode.
- `Bitbucket` - Access Bitbucket API to manage repositories, issues, and pull requests.
- `Chat with files (LlamaIndex, inline)` - plugin integrates LlamaIndex storage in any chat and provides additional knowledge into context (from indexed files).
- `Code Interpreter` - responsible for generating and executing Python code, functioning much like the *Code Interpreter* on *ChatGPT*, but locally. This means model can interface with any script, application, or code. Plugins can work in conjunction to perform sequential tasks; for example, the *Files* plugin can write generated Python code to a file, which the *Code Interpreter* can execute it and return its result to model.
- `Context history (calendar, inline)` - provides access to context history database.
- `Crontab / Task scheduler` - plugin provides cron-based job scheduling - you can schedule tasks/prompts to be sent at any

time using cron-based syntax for task setup.

- `Custom Commands` - allows you to create and execute custom commands on your system.
- `Experts (inline)` - allows calling experts in any chat mode. This is the inline Experts (co-op) mode.
- `Facebook` - Manage user info, pages, posts, and photos on Facebook pages.
- `Files I/O` - grants access to the local filesystem, enabling model to read and write files, as well as list and create directories.
- `GitHub` - Access GitHub API to manage repositories, issues, and pull requests.
- `Google` - Access Gmail, Drive, Docs, Maps, Calendar, Contacts, Colab, YouTube, Keep - for managing emails, files, events, notes, video info, and contacts.
- `Image Generation (inline)` - integrates DALL-E 3 image generation with any chat and mode. Just enable and ask for image in Chat mode, using standard model like GPT-4. The plugin does not require the `+ Tools` option to be enabled.
- `Mailer` - Provides the ability to send, receive and read emails.
- `MCP` - Provides access to remote tools via the Model Context Protocol (MCP), including stdio, SSE, and Streamable HTTP transports, with per-server allow/deny filtering, Authorization header support, and a tools cache.
- `Mouse and Keyboard` - provides the ability to control the mouse and keyboard by the model.
- `OpenStreetMap` - Search, geocode, plan routes, and generate static maps using OpenStreetMap services (Nominatim, OSRM, staticmap).
- `Real Time` - automatically appends the current date and time to the system prompt, informing the model about current time.
- `Serial port / USB` - plugin provides commands for reading and sending data to USB ports.
- `Server (SSH/FTP)` - Connect to remote servers using FTP, SFTP, and SSH. Execute remote commands, upload, download, and more.

- `Slack` - Handle users, conversations, messages, and files on Slack.
- `System (OS)` - provides access to the operating system and executes system commands.
- `System Prompt Extra` - appends additional system prompts (extra data) from a list to every current system prompt. You can enhance every system prompt with extra instructions that will be automatically appended to the system prompt.
- `Telegram` - Send messages, photos, and documents; manage chats and contacts.
- `Tuya (IoT)` - Handle Tuya Smart Home devices via Tuya Cloud API.
- `Vision (inline)` - integrates vision capabilities with any chat mode, not just Vision mode. When the plugin is enabled, the model temporarily switches to vision in the background when an image attachment or vision capture is provided.
- `Voice Control (inline)` - provides voice control command execution within a conversation.
- `Web Search` - provides the ability to connect to the Web, search web pages for current data, and index external content using LlamaIndex data loaders.
- `Wikipedia` - Search Wikipedia for information.
- `Wolfram Alpha` - Compute and solve with Wolfram Alpha: short answers, full JSON pods, math (solve, derivatives, integrals), unit conversions, matrix operations, and plots.
- `X/Twitter` - Interact with tweets and users, manage bookmarks and media, perform likes, retweets, and more.

Creating Your Own Plugins

You can create your own plugin for **PyGPT** at any time. The plugin can be written in Python and then registered with the application just before launching it. All plugins included with the app are stored in the `plugin` directory - you can use them as coding examples for your own plugins.

PyGPT can be extended with:

- custom plugins
- custom LLMs
- custom vector store providers
- custom data loaders
- custom audio input providers
- custom audio output providers
- custom web search engine providers
- custom agents (LlamaIndex or OpenAI Agents)

See the section `Extending PyGPT / Adding a custom plugin` for more details.

API calls

PyGPT lets you connect the model to the external services using custom defined API calls.

To activate this feature, turn on the `API calls` plugin found in the `Plugins` menu.

In this plugin you can provide list of allowed API calls, their parameters and request types. The model will replace provided placeholders with required params and make API call to external service.

- `Your custom API calls` *cmds*

You can provide custom API calls on the list here.

Params to specify for API call:

- **Enabled** (True / False)
- **Name:** unique API call name (ID)
- **Instruction:** description for model when and how to use this API call
- **GET params:** list, separated by comma, GET params to append to endpoint URL
- **POST params:** list, separated by comma, POST params to send in POST request
- **POST JSON:** provide the JSON object, template to send in POST JSON request, use `%param%` as POST param placeholders
- **Headers:** provide the JSON object with dictionary of extra request headers, like Authorization, API keys, etc.
- **Request type:** use GET for basic GET request, POST to send encoded POST params or POST_JSON to send JSON-encoded object as body
- **Endpoint:** API endpoint URL, use `{param}` as GET param placeholders

An example API call is provided with plugin by default, it calls the Wikipedia API:

- Name: `search_wiki`
- Instructiom: `send API call to Wikipedia to search pages by query`
- GET params: `query, limit`
- Type: `GET`
- API endpoint: https://en.wikipedia.org/w/api.php?action=opensearch&limit={limit}&format=json&search={query}

In the above example, every time you ask the model for query Wiki for provided query (e.g. `Call the Wikipedia API for query: Nikola Tesla`) it will replace placeholders in provided API endpoint URL with a generated query and it will call prepared API endpoint URL, like below:

https://en.wikipedia.org/w/api.php?action=opensearch&limit=5&format=json&search=Nikola%20Tesla

You can specify type of request: `GET`, `POST` and `POST JSON`.

In the `POST` request you can provide POST params, they will be encoded and send as POST data.

In the `POST JSON` request you must provide JSON object template to be send, using `%param%` placeholders in the JSON object to be replaced with the model.

You can also provide any required credentials, like Authorization headers, API keys, tokens, etc. using the `headers` field - you can provide a JSON object here with a dictionary `key => value` - provided JSON object will be converted to headers dictonary and send with the request.

- `Disable SSL verify` *disable_ssl*

Disables SSL verification when making requests. *Default: False*

- `Timeout` *timeout*

Connection timeout (seconds). *Default: 5*

- `User agent` *user_agent*

User agent to use when making requests, default: `Mozilla/5.0`. *Default: Mozilla/5.0*

Audio Input

The plugin facilitates speech recognition (by default using the `Whisper` model from OpenAI, `Google` and `Bing` are also available). It allows for voice commands to be relayed to the AI using your own voice. Whisper doesn’t require any extra API keys or additional configurations; it uses the main OpenAI key. In the plugin’s configuration options, you should adjust the volume level (min energy) at which the plugin will respond to your microphone. Once the plugin is activated, a new `Speak` option will appear at the bottom near the `Send` button - when this is enabled, the application will respond to the voice received from the microphone.

The plugin can be extended with other speech recognition providers.

Options

- `Provider` *provider*

Choose the provider. *Default: Whisper*

Available providers:

- Whisper (via `OpenAI API`)
- Whisper (local model) - not available in compiled and Snap versions, only Python/PyPi version
- Google (via `SpeechRecognition` library)
- Google Cloud (via `SpeechRecognition` library)
- Microsoft Bing (via `SpeechRecognition` library)

Whisper (API)

- `Model` *whisper_model*

Choose the model. *Default: whisper-1*

Whisper (local)

- `Model` *whisper_local_model*

Choose the local model. *Default: base*

Available models: https://github.com/openai/whisper

Google

- `Additional keywords arguments` *google_args*

Additional keywords arguments for r.recognize_google(audio, **kwargs)

Google Cloud

- `Additional keywords arguments` *google_args*

Additional keywords arguments for r.recognize_google_cloud(audio, **kwargs)

Bing

- `Additional keywords arguments` *bing_args*

Additional keywords arguments for r.recognize_bing(audio, **kwargs)

General options

- `Auto send` *auto_send*

Automatically send recognized speech as input text after recognition. *Default: True*

- `Advanced mode` *advanced*

Enable only if you want to use advanced mode and the settings below. Do not enable this option if you just want to use the simplified mode (default). *Default: False*

Advanced mode options

- `Timeout` *timeout*

The duration in seconds that the application waits for voice input from the microphone. *Default: 5*

- `Phrase max length` *phrase_length*

Maximum duration for a voice sample (in seconds). *Default: 10*

- `Min energy` *min_energy*

Minimum threshold multiplier above the noise level to begin recording. *Default: 1.3*

- `Adjust for ambient noise` *adjust_noise*

Enables adjustment to ambient noise levels. *Default: True*

- `Continuous listen` *continuous_listen*

Experimental: continuous listening - do not stop listening after a single input. Warning: This feature may lead to unexpected results and requires fine-tuning with the rest of the options! If disabled, listening must be started manually by enabling the `Speak` option. *Default: False*

- `Wait for response` *wait_response*

Wait for a response before initiating listening for the next input. *Default: True*

- `Magic word` *magic_word*

Activate listening only after the magic word is provided. *Default: False*

- `Reset Magic word` *magic_word_reset*

Reset the magic word status after it is received (the magic word will need to be provided again). *Default: True*

- `Magic words` *magic_words*

List of magic words to initiate listening (Magic word mode must be enabled). *Default: OK, Okay, Hey GPT, OK GPT*

- `Magic word timeout` *magic_word_timeout*

he number of seconds the application waits for magic word. *Default: 1*

- `Magic word phrase max length` *magic_word_phrase_length*

The minimum phrase duration for magic word. *Default: 2*

- `Prefix words` *prefix_words*

List of words that must initiate each phrase to be processed. For example, you can define words like “OK” or “GPT”—if set, any phrases not starting with those words will be ignored. Insert multiple words or phrases separated by commas. Leave empty to deactivate. *Default: empty*

- `Stop words` *stop_words*

List of words that will stop the listening process. *Default: stop, exit, quit, end, finish, close, terminate, kill, halt, abort*

Options related to Speech Recognition internals:

- `energy_threshold` *recognition_energy_threshold*

Represents the energy level threshold for sounds. *Default: 300*

- `dynamic_energy_threshold` *recognition_dynamic_energy_threshold*

Represents whether the energy level threshold (see recognizer_instance.energy_threshold) for sounds should be automatically adjusted based on the currently ambient noise level while listening. *Default: True*

- `dynamic_energy_adjustment_damping` *recognition_dynamic_energy_adjustment_damping*

Represents approximately the fraction of the current energy threshold that is retained after one second of dynamic threshold adjustment. *Default: 0.15*

- `pause_threshold` *recognition_pause_threshold*

Represents the minimum length of silence (in seconds) that will register as the end of a phrase. *Default: 0.8*

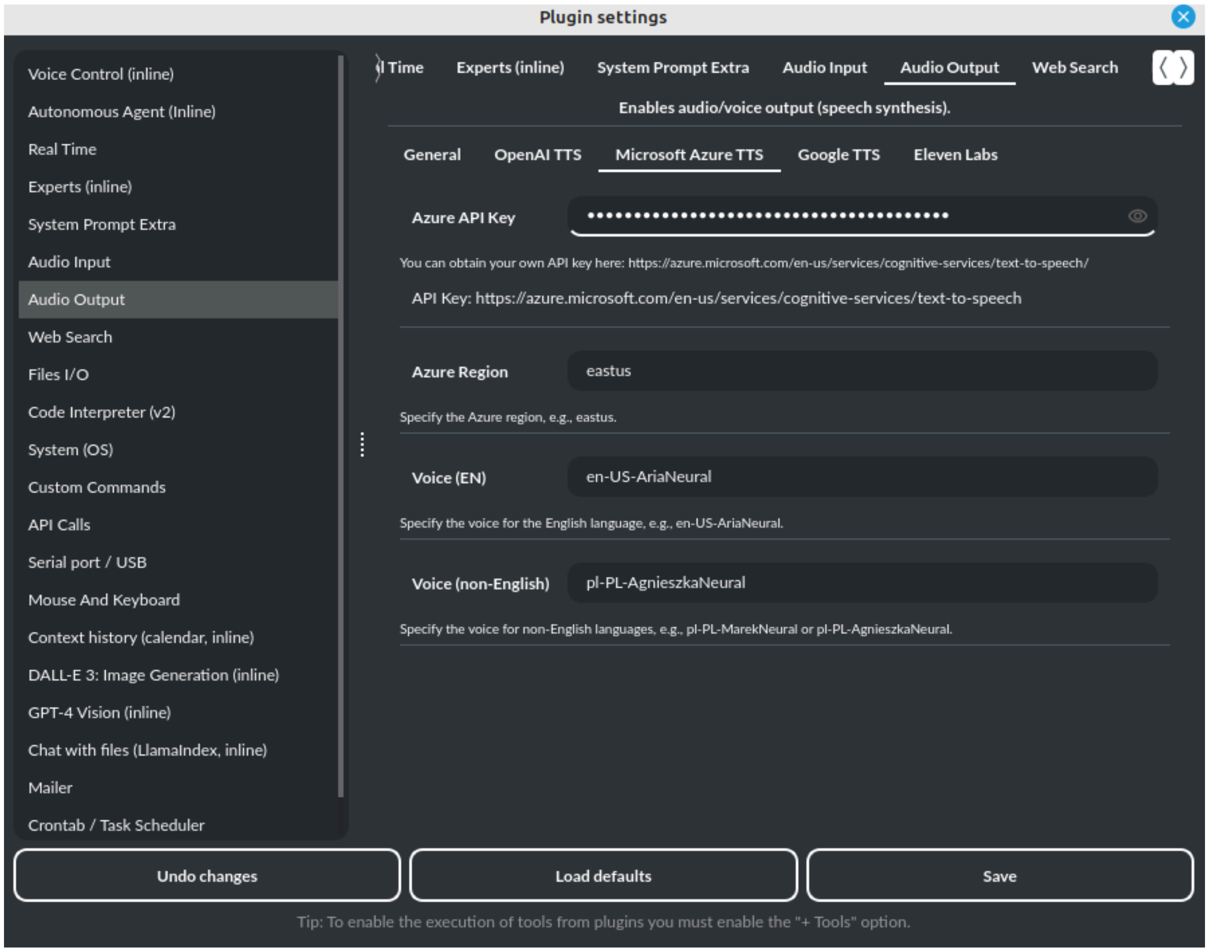
- `adjust_for_ambient_noise: duration` *recognition_adjust_for_ambient_noise_duration*

The duration parameter is the maximum number of seconds that it will dynamically adjust the threshold for before returning. *Default: 1*

Options reference:
https://pypi.org/project/SpeechRecognition/1.3.1/

Audio Output

The plugin lets you turn text into speech using the TTS model from OpenAI or other services like `Microsoft Azure`, `Google`, and `Eleven Labs`. You can add more text-to-speech providers to it too. `OpenAI TTS` does not require any additional API keys or extra configuration; it utilizes the main OpenAI key. Microsoft Azure requires to have an Azure API Key. Before using speech synthesis via `Microsoft Azure`, `Google` or `Eleven Labs`, you must configure the audio plugin with your API keys, regions and voices if required.



Enables audio/voice output (speech synthesis).

Azure API Key: ••••••••••••••••••••••••••••••••••••••

You can obtain your own API key here: https://azure.microsoft.com/en-us/services/cognitive-services/text-to-speech/

API Key: https://azure.microsoft.com/en-us/services/cognitive-services/text-to-speech

Azure Region: eastus

Specify the Azure region, e.g., eastus.

Voice (EN): en-US-AriaNeural

Specify the voice for the English language, e.g., en-US-AriaNeural.

Voice (non-English): pl-PL-AgnieszkaNeural

Specify the voice for non-English languages, e.g., pl-PL-MarekNeural or pl-PL-AgnieszkaNeural.

Tip: To enable the execution of tools from plugins you must enable the "+ Tools" option.

Through the available options, you can select the voice that you want the model to use. More voice synthesis providers coming soon.

To enable voice synthesis, activate the `Audio Output` plugin in the `Plugins` menu or turn on the `Audio Output` option in the `Audio / Voice` menu (both options in the menu achieve the same outcome).

Options

- `Provider` *provider*

Choose the provider. *Default: OpenAI TTS*

Available providers:

- OpenAI TTS
- Microsoft Azure TTS
- Google TTS
- Eleven Labs TTS

OpenAI Text-To-Speech

- `Model` *openai_model*

Choose the model. Available options:

- tts-1
- tts-1-hd

Default: tts-1

- *Voice openai_voice*

Choose the voice. Available voices to choose from:

- alloy
- echo
- fable
- onyx
- nova
- shimmer

Default: alloy

Microsoft Azure Text-To-Speech

- `Azure API Key` *azure_api_key*

Here, you should enter the API key, which can be obtained by registering for free on the following website:

https://azure.microsoft.com/en-us/services/cognitive-services/text-to-speech

- `Azure Region` *azure_region*

You must also provide the appropriate region for Azure here. *Default: eastus*

- `Voice (EN)` *azure_voice_en*

Here you can specify the name of the voice used for speech synthesis for English. *Default: en-US-AriaNeural*

- `Voice (non-English)` *azure_voice_pl*

Here you can specify the name of the voice used for speech synthesis for other non-english languages. *Default: pl-PL-AgnieszkaNeural*

Google Text-To-Speech

- `Google Cloud Text-to-speech API Key` *google_api_key*

You can obtain your own API key at: https://console.cloud.google.com/apis/library/texttospeech.googleapis.com

- `Voice` *google_voice*

Specify voice. Voices: https://cloud.google.com/text-to-speech/docs/voices

- `Language code` *google_api_key*

Language code. Language codes: https://cloud.google.com/speech-to-text/docs/speech-to-text-supported-languages

Eleven Labs Text-To-Speech

- `Eleven Labs API Key` *eleven_labs_api_key*

You can obtain your own API key at: https://elevenlabs.io/speech-synthesis

- `Voice ID` *eleven_labs_voice*

Voice ID. Voices: https://elevenlabs.io/voice-library

- `Model` *eleven_labs_model*

Specify model. Models: https://elevenlabs.io/docs/speech-synthesis/models

If speech synthesis is enabled, a voice will be additionally generated in the background while generating a response via model.

Both `OpenAI TTS` and `OpenAI Whisper` use the same single API key provided for the OpenAI API, with no additional keys required.

Autonomous Agent (inline)

Warning

Please use autonomous mode with caution! - this mode, when connected with other plugins, may produce unexpected results!

The plugin activates autonomous mode in standard chat modes, where AI begins a conversation with itself. You can set this loop to run for any number of iterations. Throughout this sequence, the model will engage in self-dialogue, answering his own questions and comments, in order to find the best possible solution, subjecting previously generated steps to criticism.

This mode is similar to `Auto-GPT` - it can be used to create more advanced inferences and to solve problems by breaking them down into subtasks that the model will autonomously perform one after another until the goal is achieved. The plugin is capable of working in cooperation with other plugins, thus it can utilize tools such as web search, access to the file system, or image generation using `DALL-E`.

Options

You can adjust the number of iterations for the self-conversation in the `Plugins / Settings...` menu under the following option:

- `Iterations` *iterations*

Default: 3

Warning

Setting this option to `0` activates an **infinity loop** which can generate a large number of requests and cause very high token consumption, so use this option with caution!

- `Prompts` *prompts*

Editable list of prompts used to instruct how to handle autonomous mode, you can create as many prompts as you want. First active prompt on list will be used to handle autonomous mode.

- `Auto-stop after goal is reached` *auto_stop*

If enabled, plugin will stop after goal is reached. *Default: True*

- `Reverse roles between iterations` *reverse_roles*

Only for Completion mode. If enabled, this option reverses the roles (AI <> user) with each iteration. For example, if in the previous iteration the response was generated for "Batman," the next iteration will use that response to generate an input for "Joker." *Default: True*

Bitbucket

The Bitbucket plugin allows for seamless integration with the Bitbucket Cloud API, offering functionalities to manage repositories, issues, and pull requests. This plugin provides highly configurable options for authentication, cached convenience, and manages HTTP requests efficiently.

- Retrieve details about the authenticated user.
- Get information about a specific user.
- List available workspaces.
- List repositories in a workspace.
- Get details about a specific repository.
- Create a new repository.
- Delete an existing repository.
- Retrieve contents of a file in a repository.
- Upload a file to a repository.
- Delete a file from a repository.
- List issues in a repository.
- Create a new issue.
- Comment on an existing issue.
- Update details of an issue.
- List pull requests in a repository.
- Create a new pull request.
- Merge an existing pull request.
- Search for repositories.

Options

- `API base` *api_base*

Define the base URL for the Bitbucket Cloud API. *Default: https://api.bitbucket.org/2.0*

- `HTTP timeout (s)` *http_timeout*

  Set the timeout for HTTP requests in seconds. *Default: 30*

Auth options

- `Auth mode` *auth_mode*

  Select the authentication mode. *Default: auto*

  Available modes: * auto * basic * bearer

- `Username` *bb_username*

  Provide your Bitbucket username (handle, not email).

- `App Password` *bb_app_password*

  Specify your Bitbucket App Password (Basic). This option is secret.

- `Bearer token` *bb_access_token*

  Enter the OAuth access token (Bearer). This option is secret.

Cached convenience

- `(auto) User UUID` *user_uuid*

  Cached after using the *bb_me* command.

- `(auto) Username` *username*

  Cached after using the *bb_me* command.

Commands

Auth Options

- `bb_auth_set_mode`

  Set the authentication mode: auto|basic|bearer.

- `bb_set_app_password`

  Set App Password credentials including username and app password.

- `bb_set_bearer`

  Set the Bearer authentication token.

- `bb_auth_check`

  Run diagnostics to show authentication results for */user*.

User Management

- `bb_me`

  Retrieve details for the authenticated user.

- `bb_user_get`

  Fetch user information by username.

- `bb_workspaces_list`

  List all accessible workspaces.

Repositories Management

- `bb_repos_list`

Display a list of repositories.

- `bb_repo_get`

  Fetch details of a specific repository.

- `bb_repo_create`

  Create a new repository in a specified workspace.

- `bb_repo_delete`

  Delete a repository (requires confirmation).

Contents Management

- `bb_contents_get`

  Retrieve file or directory contents from a repository.

- `bb_file_put`

  Create or update a file in a repository.

- `bb_file_delete`

  Delete specified files within a repository.

Issues Management

- `bb_issues_list`

  List issues in a repository.

- `bb_issue_create`

  Create a new issue within a repository.

- `bb_issue_comment`

  Add a comment to an existing issue.

- `bb_issue_update`

  Update details of an existing issue.

Pull Requests Management

- `bb_prs_list`

  Display a list of pull requests.

- `bb_pr_create`

  Create a new pull request.

- `bb_pr_merge`

  Merge an existing pull request.

Search Functionality

- `bb_search_repos`

  Search repositories using Bitbucket Query Language (BBQL).

Chat with files (LlamaIndex, inline)

Plugin integrates `LlamaIndex` storage in any chat and provides additional knowledge into context.

Options

- `Ask LlamaIndex first` *ask_llama_first*

When enabled, then *LlamaIndex* will be asked first, and response will be used as additional knowledge in prompt. When disabled, then *LlamaIndex* will be asked only when needed. **INFO: Disabled in autonomous mode (via plugin)!** *Default: False*

- `Auto-prepare question before asking LlamaIndex first` *prepare_question*

When enabled, then question will be prepared before asking LlamaIndex first to create best query.

- `Model for question preparation` *model_prepare_question*

Model used to prepare question before asking LlamaIndex. *Default: gpt-3.5-turbo*

- `Max output tokens for question preparation` *prepare_question_max_tokens*

Max tokens in output when preparing question before asking LlamaIndex. *Default: 500*

- `Prompt for question preparation` *syntax_prepare_question*

System prompt for question preparation.

- `Max characters in question` *max_question_chars*

Max characters in question when querying LlamaIndex, 0 = no limit, default: *1000*

- `Append metadata to context` *append_meta*

If enabled, then metadata from LlamaIndex will be appended to additional context. *Default: False*

- `Model` *model_query*

Model used for querying `LlamaIndex`. *Default:* `gpt-3.5-turbo`

- `Index name` *idx*

Indexes to use. If you want to use multiple indexes at once then separate them by comma. *Default: base*

Code Interpreter

Executing Code

From version `2.4.13` with built-in `IPython`.

The plugin operates similarly to the `Code Interpreter` in `ChatGPT`, with the key difference that it works locally on the user's system. It allows for the execution of any Python code on the computer that the model may generate. When combined with the `Files I/O` plugin, it facilitates running code from files saved in the `data` directory. You can also prepare your own code files and enable the model to use them or add your own plugin for this purpose. You can execute commands and code on the host machine or in Docker container.

IPython: Starting from version `2.4.13`, it is highly recommended to adopt the new option: `IPython`, which offers significant improvements over previous workflows. IPython provides a robust environment for executing code within a kernel, allowing you to maintain the state of your session by preserving the results of previous commands. This feature is particularly useful for iterative development and data analysis, as it enables you to build upon prior computations without starting from scratch. Moreover, IPython supports the use of magic commands, such as `!pip install <package_name>`, which facilitate the installation of new packages directly within the session. This capability streamlines the process of managing dependencies and enhances the flexibility of your development environment. Overall, IPython offers a more efficient and user-friendly experience for executing and managing code.

To use IPython in sandbox mode, Docker must be installed on your system.

You can find the installation instructions here: https://docs.docker.com/engine/install/

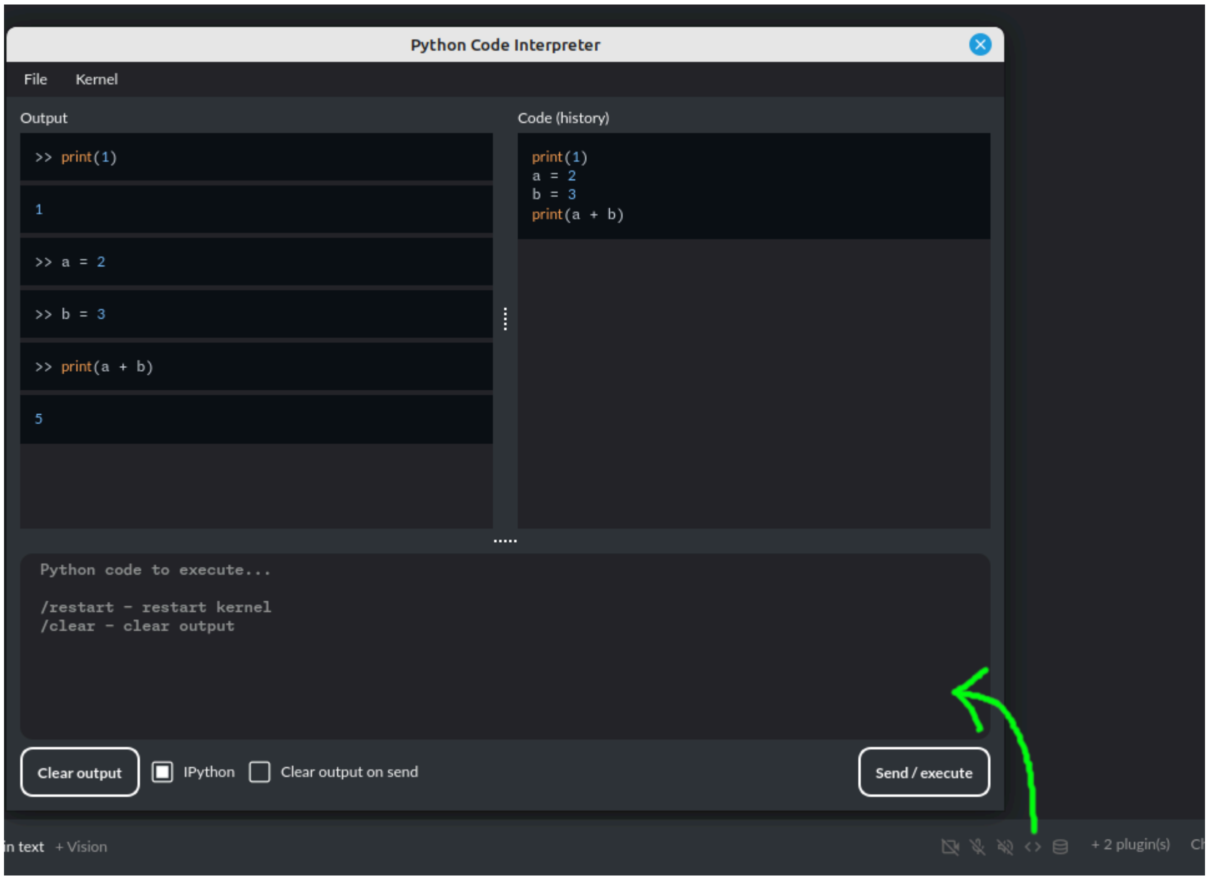
Connecting IPython in Docker in Snap version:

To use IPython in the Snap version, you must connect PyGPT to the Docker daemon:

```
$ sudo snap connect pygpt:docker-executables docker:docker-executables
```

```
$ sudo snap connect pygpt:docker docker:docker-daemon
```

Code interpreter: a real-time Python code interpreter is built-in. Click the `<>` icon to open the interpreter window. Both the input and output of the interpreter are connected to the plugin. Any output generated by the executed code will be displayed in the interpreter. Additionally, you can request the model to retrieve contents from the interpreter window output.



```
>> print(1)
1
>> a = 2
>> b = 3
>> print(a + b)
5
```

```
print(1)
a = 2
b = 3
print(a + b)
```

```
Python code to execute...

/restart - restart kernel
/clear - clear output
```

Important

Executing Python code using IPython in compiled versions requires an enabled sandbox (Docker container). You can connect the Docker container via `Plugins -> Settings`.

Tip

always remember to enable the `+ Tools` option to allow execute commands from the plugins.

Options:

General

- `Connect to the Python code interpreter window` *attach_output*

Automatically attach code input/output to the Python code interpreter window. *Default:* `True`

- `Tool: get_python_output` *cmd.get_python_output*

Allows `get_python_output` command execution. If enabled, it allows retrieval of the output from the Python code interpreter window. *Default:* `True`

- `Tool: get_python_input` *cmd.get_python_input*

Allows `get_python_input` command execution. If enabled, it allows retrieval all input code (from edit section) from the Python code interpreter window. *Default:* `True`

- `Tool: clear_python_output` *cmd.clear_python_output*

Allows `clear_python_output` command execution. If enabled, it allows clear the output of the Python code interpreter window. *Default:* `True`

IPython

- `Sandbox (docker container)` *sandbox_ipython*

Executes IPython in sandbox (docker container). Docker must be installed and running.

- `Dockerfile` *ipython_dockerfile*

You can customize the Dockerfile for the image used by IPython by editing the configuration above and rebuilding the image via Tools -> Rebuild IPython Docker Image.

- `Session Key` *ipython_session_key*

It must match the key provided in the Dockerfile.

- `Docker image name` *ipython_image_name*

Custom Docker image name

- `Docker container name` *ipython_container_name*

Custom Docker container name

- `Connection address` *ipython_conn_addr*

Default: 127.0.0.1

- `Port: shell` *ipython_port_shell*

Default: 5555

- `Port: iopub` *ipython_port_iopub*

Default: 5556

- `Port: stdin` *ipython_port_stdin*

Default: 5557

- `Port: control` *ipython_port_control*

Default: 5558

- `Port: hb` *ipython_port_hb*

Default: 5559

- `Tool: ipython_execute` *cmd.ipython_execute*

Allows Python code execution in IPython interpreter (in current kernel). *Default:* `True`

- `Tool: python_kernel_restart` *cmd.ipython_kernel_restart*

Allows to restart IPython kernel. *Default:* `True`

Python (legacy)

- `Sandbox (docker container)` *sandbox_docker*

Executes commands in sandbox (docker container). Docker must be installed and running.

- `Python command template` *python_cmd_tpl*

Python command template (use {filename} as path to file placeholder). *Default:* `python3 {filename}`

- `Dockerfile` *dockerfile*

You can customize the Dockerfile for the image used by legacy Python by editing the configuration above and rebuilding the image via Tools -> Rebuild Python (Legacy) Docker Image.

- `Docker image name` *image_name*

Custom Docker image name

- `Docker container name` *container_name*

Custom Docker container name

- `Tool: code_execute` *cmd.code_execute*

Allows `code_execute` command execution. If enabled, provides Python code execution (generate and execute from file). *Default:* `True`

- `Tool: code_execute_all` *cmd.code_execute_all*

Allows `code_execute_all` command execution. If enabled, provides execution of all the Python code in interpreter window. *Default:* `True`

- `Tool: code_execute_file` *cmd.code_execute_file*

Allows `code_execute_file` command execution. If enabled, provides Python code execution from existing .py file. *Default:* `True`

HTML Canvas

- `Tool: render_html_output` *cmd.render_html_output*

Allows `render_html_output` command execution. If enabled, it allows to render HTML/JS code in built-it HTML/JS browser (HTML Canvas). *Default:* `True`

- `Tool: get_html_output` *cmd.get_html_output*

Allows `get_html_output` command execution. If enabled, it allows retrieval current output from HTML Canvas. *Default:* `True`

- `Sandbox (docker container)` *sandbox_docker*

Execute commands in sandbox (docker container). Docker must be installed and running. *Default:* `False`

- `Docker image` *sandbox_docker_image*

Docker image to use for sandbox *Default:* `python:3.8-alpine`

Context history (calendar, inline)

Provides access to context history database. Plugin also provides access to reading and creating day notes.

Examples of use, you can ask e.g. for the following:

- Give me today day note
- Save a new note for today
- Update my today note with…
- Get the list of yesterday conversations
- Get contents of conversation ID 123

etc.

You can also use `@` ID tags to automatically use summary of previous contexts in current discussion. To use context from previous discussion with specified ID use following syntax in your query:

```
@123
```

Where `123` is the ID of previous context (conversation) in database, example of use:

```
Let's talk about discussion @123
```

Options

- `Enable: using context @ ID tags` *use_tags*

When enabled, it allows to automatically retrieve context history using @ tags, e.g. use @123 in question to use summary of context with ID 123 as additional context. *Default: False*

- `Tool: get date range context list` *cmd.get_ctx_list_in_date_range*

Allows *get_ctx_list_in_date_range* command execution. If enabled, it allows getting the list of context history (previous conversations). *Default: True*

- `Tool: get context content by ID` *cmd.get_ctx_content_by_id*

Allows *get_ctx_content_by_id* command execution. If enabled, it allows getting summarized content of context with defined ID.

Default: True

- `Tool: count contexts in date range` *cmd.count_ctx_in_date*

Allows *count_ctx_in_date* command execution. If enabled, it allows counting contexts in date range. *Default: True*

- `Tool: get day note` *cmd.get_day_note*

Allows *get_day_note* command execution. If enabled, it allows retrieving day note for specific date. *Default: True*

- `Tool: add day note` *cmd.add_day_note*

Allows *add_day_note* command execution. If enabled, it allows adding day note for specific date. *Default: True*

- `Tool: update day note` *cmd.update_day_note*

Allows *update_day_note* command execution. If enabled, it allows updating day note for specific date. *Default: True*

- `Tool: remove day note` *cmd.remove_day_note*

Allows *remove_day_note* command execution. If enabled, it allows removing day note for specific date. *Default: True*

- `Model` *model_summarize*

Model used for summarize. *Default: gpt-3.5-turbo*

- `Max summary tokens` *summary_max_tokens*

Max tokens in output when generating summary. *Default: 1500*

- `Max contexts to retrieve` *ctx_items_limit*

Max items in context history list to retrieve in one query. 0 = no limit. *Default: 30*

- `Per-context items content chunk size` *chunk_size*

Per-context content chunk size (max characters per chunk). *Default: 100000 chars*

Options (advanced)

- `Prompt: @ tags (system)` *prompt_tag_system*

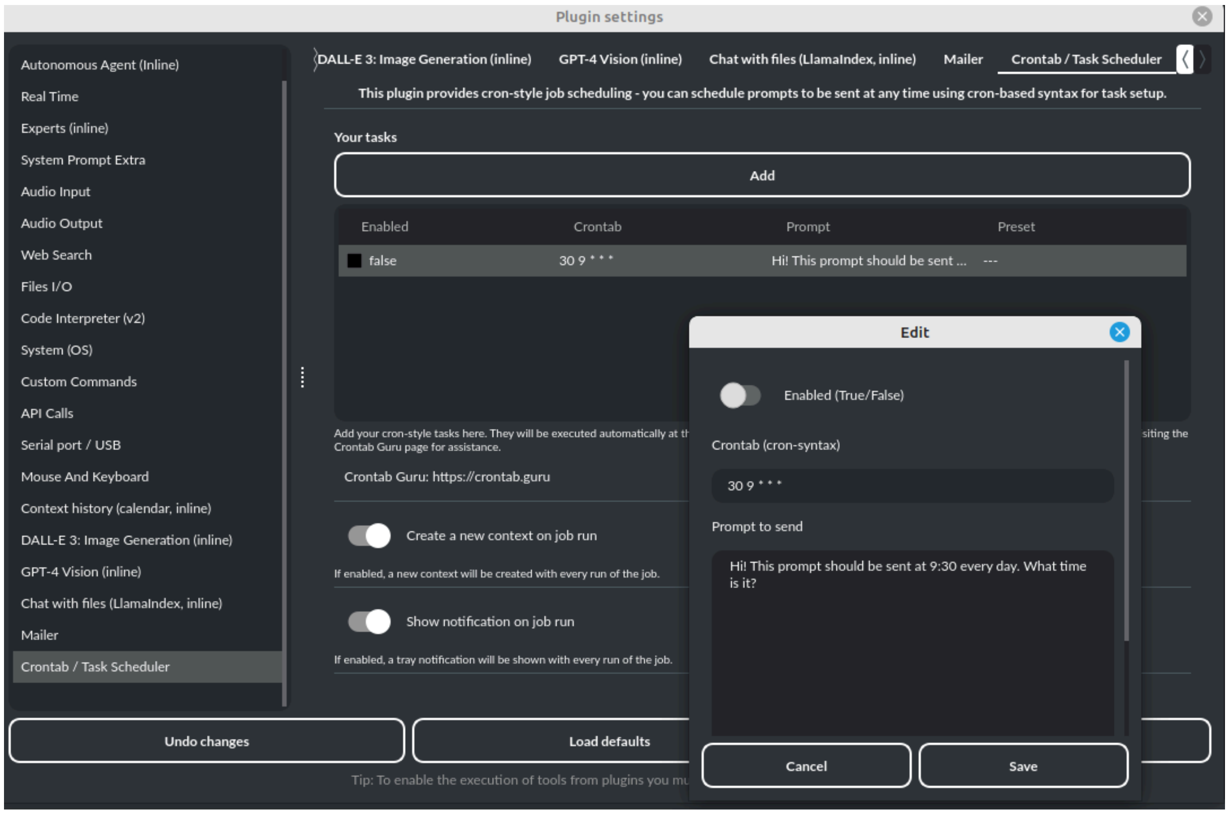
Prompt for use @ tag (system).

- `Prompt: @ tags (summary)` *prompt_tag_summary*

Prompt for use @ tag (summary).

Crontab / Task scheduler

Plugin provides cron-based job scheduling - you can schedule tasks/prompts to be sent at any time using cron-based syntax for task setup.



This plugin provides cron-style job scheduling - you can schedule prompts to be sent at any time using cron-based syntax for task setup.

Your tasks

Enabled	Crontab	Prompt	Preset
false	30 9 * * *	Hi! This prompt should be sent ...	---

Add your cron-style tasks here. They will be executed automatically at th… siting the Crontab Guru page for assistance.

Crontab Guru: https://crontab.guru

Create a new context on job run

If enabled, a new context will be created with every run of the job.

Show notification on job run

If enabled, a tray notification will be shown with every run of the job.

Tip: To enable the execution of tools from plugins you mu…

Enabled (True/False)

Crontab (cron-syntax)

30 9 * * *

Prompt to send

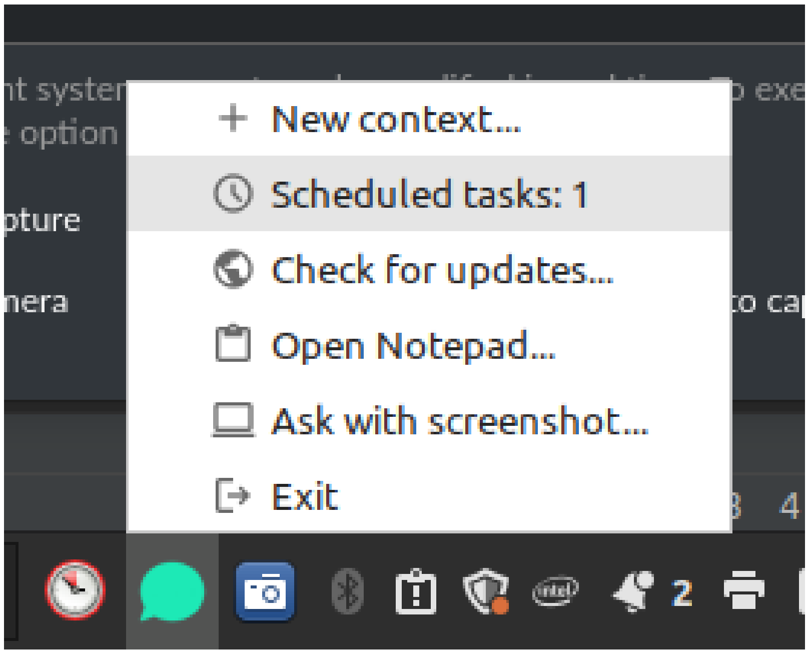
Hi! This prompt should be sent at 9:30 every day. What time is it?

Options

- `Your tasks` *crontab*

Add your cron-style tasks here. They will be executed automatically at the times you specify in the cron-based job format. If you are unfamiliar with Cron, consider visiting the Cron Guru page for assistance: https://crontab.guru

Number of active tasks is always displayed in a tray dropdown menu:



- `Create a new context on job run` *new_ctx*

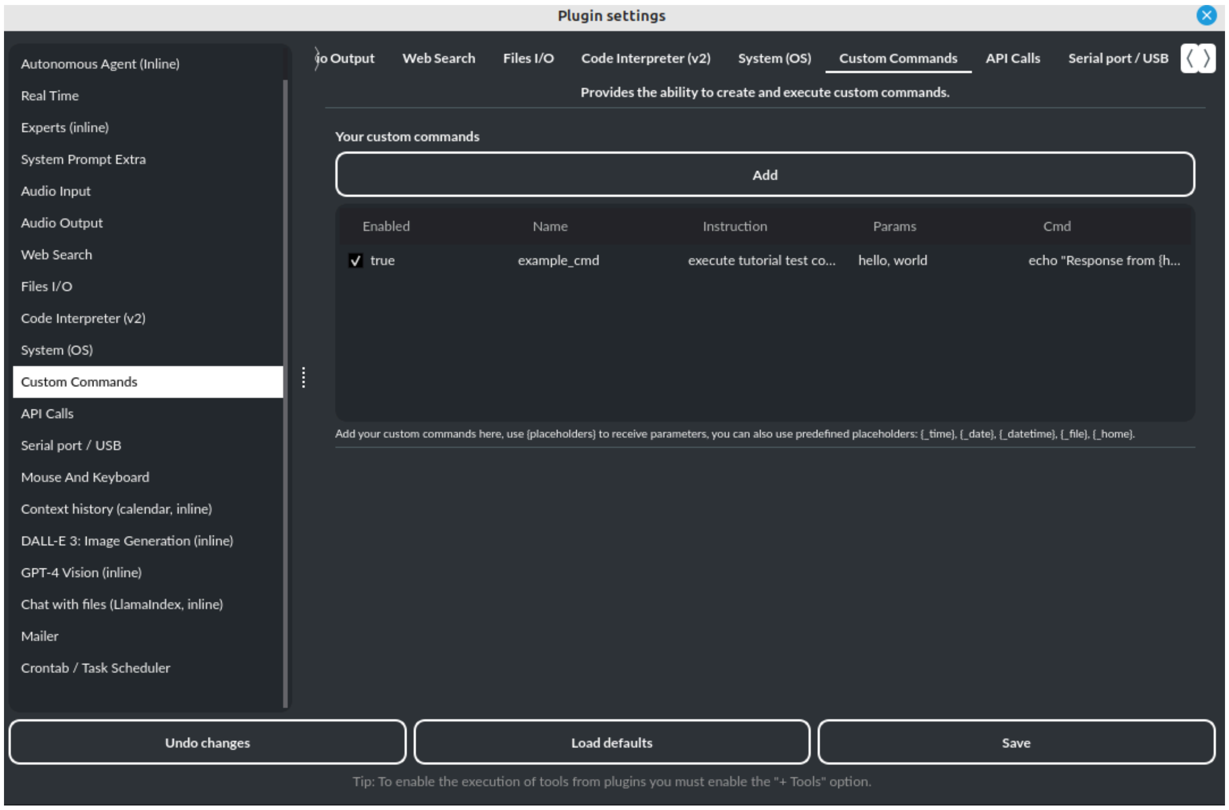
If enabled, then a new context will be created on every run of the job." *Default: True*

- `Show notification on job run` *show_notify*

If enabled, then a tray notification will be shown on every run of the job. *Default: True*

Custom Commands

With the `Custom Commands` plugin, you can integrate **PyGPT** with your operating system and scripts or applications. You can define an unlimited number of custom commands and instruct model on when and how to execute them. Configuration is straightforward, and **PyGPT** includes a simple tutorial command for testing and learning how it works:



Provides the ability to create and execute custom commands.

Your custom commands

Enabled	Name	Instruction	Params	Cmd
✓ true	example_cmd	execute tutorial test co...	hello, world	echo "Response from {h...

Add your custom commands here, use {placeholders} to receive parameters, you can also use predefined placeholders: {_time}, {_date}, {_datetime}, {_file}, {_home}.

Tip: To enable the execution of tools from plugins you must enable the "+ Tools" option.

To add a new custom command, click the **ADD** button and then:

1. Provide a name for your command: this is a unique identifier for model.
2. Provide an `instruction` explaining what this command does; model will know when to use the command based on this instruction.
3. Define `params`, separated by commas - model will send data to your commands using these params. These params will be placed into placeholders you have defined in the `cmd` field. For example:

If you want instruct model to execute your Python script named `smart_home_lights.py` with an argument, such as `1` to turn the light ON, and `0` to turn it OFF, define it as follows:

- **name**: lights_cmd

- **instruction**: turn lights on/off; use 1 as 'arg' to turn ON, or 0 as 'arg' to turn OFF
- **params**: arg
- **cmd**: `python /path/to/smart_home_lights.py {arg}`

The setup defined above will work as follows:

When you ask model to turn your lights ON, model will locate this command and prepare the command `python /path/to/smart_home_lights.py {arg}` with `{arg}` replaced with `1`. On your system, it will execute the command:

```
python /path/to/smart_home_lights.py 1
```

And that's all. Model will take care of the rest when you ask to turn ON the lights.

You can define as many placeholders and parameters as you desire.

Here are some predefined system placeholders for use:

- `{_time}` - current time in `H:M:S` format
- `{_date}` - current date in `Y-m-d` format
- `{_datetime}` - current date and time in `Y-m-d H:M:S` format
- `{_file}` - path to the file from which the command is invoked
- `{_home}` - path to PyGPT's home/working directory

You can connect predefined placeholders with your own params.

Example:

- **name**: song_cmd
- **instruction**: store the generated song on hard disk
- **params**: song_text, title
- **cmd**: `echo "{song_text}" > {_home}/{title}.txt`

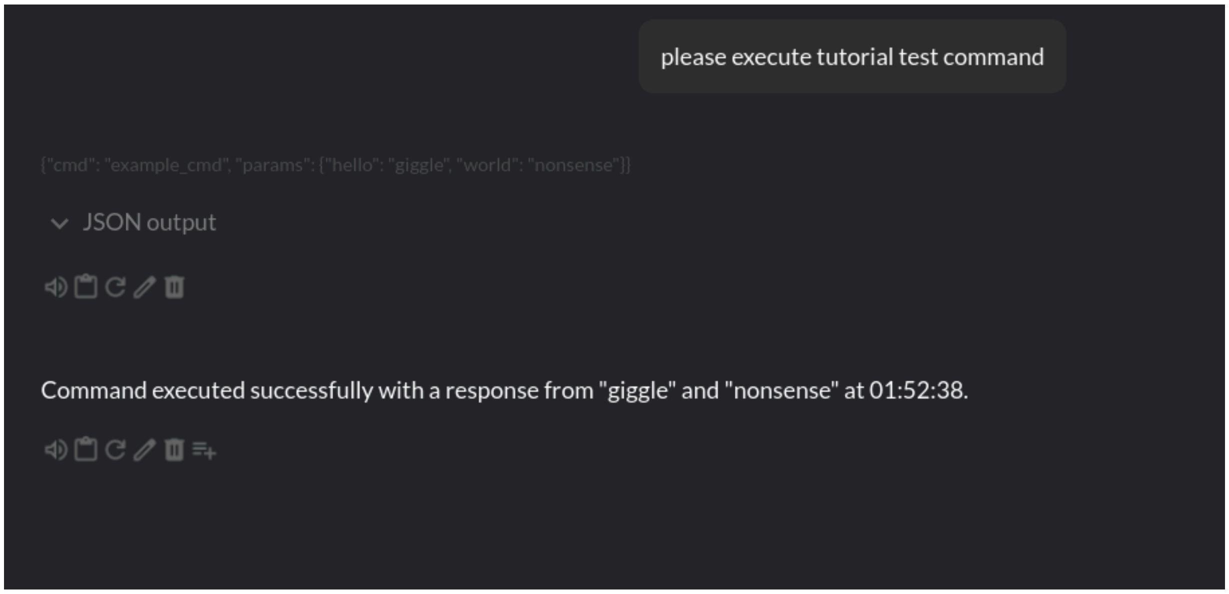
With the setup above, every time you ask model to generate a song for you and save it to the disk, it will:

1. Generate a song.
2. Locate your command.
3. Execute the command by sending the song's title and text.
4. The command will save the song text into a file named with the song's title in the **PyGPT** working directory.

Example tutorial command

PyGPT provides simple tutorial command to show how it work, to run it just ask model for execute `tutorial test command` and it will show you how it works:

```
> please execute tutorial test command
```



please execute tutorial test command

{"cmd": "example_cmd", "params": {"hello": "giggle", "world": "nonsense"}}

Command executed successfully with a response from "giggle" and "nonsense" at 01:52:38.

Experts (inline)

The plugin allows calling experts in any chat mode. This is the inline Experts (co-op) mode.

See the `Work modes -> Experts` section for more details.

Facebook

The plugin integrates with Facebook's Graph API to enable various actions such as managing pages, posts, and media uploads. It uses OAuth2 for authentication and supports automatic token exchange processes.

- Retrieving basic information about the authenticated user.
- Listing all Facebook pages the user has access to.
- Setting a specified Facebook page as the default.
- Retrieving a list of posts from a Facebook page.
- Creating a new post on a Facebook page.
- Deleting a post from a Facebook page.
- Uploading a photo to a Facebook page.

Options

- `Graph API Version` *graph_version*

Specify the API version. *Default: v21.0*

- `API Base` *api_base*

Base address for the Graph API. The version will be appended automatically.

- `Authorize Base` *authorize_base*

Base address for OAuth authorization. The version will be appended automatically.

- `HTTP Timeout (s)` *http_timeout*

Set the timeout for HTTP requests in seconds. *Default: 30*

OAuth2 (PKCE) Settings

- `App ID (client_id)` *oauth2_client_id*

Provide your Facebook App ID.

- `App Secret (optional)` *oauth2_client_secret*

Required for long-lived token exchange unless using PKCE. *Secret*

- `Confidential Client` *oauth2_confidential*

Use *client_secret* on exchange instead of *code_verifier*.

- `Redirect URI` *oauth2_redirect_uri*

Matches one of the valid OAuth Redirect URIs in your Meta App.

- `Scopes` *oauth2_scopes*

Space-separated authorized permissions.

- `User Access Token` *oauth2_access_token*

Stores user access token. *Secret*

Cache

- `User ID` *user_id*

Cached after calling *fb_me* or OAuth exchange.

- `User Name` *user_name*

Cached after calling *fb_me* or OAuth exchange.

- `Default Page ID` *fb_page_id*

Selected via *fb_page_set_default*.

- `Default Page Name` *fb_page_name*

Selected via *fb_page_set_default*.

- `Default Page Access Token` *fb_page_access_token*

Cached with *fb_page_set_default* or on demand. *Secret*

OAuth UX Options

- `Auto-start OAuth` *oauth_auto_begin*

Automatically begin PKCE flow when commands need a user token.

- `Open Browser Automatically` *oauth_open_browser*

Open authorization URL in the default web browser.

- `Use Local Server for OAuth` *oauth_local_server*

Start a local HTTP server to capture redirect.

- `OAuth Local Timeout (s)` *oauth_local_timeout*

Duration to wait for a redirect with code. *Default: 180*

- `Success HTML` *oauth_success_html*

HTML displayed on successful local callback.

- `Fail HTML` *oauth_fail_html*

HTML displayed on callback error.

- `OAuth Local Port` *oauth_local_port*

Set the local HTTP port; should be above 1024 and allowed in the app. *Default: 8732*

- `Allow Fallback Port` *oauth_allow_port_fallback*

Choose a free local port if the preferred port is busy or forbidden.

Commands

- `Auth: Begin OAuth2` *fb_oauth_begin*

Starts OAuth2 (PKCE) flow and returns the authorization URL.

- `Auth: Exchange Code` *fb_oauth_exchange*

Trades authorization code for a user access token.

- `Auth: Extend User Token` *fb_token_extend*

Exchanges a short-lived token for a long-lived token; requires app secret.

- `Users: Me` *fb_me*

Retrieves the authorized user’s profile.

- `Pages: List` *fb_pages_list*

Lists pages the user manages with details like ID, name, and access token.

- `Pages: Set Default` *fb_page_set_default*

Caches name and access token for a default page.

- `Posts: List` *fb_page_posts*

Retrieves the page’s feed (posts).

- `Posts: Create` *fb_page_post_create*

Publishes a post with optional text, links, and photos.

- `Posts: Delete` *fb_page_post_delete*

Removes a specified page post.

- `Media: Upload Photo` *fb_page_photo_upload*

Uploads a photo to a page from a local path or URL.

Files I/O

The plugin allows for file management within the local filesystem. It enables the model to create, read, write and query files located in the `data` directory, which can be found in the user's work directory. With this plugin, the AI can also generate Python code files and thereafter execute that code within the user's system.

Plugin capabilities include:

- Sending files as attachments
- Reading files
- Appending to files
- Writing files
- Deleting files and directories
- Listing files and directories
- Creating directories
- Downloading files
- Copying files and directories
- Moving (renaming) files and directories
- Reading file info
- Indexing files and directories using LlamaIndex

- Querying files using LlamaIndex

- Searching for files and directories

If a file being created (with the same name) already exists, a prefix including the date and time is added to the file name.

Options:

General

- `Tool: send (upload) file as attachment` *cmd.send_file*

Allows *send_file* command execution. *Default: True*

- `Tool: read file` *cmd.read_file*

Allows *read_file* command execution. *Default: True*

- `Tool: append to file` *cmd.append_file*

Allows *append_file* command execution. Text-based files only (plain text, JSON, CSV, etc.) *Default: True*

- `Tool: save file` *cmd.save_file*

Allows *save_file* command execution. Text-based files only (plain text, JSON, CSV, etc.) *Default: True*

- `Tool: delete file` *cmd.delete_file*

Allows *delete_file* command execution. *Default: True*

- `Tool: list files (ls)` *cmd.list_files*

Allows *list_dir* command execution. *Default: True*

- `Tool: list files in dirs in directory (ls)` *cmd.list_dir*

Allows *mkdir* command execution. *Default: True*

- `Tool: downloading files` *cmd.download_file*

Allows *download_file* command execution. *Default: True*

- `Tool: removing directories` *cmd.rmdir*

Allows *rmdir* command execution. *Default: True*

- `Tool: copying files` *cmd.copy_file*

Allows *copy_file* command execution. *Default: True*

- `Tool: copying directories (recursive)` *cmd.copy_dir*

Allows *copy_dir* command execution. *Default: True*

- `Tool: move files and directories (rename)` *cmd.move*

Allows *move* command execution. *Default: True*

- `Tool: check if path is directory` *cmd.is_dir*

Allows *is_dir* command execution. *Default: True*

- `Tool: check if path is file` *cmd.is_file*

Allows *is_file* command execution. *Default: True*

- `Tool: check if file or directory exists` *cmd.file_exists*

Allows *file_exists* command execution. *Default: True*

- `Tool: get file size` *cmd.file_size*

Allows *file_size* command execution. *Default: True*

- `Tool: get file info` *cmd.file_info*

Allows *file_info* command execution. *Default: True*

- `Tool: find file or directory` *cmd.find*

Allows *find* command execution. *Default: True*

- `Tool: get current working directory` *cmd.cwd*

Allows *cwd* command execution. *Default: True*

- `Use data loaders` *use_loaders*

Use data loaders from LlamaIndex for file reading (*read_file* command). *Default: True*

Indexing

- `Tool: quick query the file with LlamaIndex` *cmd.query_file*

Allows *query_file* command execution (in-memory index). If enabled, model will be able to quick index file into memory and query it for data (in-memory index) *Default: True*

- `Model for query in-memory index` *model_tmp_query*

Model used for query temporary index for *query_file* command (in-memory index). *Default: gpt-3.5-turbo*

- `Tool: indexing files to persistent index` *cmd.file_index*

Allows *file_index* command execution. If enabled, model will be able to index file or directory using LlamaIndex (persistent index). *Default: True*

- `Index to use when indexing files` *idx*

ID of index to use for indexing files (persistent index). *Default: base*

- `Auto index reading files` *auto_index*

If enabled, every time file is read, it will be automatically indexed (persistent index). *Default: False*

- `Only index reading files` *only_index*

If enabled, file will be indexed without return its content on file read (persistent index). *Default: False*

GitHub

The plugin provides seamless integration with GitHub, allowing various operations such as repository management, issue tracking, pull requests, and more through GitHub's API. This plugin requires authentication, which can be configured using a Personal Access Token (PAT) or OAuth Device Flow.

- Retrieve details about your GitHub profile.
- Get information about a specific GitHub user.
- List repositories for a user or organization.
- Retrieve details about a specific repository.
- Create a new repository.
- Delete an existing repository.
- Retrieve the contents of a file in a repository.
- Upload or update a file in a repository.
- Delete a file from a repository.
- List issues in a repository.
- Create a new issue in a repository.
- Add a comment to an existing issue.
- Close an existing issue.
- List pull requests in a repository.
- Create a new pull request.
- Merge an existing pull request.
- Search for repositories based on a query.

- Search for issues based on a query.
- Search for code based on a query.

Options

- `API base` *api_base*

  Configure the base URL for GitHub's API. *Default: https://api.github.com*

- `Web base` *web_base*

  Set the GitHub website base URL. *Default: https://github.com*

- `API version header` *api_version*

  Specify the API version for requests. *Default: 2022-11-28*

- `HTTP timeout (s)` *http_timeout*

  Define timeout for API requests in seconds. *Default: 30*

OAuth Device Flow

- `OAuth Client ID` *oauth_client_id*

  Set the Client ID from your GitHub OAuth App. Supports Device Flow. *Secret*

- `Scopes` *oauth_scopes*

  List the space-separated OAuth scopes. *Default: repo read:org read:user user:email*

- `Open browser automatically` *oauth_open_browser*

  Automatically open the verification URL in the default browser. *Default: True*

- `Auto-start auth when required` *oauth_auto_begin*

  Start Device Flow automatically when a command requires a token. *Default: True*

Tokens

- `(auto) OAuth access token` *gh_access_token*

  Store OAuth access token for Device/Web. *Secret*

- `PAT token (optional)` *pat_token*

  Provide a Personal Access Token (classic or fine-grained) for authentication. *Secret*

- `Auth scheme` *auth_scheme*

  Choose the authentication scheme: *Bearer* or *Token* (use *Token* for PAT).

Cache

- `(auto) User ID` *user_id*

  Cache User ID after *gh_me* or authentication.

- `(auto) Username` *username*

  Cache username after *gh_me* or authentication.

Commands

- **Auth**
  - `gh_device_begin` - Begin OAuth Device Flow.
  - `gh_device_poll` - Poll for access token using device code.
  - `gh_set_pat` - Set Personal Access Token.
- **Users**

    - `gh_me` - Get authenticated user details.
    - `gh_user_get` - Retrieve user information by username.
- **Repositories**
    - `gh_repos_list` - List all repositories.
    - `gh_repo_get` - Get details for a specific repository.
    - `gh_repo_create` - Create a new repository.
    - `gh_repo_delete` - Delete an existing repository. (*Disabled by default*)
- **Contents**
    - `gh_contents_get` - Get file or directory contents.
    - `gh_file_put` - Create or update a file via Contents API.
    - `gh_file_delete` - Delete a file via Contents API.
- **Issues**
    - `gh_issues_list` - List issues in a repository.
    - `gh_issue_create` - Create a new issue.
    - `gh_issue_comment` - Comment on an issue.
    - `gh_issue_close` - Close an existing issue.
- **Pull Requests**
    - `gh_pulls_list` - List all pull requests.
    - `gh_pull_create` - Create a new pull request.
    - `gh_pull_merge` - Merge an existing pull request.
- **Search**
    - `gh_search_repos` - Search for repositories.
    - `gh_search_issues` - Search for issues and pull requests.
    - `gh_search_code` - Search for code across repositories.

Google (Gmail, Drive, Calendar, Contacts, YT, Keep, Docs, Maps, Colab)

The plugin integrates with various Google services, enabling features such as email management, calendar events, contact handling, and document manipulation through Google APIs.

Gmail

- Listing recent emails from Gmail.
- Listing all emails from Gmail.
- Searching emails in Gmail.
- Retrieving email details by ID in Gmail.
- Sending an email via Gmail.

Google Calendar

- Listing recent calendar events.
- Listing today's calendar events.
- Listing tomorrow's calendar events.
- Listing all calendar events.
- Retrieving calendar events by a specific date.
- Adding a new event to the calendar.
- Deleting an event from the calendar.

Google Keep

- Listing notes from Google Keep.
- Adding a new note to Google Keep.

Google Drive

- Listing files from Google Drive.
- Finding a file in Google Drive by its path.
- Downloading a file from Google Drive.
- Uploading a file to Google Drive.

YouTube

- Retrieving information about a YouTube video.
- Retrieving the transcript of a YouTube video.

Google Contacts

- Listing contacts from Google Contacts.
- Adding a new contact to Google Contacts.

Google Docs

- Creating a new document.
- Retrieving a document.
- Listing documents.
- Appending text to a document.
- Replacing text in a document.
- Inserting a heading in a document.
- Exporting a document.
- Copying from a template.

Google Maps

- Geocoding an address.
- Reverse geocoding coordinates.
- Getting directions between locations.
- Using the distance matrix.
- Text search for places.
- Finding nearby places.
- Generating static map images.

Google Colab

- Listing notebooks.
- Creating a new notebook.
- Adding a code cell.
- Adding a markdown cell.
- Getting a link to a notebook.
- Renaming a notebook.
- Duplicating a notebook.

Options

- `Google credentials.json (content)` *credentials*

  Paste the JSON content of your OAuth client or Service Account. This is mandatory for the plugin to access your Google services. *Secret:* Yes

- `OAuth token store (auto)` *oauth_token*

  Automatically stores and updates the refresh token necessary for Google service access. *Secret:* Yes

- `Use local server for OAuth` *oauth_local_server*

  Run a local server for the installed app OAuth flow to simplify the authentication process. *Default: True*

- `OAuth local port (0=random)` *oauth_local_port*

  Specify the port for *InstalledAppFlow.run_local_server*. A value of *0* lets the system choose a random available port. *Default: 0*

- `Scopes` *oauth_scopes*

  Define space-separated OAuth scopes for services like Gmail, Calendar, Drive, Contacts, YouTube, Docs, and Keep. Extend scopes to include Keep services if needed.

- `Impersonate user (Workspace DWD)` *impersonate_user*

  Optionally provide a subject for service account domain-wide delegation.

- `YouTube API Key (optional)` *youtube_api_key*

  If provided, allows fetching public video information without needing OAuth tokens. *Secret:* Yes

- `Allow unofficial YouTube transcript` *allow_unofficial_youtube_transcript*

  Enables the use of *youtube-transcript-api* for transcripts when official captions are unavailable. *Default: False*

- `Keep mode` *keep_mode*

  Determines the mode for accessing Keep: *official*, *unofficial*, or *auto*. *Default: auto*

- `Allow unofficial Keep` *allow_unofficial_keep*

  Use *gkeepapi* as a fallback for Keep services, requiring *keep_username* and *keep_master_token*. *Default: True*

- `Keep username (unofficial)` *keep_username*

  Set the email used for *gkeepapi*.

- `Keep master token (unofficial)` *keep_master_token*

  Provide the master token for *gkeepapi* usage, ensuring secure handling. *Secret:* Yes

- `Google Maps API Key` *google_maps_api_key*

  Necessary for accessing Google Maps features like Geocoding, Directions, and Distance Matrix. *Secret:* Yes

- `Maps API Key (alias)` *maps_api_key*

  Alias for *google_maps_api_key* for backward compatibility. *Secret:* Yes

Commands

- **Gmail**

    - `gmail_list_recent` - List n newest Gmail messages.
    - `gmail_list_all` - List all Gmail messages (paginated).
    - `gmail_search` - Search Gmail.
    - `gmail_get_by_id` - Get Gmail message by ID.
    - `gmail_send` - Send Gmail message.
- **Calendar**
    - `calendar_events_recent` - Upcoming events (from now).
    - `calendar_events_today` - Events for today (UTC day bounds).
    - `calendar_events_tomorrow` - Events for tomorrow (UTC day bounds).
    - `calendar_events_all` - All events in range.
    - `calendar_events_by_date` - Events for date or date range.
    - `calendar_add_event` - Add calendar event.
    - `calendar_delete_event` - Delete event by ID.
- **Keep**
    - `keep_list_notes` - List notes (Keep).
    - `keep_add_note` - Add note (Keep).
- **Drive**
    - `drive_list_files` - List Drive files.
    - `drive_find_by_path` - Find Drive file by path.
    - `drive_download_file` - Download Drive file.
    - `drive_upload_file` - Upload local file to Drive.
- **YouTube**
    - `youtube_video_info` - Get YouTube video info.
    - `youtube_transcript` - Get YouTube transcript.
- **Contacts**
    - `contacts_list` - List contacts.
    - `contacts_add` - Add new contact.
- **Google Docs**
    - `docs_create` - Create Google Doc.
    - `docs_get` - Get Google Doc (structure + plain text).
    - `docs_list` - List Google Docs.
    - `docs_append_text` - Append text to Google Doc.

  - `docs_replace_text` - Replace all text occurrences in Google Doc.
  - `docs_insert_heading` - Insert heading at end of Google Doc.
  - `docs_export` - Export Google Doc to file.
  - `docs_copy_from_template` - Make a copy of template Google Doc.
- **Google Maps**
  - `maps_geocode` - Geocode an address.
  - `maps_reverse_geocode` - Reverse geocode coordinates.
  - `maps_directions` - Get directions between origin and destination.
  - `maps_distance_matrix` - Distance Matrix for origins and destinations.
  - `maps_places_textsearch` - Places Text Search.
  - `maps_places_nearby` - Nearby Places.
  - `maps_static_map` - Generate Static Map image.
- **Google Colab**
  - `colab_list_notebooks` - List Colab notebooks on Drive.
  - `colab_create_notebook` - Create new Colab notebook.
  - `colab_add_code_cell` - Add code cell to notebook.
  - `colab_add_markdown_cell` - Add markdown cell to notebook.
  - `colab_get_link` - Get Colab edit link.
  - `colab_rename` - Rename notebook.
  - `colab_duplicate` - Duplicate notebook.

Image Generation (inline)

The plugin integrates `DALL-E 3` image generation with any chat mode. Simply enable it and request an image in Chat mode, using a standard model such as `GPT-4`. The plugin does not require the `+ Tools` option to be enabled.

Options

- `Prompt` *prompt*

The prompt is used to generate a query for the `DALL-E` image generation model, which runs in the background.

Mailer

Enables the sending, receiving, and reading of emails from the inbox. Currently, only SMTP is supported. More options coming soon.

Options

- `From (email)` *from_email*

From (email), e.g. me@domain.com

- `Tool: send_mail` *cmd.send_mail*

Allows `send_mail` command execution. If enabled, model will be able to sending emails.

- `Tool: receive_emails` *cmd.receive_emails*

Allows `receive_emails` command execution. If enabled, model will be able to receive emails from the server.

- `Tool: get_email_body` *cmd.get_email_body*

Allows `get_email_body` command execution. If enabled, model will be able to receive message body from the server.

- `SMTP Host` *smtp_host*

SMTP Host, e.g. smtp.domain.com

- `SMTP Port (Inbox)` *smtp_port_inbox*

SMTP Port, default: 995

- `SMTP Port (Outbox)` *smtp_port_outbox*

SMTP Port, default: 465

- `SMTP User` *smtp_user*

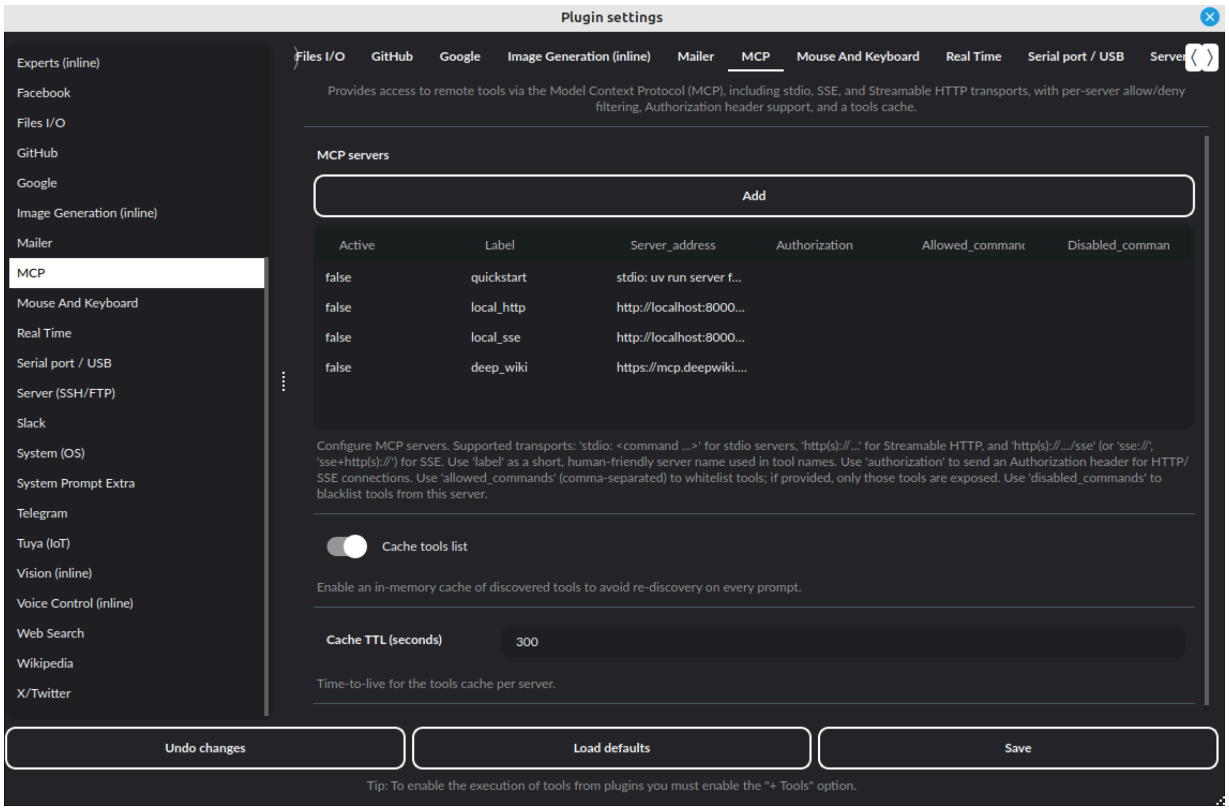
SMTP User, e.g. user@domain.com

- `SMTP Password` *smtp_password*

SMTP Password.

MCP

With the `MCP` plugin, you can connect **PyGPT** to remote tools exposed by Model Context Protocol servers (stdio, Streamable HTTP, or SSE). The plugin discovers available tools on your configured servers and publishes them to the model as callable commands with proper parameter schemas. You can whitelist/blacklist tools per server and optionally cache discovery results for speed.



Provides access to remote tools via the Model Context Protocol (MCP), including stdio, SSE, and Streamable HTTP transports, with per-server allow/deny filtering, Authorization header support, and a tools cache.

MCP servers

Active	Label	Server_address	Authorization	Allowed_commanc	Disabled_comman
false	quickstart	stdio: uv run server f...			
false	local_http	http://localhost:8000...			
false	local_sse	http://localhost:8000...			
false	deep_wiki	https://mcp.deepwiki....			

Configure MCP servers. Supported transports: 'stdio: <command ...>' for stdio servers, 'http(s)://...' for Streamable HTTP, and 'http(s)://.../sse' (or 'sse://', 'sse+http(s)://') for SSE. Use 'label' as a short, human-friendly server name used in tool names. Use 'authorization' to send an Authorization header for HTTP/SSE connections. Use 'allowed_commands' (comma-separated) to whitelist tools; if provided, only those tools are exposed. Use 'disabled_commands' to blacklist tools from this server.

Cache tools list

Enable an in-memory cache of discovered tools to avoid re-discovery on every prompt.

Cache TTL (seconds) 300

Time-to-live for the tools cache per server.

Tip: To enable the execution of tools from plugins you must enable the "+ Tools" option.

How it works

- You define one or more MCP servers in the plugin settings.
- For every active server, the plugin discovers its tools and exposes them to the model as commands.
- Each exposed command name is derived from the server label and tool name in the form `label__tool`. - Names are sanitized to allowed characters `[a-zA-Z0-9_-]` and kept under 64 characters. The plugin truncates and adds a short hash if needed to ensure uniqueness.
- If `allowed_commands` is set, only those tools are exposed (whitelist).
- If `disabled_commands` is set, those tools are hidden (blacklist).
- Discovery results can be cached (TTL) so you don't re-fetch the list on every prompt.
- When the model chooses a command, the plugin opens a session to the appropriate server and calls the tool with typed

arguments mapped from JSON Schema.

Adding a new MCP server

Click the **ADD** button and fill in:

1. label - A short, human-friendly server name used in tool names (`label__tool`). - It should be unique across servers. Only `[a-zA-Z0-9_-]` are used; other characters are replaced automatically.

2. server_address - Transport is detected from the address:

   - stdio: - `stdio: uv run server fastmcp_quickstart stdio` - `stdio: python /path/to/your_mcp_server.py --stdio`
   - Streamable HTTP: - `http://localhost:8000/mcp` - `https://your-host.tld/mcp`
   - SSE: - `http://localhost:8000/sse` - `sse://your-host.tld/sse` or `sse+https://your-host.tld/sse`

3. authorization (optional) - Value for the `Authorization` header on HTTP/SSE (e.g., `Bearer YOUR_TOKEN`). - Not used for stdio.

4. allowed_commands (optional) - Comma-separated list of tool names to expose from this server (whitelist). - Match the exact tool names reported by the MCP server (not titles).

5. disabled_commands (optional) - Comma-separated list of tool names to hide from this server (blacklist).

6. active - Enable/disable this server.

After saving, open any chat. During the command syntax build step, the plugin discovers tools on all active servers and publishes them to the model.

Example: quickstart via stdio

- label: `quickstart`
- server_address: `stdio: uv run server fastmcp_quickstart stdio`
- allowed_commands: (leave empty)
- disabled_commands: (leave empty)
- active: `ON`

Discovered commands might look like:

```
[
  {
    "cmd": "quickstart__echo",
    "instruction": "Echo a message back (server: quickstart)",
    "params": [{ "name": "message", "type": "str",
"description": "[required]" }],
    "enabled": true
  }
]
```

Example: remote docs server (HTTP/SSE)

- label: `deepwiki`
- server_address: `https://example.com/mcp` (or SSE: `https://example.com/sse`)
- authorization: `Bearer YOUR_TOKEN` (if required)
- allowed_commands: `read_wiki_structure, read_wiki_contents, ask_question`
- active: `ON`

The model will see commands like:

```
[
  {
    "cmd": "deepwiki__read_wiki_structure",
    "instruction": "Get a list of documentation topics for a
GitHub repository (server: deepwiki)",
    "params": [
      { "name": "repoName", "type": "str", "description":
"GitHub repository: owner/repo (e.g. \"facebook/react\")
[required]" }
    ],
    "enabled": true
  },
  {
    "cmd": "deepwiki__read_wiki_contents",
    "instruction": "View documentation about a GitHub
repository (server: deepwiki)",
    "params": [
      { "name": "repoName", "type": "str", "description":
"GitHub repository: owner/repo (e.g. \"facebook/react\")
[required]" }
    ],
    "enabled": true
  },
  {
    "cmd": "deepwiki__ask_question",
    "instruction": "Ask any question about a GitHub repository
(server: deepwiki)",
    "params": [
      { "name": "repoName", "type": "str", "description":
"GitHub repository: owner/repo (e.g. \"facebook/react\")
[required]" },
      { "name": "question", "type": "str", "description": "The
question to ask about the repository [required]" }
    ],
    "enabled": true
  }
]
```

Caching (Tools Cache)

- Enable “Cache tools list” to avoid discovering tools on every prompt.
- TTL defines how long (in seconds) the cache stays valid per server (default: 300).

- The plugin automatically invalidates the cache if you change server configuration (label, address, authorization, allow/deny lists).
- To force refresh immediately, toggle a server off/on or modify any of its fields and save.

Transports

- stdio: best for local processes started by PyGPT; no network involved.
- Streamable HTTP: recommended for production; uses a duplex HTTP stream.
- SSE: fully supported; useful for servers exposing an SSE endpoint.
- The plugin detects the transport from `server_address` automatically.

Security notes

- `authorization` is sent as the `Authorization` header to HTTP/SSE servers; put the exact value you need (e.g., `Bearer <token>`).
- Only connect to servers you trust. Tools can perform actions as implemented by the server.
- Keep labels short and non-sensitive (labels are visible in tool names and logs).

Mouse And Keyboard

Introduced in version: `2.4.4` (2024-11-09)

Warning

Use this plugin with caution - allowing all options gives the model full control over the mouse and keyboard

The plugin allows for controlling the mouse and keyboard by the model. With this plugin, you can send a task to the model, e.g., "open notepad, type something in it" or "open web browser, do search, find something."

Plugin capabilities include:

- Get mouse cursor position
- Control mouse cursor position
- Control mouse clicks
- Control mouse scroll
- Control the keyboard (pressing keys, typing text)
- Making screenshots

The `+ Tools` option must be enabled to use this plugin.

Options:

General

- `Prompt` *prompt*

Prompt used to instruct how to control the mouse and keyboard.

- `Enable: Allow mouse movement` *allow_mouse_move*

Allows mouse movement. *Default: True*

- `Enable: Allow mouse click` *allow_mouse_click*

Allows mouse click. *Default: True*

- `Enable: Allow mouse scroll` *allow_mouse_scroll*

Allows mouse scroll. *Default: True*

- `Enable: Allow keyboard key press` *allow_keyboard*

Allows keyboard typing. *Default: True*

- `Enable: Allow making screenshots` *allow_screenshot*

Allows making screenshots. *Default: True*

- `Tool: mouse_get_pos` *cmd.mouse_get_pos*

Allows `mouse_get_pos` command execution. *Default: True*

- `Tool: mouse_set_pos` *cmd.mouse_set_pos*

Allows `mouse_set_pos` command execution. *Default: True*

- `Tool: make_screenshot` *cmd.make_screenshot*

Allows `make_screenshot` command execution. *Default: True*

- `Tool: mouse_click` *cmd.mouse_click*

Allows `mouse_click` command execution. *Default: True*

- `Tool: mouse_move` *cmd.mouse_move*

Allows `mouse_move` command execution. *Default: True*

- `Tool: mouse_scroll` *cmd.mouse_scroll*

Allows `mouse_scroll` command execution. *Default: True*

- `Tool: keyboard_key` *cmd.keyboard_key*

Allows `keyboard_key` command execution. *Default: True*

- `Tool: keyboard_type` *cmd.keyboard_type*

Allows `keyboard_type` command execution. *Default: True*

OpenStreetMap

Provides everyday mapping utilities using OpenStreetMap services:

- Forward and reverse geocoding via Nominatim
- Search with optional near/bbox filters
- Routing via OSRM (driving, walking, cycling)
- Generate openstreetmap.org URL (center/zoom or bbox; optional marker)
- Utility helpers: open an OSM website URL centered on a point; download a single XYZ tile

By default no images are downloaded; commands return URLs. The `osm_tile` command saves tiles under `data/openstreetmap/` in the user data directory.

Important notes and usage etiquette

- Nominatim requires a proper User-Agent and recommends including a contact email. Set both in the plugin options.
- Public endpoints are community/demo services and not intended for heavy production use. If you need higher throughput, host your own services and set custom base URLs in options.
- Always provide attribution to OpenStreetMap contributors where required by the data license.

Options

- `HTTP timeout (s)` *http_timeout*
- `User-Agent` *user_agent*

- `Contact email (Nominatim)` *contact_email*
- `Accept-Language` *accept_language*
- `Nominatim base` *nominatim_base*
- `OSRM base` *osrm_base*
- `Tile base` *tile_base*
- `Default zoom` *map_zoom*
- `Default width` *map_width* Used only to estimate zoom for bbox (no image rendering).
- `Default height` *map_height* Used only to estimate zoom for bbox (no image rendering).

Tools (Commands)

- `Tool: osm_geocode`

  Forward geocoding of free-text addresses/places using Nominatim. Supports optional country filtering and near/viewbox bounding. Each result includes `map_url` (openstreetmap.org link centered on the result with a marker).

  Parameters: - `query` (str, required) - `limit` (int, optional) - `countrycodes` (str, optional, e.g. `pl,de`) - `viewbox` (str|list, optional) — `minlon,minlat,maxlon,maxlat` - `bounded` (bool, optional) - `addressdetails` (bool, optional) - `polygon_geojson` (bool, optional) - `near` (str, optional) — address or `lat,lon` to bias results (creates a viewbox) - `near_lat` (float, optional), `near_lon` (float, optional), `radius_m` (int, optional) - `zoom` (int, optional) — zoom used for `map_url` (defaults to option `Default zoom`) - `layers` (str, optional) — optional `layers=` value for OSM site URLs

- `Tool: osm_reverse`

  Reverse geocoding for a given coordinate. Response includes `map_url` (openstreetmap.org link).

Parameters: - `lat` (float) and `lon` (float) or `point` (str `lat,lon`) - `zoom` (int, optional) — also used for `map_url` zoom - `addressdetails` (bool, optional) - `layers` (str, optional) — optional `layers=` value for OSM site URLs

- `Tool: osm_search`

  Alias convenience wrapper for `osm_geocode` with the same parameters (results also include `map_url`).

- `Tool: osm_route`

  Plan a route via OSRM. Accepts addresses or coordinates for start/end and optional waypoints. Always returns `map_url` pointing to the openstreetmap.org Directions page (the route is drawn there). In addition: - mode=url — no OSRM call; returns only `map_url` (Directions) and waypoints, - mode=summary (default) — returns distance/duration (no geometry) + `map_url`, - mode=full — can include compact geometry (`geometry_polyline6`) and optionally steps.

  If `save_map` is true, an additional `preview_url` (regular map view centered/bbox) is returned; it does not replace `map_url`.

  Parameters: - `start` (str) or `start_lat`/`start_lon` - `end` (str) or `end_lat`/`end_lon` - `waypoints` (list, optional) - `profile` (str, optional) — `driving` | `walking` | `cycling` (default `driving`) - `mode` (str, optional) — `url` | `summary` | `full` (default `summary`) - `include_geometry` (bool, optional) — include compact geometry (polyline6) in `full` mode - `include_steps` (bool, optional) — include step-by-step (`full` mode only) - `alternatives` (int, optional) — 0|1; if >0 treated as true for OSRM alternatives - `max_polyline_chars` (int, optional) — limit geometry string length (default 5000) - `debug_url` (bool, optional) — include OSRM request URL in the response - `save_map` (bool, optional)

— build an additional `preview_url` (OSM map centered/bbox) - `zoom` (int, optional) — zoom for `preview_url` when applicable - `layers` (str, optional) — optional `layers=` for `preview_url` - `width` (int, optional), `height` (int, optional) — used only to estimate bbox zoom - `markers` (list, optional) — for `preview_url` the first valid point is used as marker

Deprecated/no-op parameters (kept for backward compatibility): `out`, `color`, `weight`.

- `Tool: osm_staticmap`

  Build an openstreetmap.org URL (center/zoom or bbox; optional marker). Only the first valid marker is used. `width`/`height` are used only to estimate zoom when a bbox is provided.

  Parameters: - `center` (str) or `lat`/`lon` (optional), `zoom` (int, optional) - `bbox` (list[4], optional) — `minlon,minlat,maxlon,maxlat` - `markers` (list, optional) — candidates for a single marker; the first valid point is used - `marker` (bool, optional) — if true and no markers provided, place a marker at center - `layers` (str, optional) — optional `layers=` value - `width` (int), `height` (int) — used only to estimate zoom for bbox

- `Tool: osm_bbox_map`

  Shortcut to build an openstreetmap.org URL from a bounding box.

  Parameters: - `bbox` (list[4], required) — `minlon,minlat,maxlon,maxlat` - optional `markers`, `width`, `height`

- `Tool: osm_show_url`

Build an openstreetmap.org URL centered at a point with a marker.

Parameters: - `point` (str) or `lat`/`lon` - `zoom` (int, optional) - `layers` (str, optional)

- `Tool: osm_route_url`

  Build an openstreetmap.org Directions URL for start/end (the route is drawn on the page).

  Parameters: - `start` (str) / `end` (str) or coordinate pairs - `mode` (str, optional) — `car` | `bike` | `foot`

- `Tool: osm_tile`

  Download a single XYZ tile (z/x/y.png). Useful for diagnostics or custom composition. The file is saved under `data/openstreetmap/` by default.

  Parameters: - `z` (int), `x` (int), `y` (int), `out` (str, optional)

Real Time

This plugin automatically adds the current date and time to each system prompt you send. You have the option to include just the date, just the time, or both.

When enabled, it quietly enhances each system prompt with current time information before sending it to model.

Options

- `Append time` *hour*

If enabled, it appends the current time to the system prompt.
Default: True

- `Append date` *date*

If enabled, it appends the current date to the system prompt.
Default: True

- `Template` *tpl*

Template to append to the system prompt. The placeholder `{time}` will be replaced with the current date and time in real-time. *Default: Current time is {time}.*

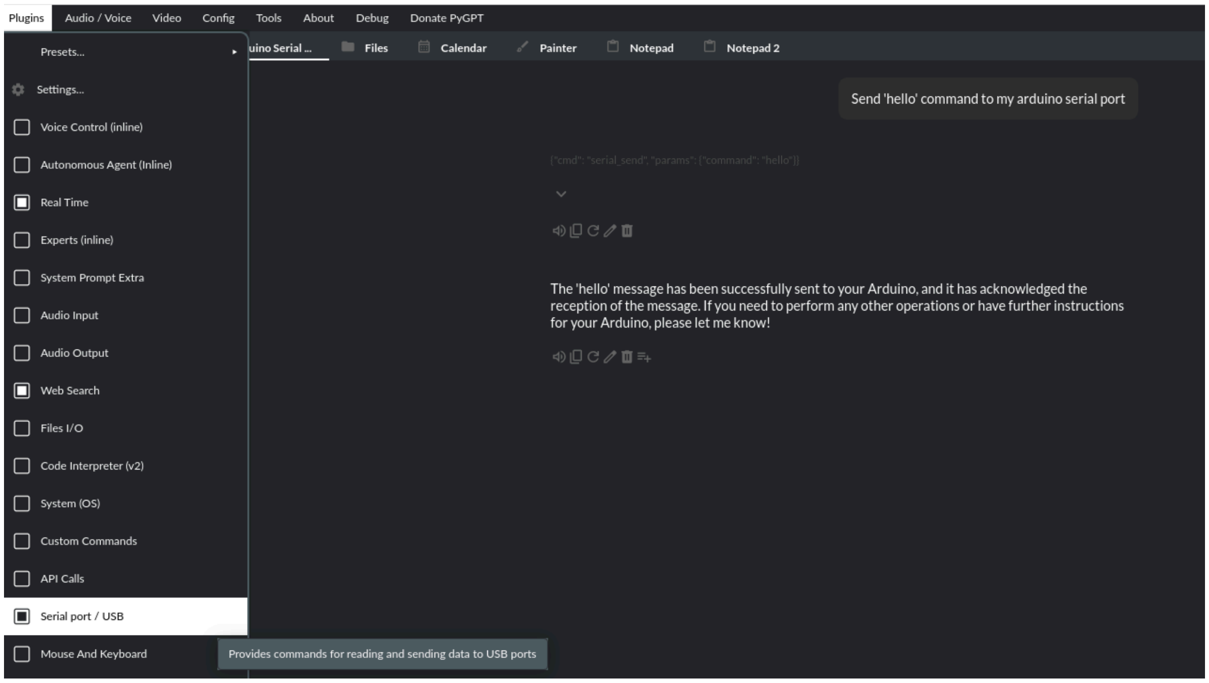
Serial port / USB

Provides commands for reading and sending data to USB ports.

Note

In the Snap version you must connect the interface first: https://snapcraft.io/docs/serial-port-interface

You can send commands to, for example, an Arduino or any other controllers using the serial port for communication.



Send 'hello' command to my arduino serial port

```
{"cmd": "serial_send", "params": {"command": "hello"}}
```

The 'hello' message has been successfully sent to your Arduino, and it has acknowledged the reception of the message. If you need to perform any other operations or have further instructions for your Arduino, please let me know!

Above is an example of co-operation with the following code uploaded to `Arduino Uno` and connected via USB:

```
// example.ino

void setup() {
  Serial.begin(9600);
}

void loop() {
  if (Serial.available() > 0) {
    String input = Serial.readStringUntil('\n');
    if (input.length() > 0) {
      Serial.println("OK, response for: " + input);
    }
  }
}
```

Options

- `USB port` *serial_port*

USB port name, e.g. /dev/ttyUSB0, /dev/ttyACM0, COM3, *Default:* `/dev/ttyUSB0`

- `Connection speed (baudrate, bps)` *serial_bps*

Port connection speed, in bps. *Default:* `9600`

- `Timeout` *timeout*

Timeout in seconds. *Default:* `1`

- `Sleep` *sleep*

Sleep in seconds after connection. *Default:* `2`

- `Tool: Send text commands to USB port` *cmd.serial_send*

Allows `serial_send` command execution". *Default: True*

- `Tool: Send raw bytes to USB port` *cmd.serial_send_bytes*

Allows `serial_send_bytes` command execution. *Default: True*

- `Tool: Read data from USB port` *cmd.serial_read*

Allows `serial_read` command execution. *Default: True*

Server (SSH/FTP)

The Server plugin provides integration for remote server management via SSH, SFTP, and FTP protocols. This plugin allows executing commands, transferring files, and managing directories on remote servers.

The plugin can be configured with various options to customize connectivity and feature access.

Options

- `Servers` *servers*

Define server configurations with credentials and server details. **The model does not access credentials, only names and ports.**

  - `enabled` - Enable or disable server configuration
  - `name` - Name of the server. **(visible for the model)**
  - `host` - Hostname of the server.
  - `login` - Login username.
  - `password` - Password for the connection (hidden).
  - `port` - Connection port (SSH by default). **(visible for the model)**
  - `desc` - Description of the server configuration.

- `Network timeout (s)` *net_timeout*

Set the timeout for network operations. *Default: 30*

- `Prefer system ssh/scp/sftp` *prefer_system_ssh*

Choose whether to use native ssh/scp/sftp binaries and system keys. *Default: False*

- `ssh binary` *ssh_binary*

Specify the path to the ssh binary. *Default: "ssh"*

- `scp binary` *scp_binary*

Specify the path to the scp binary. *Default: "scp"*

- `sftp binary` *sftp_binary*

Specify the path to the sftp binary. *Default: "sftp"*

- `Extra ssh options` *ssh_options*

Add extra options to be appended to ssh/scp commands. *Default:* "″

- `Paramiko: Auto add host keys` *ssh_auto_add_hostkey*

Enable automatic addition of host keys for Paramiko SSHClient. *Default: True*

- **FTP/FTPS**
    - `FTP TLS default` *ftp_use_tls_default*

      Choose whether to use FTP over TLS (explicit) by default. *Default: False*

    - `FTP passive mode` *ftp_passive_default*

      Set the default FTP mode to passive. *Default: True*

- **Telnet**
    - `Telnet: login prompt` *telnet_login_prompt*

      Expected prompt for username during Telnet login. *Default: "login:"*

    - `Telnet: password prompt` *telnet_password_prompt*

      Expected prompt for password input during Telnet login. *Default: "Password:"*

    - `Telnet: shell prompt` *telnet_prompt*

      Define the prompt used to delimit command output in Telnet. *Default: "$* "

- **SMTP**

  - `SMTP STARTTLS default` *smtp_use_tls_default*

    Enable STARTTLS by default for SMTP connections. *Default: True*

  - `SMTP SSL default` *smtp_use_ssl_default*

    Enable SMTP over SSL by default. *Default: False*

  - `Default From address` *smtp_from_default*

    Default address used if 'from_addr' not provided in smtp_send command. *Default: ""*

Commands

- `srv_exec`

  Execute remote shell command via SSH or Telnet.

- `srv_ls`

  List remote directory contents using SFTP/FTP or SSH.

- `srv_get`

  Download files from remote servers to local directories.

- `srv_put`

  Upload local files to remote servers.

- `srv_rm`

  Remove remote files or empty directories (non-recursive).

- `srv_mkdir`

Create directories on remote servers.

- `srv_stat`

  Retrieve information about remote files, such as type, size, and last modification time.

- `smtp_send`

  Send emails via SMTP, using the server configurations provided.

Slack

The Slack plugin integrates with the Slack Web API, enabling interaction with Slack workspaces through the application. This plugin supports OAuth2 for authentication, which allows for seamless integration with Slack services, enabling actions such as posting messages, retrieving users, and managing conversations.

- Retrieving a list of users.
- Listing all conversations.
- Accessing conversation history.
- Retrieving conversation replies.
- Opening a conversation.
- Posting a message in a chat.
- Deleting a chat message.
- Uploading files to Slack.

The plugin can be configured with various options to customize connectivity and feature access.

Options

- `API base` *api_base*

Set the base URL for Slack's API. *Default: https://slack.com/api*

- `OAuth base` *oauth_base*

Set the base URL for OAuth authorization. *Default: https://slack.com*

- `HTTP timeout (s)` *http_timeout*

Specify the request timeout in seconds. *Default: 30*

OAuth2 (Slack)

- `OAuth2 Client ID` *oauth2_client_id*

Provide the Client ID from your Slack App. This field is secret.

- `OAuth2 Client Secret` *oauth2_client_secret*

Provide the Client Secret from your Slack App. This field is secret.

- `Redirect URI` *oauth2_redirect_uri*

Specify the redirect URI that matches one in your Slack App.
Default: http://127.0.0.1:8733/callback

- `Bot scopes (comma-separated)` *bot_scopes*

Define the scopes for the bot token. *Default: chat:write,users:read,…*

- `User scopes (comma-separated)` *user_scopes*

Specify optional user scopes for user token if required.

Tokens/cache

- `(auto/manual) Bot token` *bot_token*

Input or obtain the bot token automatically or manually. This field is secret.

- `(auto) User token (optional)` *user_token*

Get the user token if user scopes are required. This field is secret.

- `(auto) Refresh token` *oauth2_refresh_token*

Store refresh token if rotation is enabled. This field is secret.

- `(auto) Expires at (unix)` *oauth2_expires_at*

Automatically calculate the token expiry time.

- `(auto) Team ID` *team_id*

Cache the Team ID after auth.test or OAuth.

- `(auto) Bot user ID` *bot_user_id*

Cache the Bot user ID post OAuth exchange.

- `(auto) Authed user ID` *authed_user_id*

Cache the authenticated user ID after auth.test/OAuth.

- `Auto-start OAuth when required` *oauth_auto_begin*

Enable automatic initiation of OAuth flow if a command needs a token. *Default: True*

- `Open browser automatically` *oauth_open_browser*

Open the authorize URL in default browser. *Default: True*

- `Use local server for OAuth` *oauth_local_server*

Activate local HTTP server to capture redirect. *Default: True*

- `OAuth local timeout (s)` *oauth_local_timeout*

Set time to wait for redirect with code. *Default: 180*

- `Success HTML` *oauth_success_html*

Specify HTML displayed on successful local callback.

- `Fail HTML` *oauth_fail_html*

Specify HTML displayed on failed local callback.

- `OAuth local port (0=auto)` *oauth_local_port*

Set local HTTP port; must be registered in Slack App. *Default: 8733*

- `Allow fallback port if busy` *oauth_allow_port_fallback*

Fallback to a free local port if preferred port is busy. *Default: True*

Commands

- `slack_oauth_begin`

Begin the OAuth2 flow and return the authorize URL.

- `slack_oauth_exchange`

Exchange authorization code for tokens.

- `slack_oauth_refresh`

Refresh token if rotation is enabled.

- `slack_auth_test`

Test authentication and retrieve IDs.

- `slack_users_list`

List workspace users (contacts).

- `slack_conversations_list`

List channels/DMs visible to the token.

- `slack_conversations_history`

Fetch channel/DM history.

- `slack_conversations_replies`

Fetch a thread by root ts.

- `slack_conversations_open`

Open or resume DM or MPDM.

- `slack_chat_post_message`

Post a message to a channel or DM.

- `slack_chat_delete`

Delete a message from a channel or DM.

- `slack_files_upload`

Upload a file via external flow and share in Slack.

System Prompt Extra (append)

The plugin appends additional system prompts (extra data) from a list to every current system prompt. You can enhance every system prompt with extra instructions that will be automatically appended to the system prompt.

Options

- `Prompts` *prompts*

List of extra prompts - prompts that will be appended to system prompt. All active extra prompts defined on list will be appended to the system prompt in the order they are listed here.

System (OS)

The plugin provides access to the operating system and executes system commands.

Options:

General

- `Auto-append CWD to sys_exec` *auto_cwd*

Automatically append current working directory to `sys_exec` command. *Default:* `True`

- `Tool: sys_exec` *cmd.sys_exec*

Allows `sys_exec` command execution. If enabled, provides system commands execution. *Default:* `True`

Sandbox (Docker)

- `Sandbox (docker container)` *sandbox_docker*

  Executes all `sys_exec` shell commands inside an isolated Docker container. Requires Docker to be installed and running. Default: `False`

- `Dockerfile` *dockerfile*

The Dockerfile used to build the sandbox image. You can customize it and rebuild via Tools → “Rebuild Docker sandbox Images”. Default: a minimal Python image with `/data` workdir.

- `Docker image name` *image_name*

  Name of the Docker image used by the sandbox. Default: `pygpt_system`

- `Docker container name` *container_name*

  Name of the Docker container started for the sandbox. Default: `pygpt_system_container`

- `Docker run command` *docker_entrypoint*

  Command executed when starting the container (keeps container alive). Default: `tail -f /dev/null`

- `Docker volumes` *docker_volumes*

  Host ↔ container volume mappings. By default, user data directory on host (`{workdir}`) is mapped read/write to `/data` in the container.

  Structure of each item:

  - `enabled` (bool) – include this mapping
  - `docker` (text) – container path (e.g. `/data`)
  - `host` (text) – host path (e.g. `{workdir}`)

  Default: one mapping of `{workdir}` → `/data`

- `Docker ports` *docker_ports*

  Host ↔ container port mappings. You can specify protocol on the container side (e.g. `8888/tcp`), otherwise TCP is assumed.

Structure of each item:

  - `enabled` (bool) – include this mapping
  - `docker` (text) – container port (e.g. `8888` or `8888/tcp`)
  - `host` (int) – host port (e.g. `8888`)

  Default: empty list (no ports exposed)

Notes:

- When sandboxing is enabled, relative paths passed in commands are resolved against the user data directory and mounted into the container at `/data`.
- The plugin checks for Docker availability and will prompt to build the image if it does not exist.

WinAPI (Windows)

- `Enable WinAPI` *winapi_enabled*

  Enables Windows Desktop/WinAPI integration (window management, input, screenshots) on Microsoft Windows. Default: `True`

- `Keys: per-char delay (ms)` *win_keys_per_char_delay_ms*

  Delay between characters when typing Unicode text with `win_keys_text`. Default: `2`

- `Keys: hold (ms)` *win_keys_hold_ms*

  Hold duration for modifier keys (e.g., CTRL/ALT/SHIFT) in `win_keys_send`. Default: `50`

- `Keys: gap (ms)` *win_keys_gap_ms*

  Gap between consecutive key taps in `win_keys_send`. Default: `30`

- `Drag: step delay (ms)` *win_drag_step_delay_ms*

  Delay between intermediate mouse-move steps during `win_drag`. Default: `10`

Notes:

- WinAPI features are available only on Microsoft Windows. On other platforms, WinAPI commands are not exposed.
- WinAPI commands are added to the command syntax list only when `platform == "Windows"` and `winapi_enabled` is `True`.
- Window and area screenshots use Qt's `QScreen.grabWindow(...)`; images are saved as PNG files (non-absolute paths are stored under the user data directory).

Telegram

The plugin enables integration with Telegram for both bots and user accounts through the `Bot API` and the `Telethon` library respectively. It allows sending and receiving messages, managing chats, and handling updates.

- Sending text messages to a chat or channel.
- Sending photos with an optional caption to a chat or channel.
- Sending documents or files to a chat or channel.
- Retrieving information about a specific chat or channel.
- Polling for updates in bot mode.
- Downloading files using a file identifier.
- Listing contacts in user mode.
- Listing recent dialogs or chats in user mode.
- Retrieving recent messages from a specific chat or channel in user mode.

Options

- `Mode` *mode*

  Choose the mode of operation. *Default: bot*

  Available modes:

  - Bot (via `Bot API`)
  - User (via `Telethon`)

- `API base (Bot)` *api_base*

  Base URL for the Telegram Bot API. *Default: https://api.telegram.org*

- `HTTP timeout (s)` *http_timeout*

  Timeout in seconds for HTTP requests. *Default: 30*

Bot Options

- `Bot token` *bot_token*

  Token obtained from BotFather for authentication.

- `Default parse_mode` *default_parse_mode*

  Default parse mode for sending messages. *Default: HTML*

  Available modes:

  - HTML
  - Markdown
  - MarkdownV2

- `Disable link previews (default)` *default_disable_preview*

  Option to disable link previews by default. *Default: False*

- `Disable notifications (default)` *default_disable_notification*

  Option to disable message notifications by default. *Default: False*

- `Protect content (default)` *default_protect_content*

  Option to protect the content by default. *Default: False*

- `(auto) last update id` *last_update_id*

  Automatically stored ID after using tg_get_updates.

User Options (Telethon)

- `API ID (user mode)` *api_id*

  ID required for user authentication. Get from: *https://my.telegram.org*

- `API Hash (user mode)` *api_hash*

  Hash required for user authentication. Get from: *https://my.telegram.org*

- `Phone number (+CC...)` *phone_number*

  Phone number used to send login code in user mode.

- `(optional) 2FA password` *password_2fa*

  Password for two-step verification if enabled.

- `(auto) Session (StringSession)` *user_session*

  Session string saved after successful login in user mode.

- `Auto-begin login when needed` *auto_login_begin*

Automatically send login code if authentication is needed and not available. *Default: True*

Commands

- `tg_login_begin`

  Begin Telegram user login (sends code to phone).

- `tg_login_complete`

  Complete login with code and optional 2FA password.

- `tg_logout`

  Log out and clear saved session.

- `tg_mode`

  Return current mode (bot|user).

- `tg_me`

  Get authorized identity using Bot getMe or User get_me.

- `tg_send_message`

  Send text message to chat/channel.

- `tg_send_photo`

  Send photo to chat/channel.

- `tg_send_document`

  Send document/file to chat/channel.

- `tg_get_chat`

  Get chat info by id or @username.

- `tg_get_updates`

  Poll updates in bot mode, automatically store last_update_id.

- `tg_download_file`

  Download file by file_id in bot mode.

- `tg_contacts_list`

  List contacts in user mode.

- `tg_dialogs_list`

  List recent dialogs or chats in user mode.

- `tg_messages_get`

  Get recent messages from a chat in user mode.

Tuya (IoT)

The Tuya plugin integrates with Tuya's Smart Home platform, enabling seamless interactions with your smart devices via the Tuya Cloud API. This plugin provides a user-friendly interface to manage and control devices directly from your assistant.

- Provide your Tuya Cloud credentials to enable communication.
- Access and list all smart devices connected to your Tuya app account.
- Retrieve detailed information about each device, including its status and supported functions.
- Effortlessly search for devices by their names using cached data for quick access.

- Control devices by turning them on or off, toggle states, and set specific device parameters.
- Send custom commands to devices for more advanced control.
- Read sensor values and normalize them for easy interpretation.

Options

- `API base` *api_base*

  Base URL for interacting with the Tuya API. *Default: https://openapi.tuyaeu.com*

- `HTTP timeout (s)` *http_timeout*

  Requests timeout duration in seconds. *Default: 30*

- `Language` *lang*

  Language setting for API interactions. *Default: en*

Commands

Auth

- `tuya_set_keys`

  Input your Tuya Cloud credentials to enable device interactions.

- `tuya_set_uid`

  Set your Tuya App Account UID for managing device listings.

- `tuya_token_get`

  Obtain an access token for authenticated API requests.

Devices

- `tuya_devices_list`

  List all devices associated with your account UID, with options to paginate results.

- `tuya_device_get`

  Retrieve detailed information for a specified device.

- `tuya_device_status`

  Check the current status and data point values of a device.

- `tuya_device_functions`

  Discover supported functions and data point codes for a specific device.

- `tuya_find_device`

  Quickly locate devices using name-based searches from cached data.

Control

- `tuya_device_set`

  Set specific data point values for a device or use multiple settings at once.

- `tuya_device_send`

  Send a list of raw commands directly to a device for execution.

- `tuya_device_on`

  Turn a device on with an optional switch code.

- `tuya_device_off`

  Switch a device off, with automatic code detection if needed.

- `tuya_device_toggle`

  Toggle a device’s on/off state.

Sensors

- `tuya_sensors_read`

  Read normalized sensor values from your connected devices.

Vision (inline)

The plugin integrates vision capabilities across all chat modes, not just Vision mode. Once enabled, it allows the model to seamlessly switch to vision processing in the background whenever an image attachment or vision capture is detected.

Tip

When using `Vision (inline)` by utilizing a plugin in standard mode, such as `Chat` (not `Vision` mode), the `+ Vision` special checkbox will appear at the bottom of the Chat window. It will be automatically enabled any time you provide content for analysis (like an uploaded photo). When the checkbox is enabled, the vision model is used. If you wish to exit the vision model after image analysis, simply uncheck the checkbox. It will activate again automatically when the next image content for analysis is provided.

Options

- `Model` *model*

The model used to temporarily provide vision capabilities. *Default: gpt-4-vision-preview*.

- `Prompt` *prompt*

The prompt used for vision mode. It will append or replace current system prompt when using vision model.

- `Replace prompt` *replace_prompt*

Replace whole system prompt with vision prompt against appending it to the current prompt. *Default: False*

- `Tool: capturing images from camera` *cmd.camera_capture*

Allows *capture* command execution. If enabled, model will be able to capture images from camera itself. The *+ Tools* option must be enabled. *Default: False*

- `Tool: making screenshots` *cmd.make_screenshot*

Allows *screenshot* command execution. If enabled, model will be able to making screenshots itself. The *+ Tools* option must be enabled. *Default: False*

Voice Control (inline)

The plugin provides voice control command execution within a conversation.

See the `Accessibility` section for more details.

Web Search

PyGPT lets you connect model to the internet and carry out web searches in real time as you make queries.

To activate this feature, turn on the `Web Search` plugin found in the `Plugins` menu.

Web searches are provided by `DuckDuckGo`, `Google Custom Search Engine` and `Microsoft Bing` APIs and can be extended with other search engine providers.

Options

- *Provider provider*

Choose the provider. *Default: Google*

Available providers:

- Google
- Microsoft Bing

Google

To use this provider, you need an API key, which you can obtain by registering an account at:

https://developers.google.com/custom-search/v1/overview

After registering an account, create a new project and select it from the list of available projects:

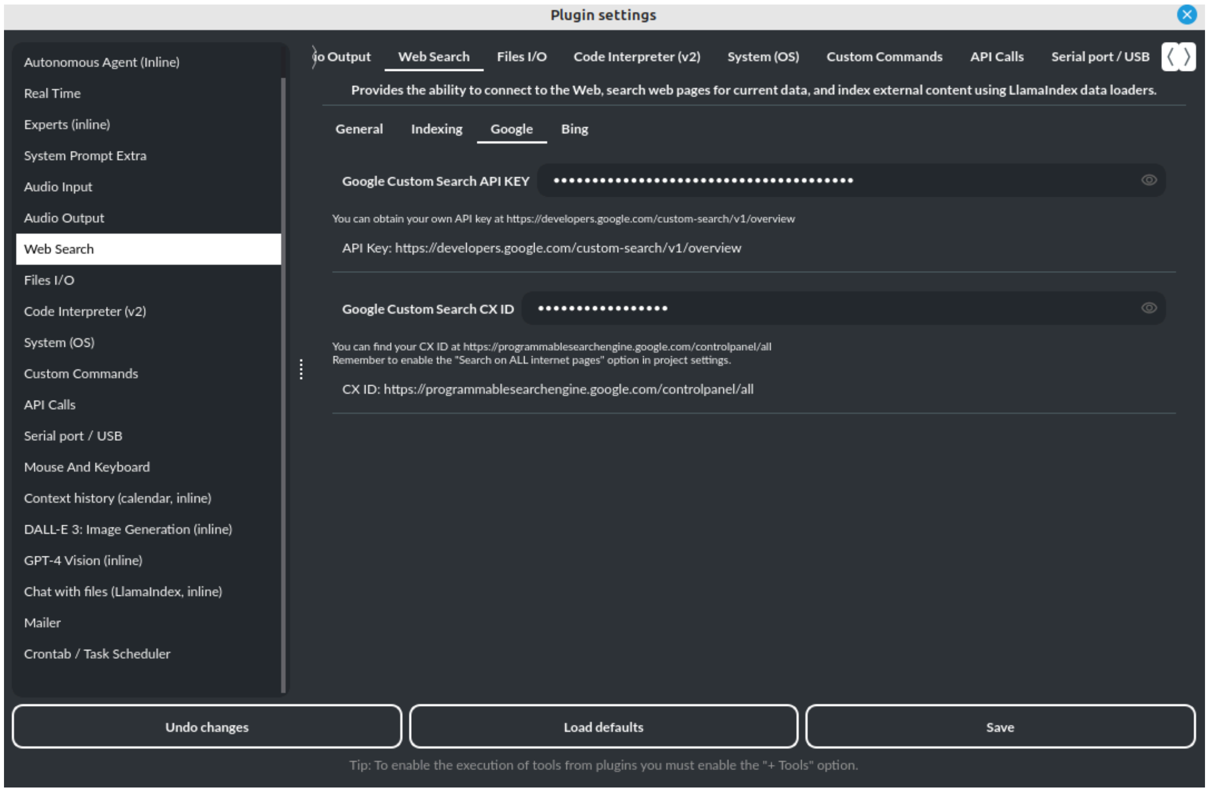
https://programmablesearchengine.google.com/controlpanel/all

After selecting your project, you need to enable the `Whole Internet Search` option in its settings. Then, copy the following two items into **PyGPT**:

- Api Key

- CX ID

These data must be configured in the appropriate fields in the `Plugins / Settings...` menu:



Provides the ability to connect to the Web, search web pages for current data, and index external content using LlamaIndex data loaders.

Google Custom Search API KEY: ••••••••••••••••••••••••••••••••••••••

You can obtain your own API key at https://developers.google.com/custom-search/v1/overview

API Key: https://developers.google.com/custom-search/v1/overview

Google Custom Search CX ID: •••••••••••••••••

You can find your CX ID at https://programmablesearchengine.google.com/controlpanel/all
Remember to enable the "Search on ALL internet pages" option in project settings.

CX ID: https://programmablesearchengine.google.com/controlpanel/all

Tip: To enable the execution of tools from plugins you must enable the "+ Tools" option.

Options

- `Google Custom Search API KEY` *google_api_key*

You can obtain your own API key at https://developers.google.com/custom-search/v1/overview

- `Google Custom Search CX ID` *google_api_cx*

You will find your CX ID at https://programmablesearchengine.google.com/controlpanel/all - remember to enable "Search on ALL internet pages" option in project settings.

Microsoft Bing

- `Bing Search API KEY` *bing_api_key*

You can obtain your own API key at https://www.microsoft.com/en-us/bing/apis/bing-web-search-api

- `Bing Search API endpoint` *bing_endpoint*

API endpoint for Bing Search API, default: https://api.bing.microsoft.com/v7.0/search

General options

- `Number of pages to search` *num_pages*

Number of max pages to search per query. *Default: 10*

- `Max content characters` *max_page_content_length*

Max characters of page content to get (0 = unlimited). *Default: 0*

- `Per-page content chunk size` *chunk_size*

Per-page content chunk size (max characters per chunk). *Default: 20000*

- `Disable SSL verify` *disable_ssl*

Disables SSL verification when crawling web pages. *Default: False*

- `Use raw content (without summarization)` *raw*

Return raw content from web search instead of summarized content. Provides more data but consumes more tokens. *Default: True*

- `Timeout` *timeout*

Connection timeout (seconds). *Default: 5*

- `User agent` *user_agent*

User agent to use when making requests. *Default: Mozilla/5.0*.

- `Max result length` *max_result_length*

Max length of the summarized or raw result (characters). *Default: 50000*

- `Max summary tokens` *summary_max_tokens*

Max tokens in output when generating summary. *Default: 1500*

- `Tool: web_search` *cmd.web_search*

Allows *web_search* command execution. If enabled, model will be able to search the Web. *Default: True*

- `Tool: web_url_open` *cmd.web_url_open*

Allows *web_url_open* command execution. If enabled, model will be able to open specified URL and summarize content. *Default: True*

- `Tool: web_url_raw` *cmd.web_url_raw*

Allows *web_url_raw* command execution. If enabled, model will be able to open specified URL and get the raw content. *Default: True*

- `Tool: web_request` *cmd.web_request*

Allows *web_request* command execution. If enabled, model will be able to send any HTTP request to specified URL or API endpoint. *Default: True*

- `Tool: web_extract_links` *cmd.web_extract_links*

Allows *web_extract_links* command execution. If enabled, model will be able to open URL and get list of all links from it. *Default: True*

- `Tool: web_extract_images` *cmd.web_extract_images*

Allows *web_extract_images* command execution. If enabled, model will be able to open URL and get list of all images from it.. *Default: True*

Advanced

- `Model used for web page summarize` *summary_model*

Model used for web page summarize. *Default: gpt-3.5-turbo-1106*

- `Summarize prompt` *prompt_summarize*

Prompt used for web search results summarize, use {query} as a placeholder for search query

- `Summarize prompt (URL open)` *prompt_summarize_url*

Prompt used for specified URL page summarize

Indexing

- `Tool: web_index` *cmd.web_index*

Allows *web_index* command execution. If enabled, model will be able to index pages and external content using LlamaIndex (persistent index). *Default: True*

- `Tool: web_index_query` *cmd.web_index_query*

Allows *web_index_query* command execution. If enabled, model will be able to quick index and query web content using LlamaIndex (in-memory index). *Default: True*

- `Auto-index all used URLs using LlamaIndex` *auto_index*

If enabled, every URL used by the model will be automatically indexed using LlamaIndex (persistent index). *Default: False*

- `Index to use` *idx*

ID of index to use for web page indexing (persistent index). *Default: base*

_

Wikipedia

The Wikipedia plugin allows for comprehensive interactions with Wikipedia, including language settings, article searching, summaries, and random article discovery. This plugin offers a variety of options to optimize your search experience.

- Set your preferred language for Wikipedia queries.
- Retrieve and check the current language setting.
- Explore a list of supported languages.
- Search for articles using keywords or get suggestions for queries.
- Obtain summaries and detailed page content.
- Discover articles by geographic location or randomly.
- Open articles directly in your web browser.

Options

- `Language` *lang*

  Default Wikipedia language. *Default: en*

- `Auto Suggest` *auto_suggest*

Enable automatic suggestions for titles. *Default: True*

- `Follow Redirects` *redirect*

  Enable following of page redirects. *Default: True*

- `Rate Limit` *rate_limit*

  Control Wikipedia API request rate limiting. *Default: True*

- `User-Agent` *user_agent*

  Custom User-Agent string for requests. *Default: pygpt-net-wikipedia-plugin/1.0 (+https://pygpt.net)*

- `Summary Sentences` *summary_sentences*

  Default number of sentences in summaries. *Default: 3*

- `Default Results Limit` *results_default*

  Number of results for searches. *Default: 10*

- `Content Maximum Characters` *content_max_chars*

  Maximum characters for page content. *Default: 5000*

- `Max List Items` *max_list_items*

  Maximum items from article lists. *Default: 50*

- `Full Content by Default` *content_full_default*

  Return full content by default. *Default: False*

Commands

Language

- `wp_set_lang`

  Set the language for Wikipedia queries.

- `wp_get_lang`

  Retrieve current language setting.

- `wp_languages`

  Get list of supported languages.

Search / Suggest

- `wp_search`

  Search for articles using keywords.

- `wp_suggest`

  Get title suggestions for queries.

Read

- `wp_summary`

  Fetch a summary of a Wikipedia article.

- `wp_page`

  Access full details of a Wikipedia article.

- `wp_section`

  Get content of a specific article section.

Discover

- `wp_random`

  Discover random Wikipedia article titles.

- `wp_geosearch`

  Find articles near specific coordinates.

Utilities

- `wp_open`

  Open articles in a web browser by title or URL.

Wolfram Alpha

Provides computational knowledge via Wolfram Alpha: short answers, full JSON pods, numeric and symbolic math (solve, derivatives, integrals), unit conversions, matrix operations, and plots rendered as images. Images are saved under `data/wolframalpha/` in the user data directory.

Options

- `API base` *api_base*

  Base API URL (default: `https://api.wolframalpha.com`). Change only if you use a proxy/gateway.

- `HTTP timeout (s)` *http_timeout*

  Request timeout in seconds.

- `Wolfram Alpha AppID` *wa_appid*

AppID used to authenticate requests. Stored as a secret. Get it from: https://developer.wolframalpha.com/portal/myapps/

- `Units` *units*

  Preferred unit system for supported endpoints: `metric` or `nonmetric`.

- `Simple background` *simple_background*

  Background for Simple API images: `white` or `transparent`.

- `Simple layout` *simple_layout*

  Layout for Simple API images, e.g., `labelbar` or `inputonly`.

- `Simple width` *simple_width*

  Target width for Simple API images (pixels). Leave empty to use the service default.

Tools (Commands)

- `Tool: wa_short` *cmd.wa_short*

  Returns a concise text answer suitable for quick facts and simple numeric results.

  Parameters: - `query` (str, required) — natural-language or math input.

- `Tool: wa_spoken` *cmd.wa_spoken*

  Returns a one-sentence, spoken-style answer.

  Parameters: - `query` (str, required)

- `Tool: wa_simple` *cmd.wa_simple*

Renders a compact result image (PNG/GIF) via the Simple API and saves it to a file.

Parameters: - `query` (str, required) - `out` (str, optional) — output path; if relative, saved under `data/wolframalpha/`. - `background` (str, optional) — `white` | `transparent`. - `layout` (str, optional) — e.g., `labelbar` | `inputonly`. - `width` (int, optional) — target image width (px).

- `Tool: wa_query` *cmd.wa_query*

  Runs a full query and returns pods as JSON (`queryresult.pods`). Can optionally download pod images.

  Parameters: - `query` (str, required) - `format` (str, optional) — e.g., `plaintext,image`. - `assumptions` (list[str], optional) — repeated `assumption` parameters. - `podstate` (str, optional) — pod state id. - `scantimeout` (int, optional), `podtimeout` (int, optional) - `maxwidth` (int, optional) — max image width. - `download_images` (bool, optional) — if true, downloads pod images. - `max_images` (int, optional) — limit for downloaded images.

- `Tool: wa_calculate` *cmd.wa_calculate*

  Evaluates or simplifies an expression. Tries the short-answer endpoint first, then falls back to a full JSON query.

  Parameters: - `expr` (str, required)

- `Tool: wa_solve` *cmd.wa_solve*

  Solves one equation or a system of equations over a chosen domain.

Parameters: - `equation` (str, optional) — single equation. - `equations` (list[str], optional) — list of equations. - `var` (str, optional) — single variable. - `variables` (list[str], optional) — variables set. - `domain` (str, optional) — `reals` | `integers` | `complexes`.

- `Tool: wa_derivative` *cmd.wa_derivative*

  Computes a derivative (with optional evaluation at a point).

  Parameters: - `expr` (str, required) - `var` (str, optional, default `x`) - `order` (int, optional, default `1`) - `at` (str, optional) — evaluation point, e.g., `x=0`.

- `Tool: wa_integral` *cmd.wa_integral*

  Computes an indefinite or definite integral.

  Parameters: - `expr` (str, required) - `var` (str, optional, default `x`) - `a` (str, optional) — lower bound (for definite integrals). - `b` (str, optional) — upper bound.

- `Tool: wa_units_convert` *cmd.wa_units_convert*

  Converts numeric values between units.

  Parameters: - `value` (str, required) — numeric value. - `from` (str, required) — source unit. - `to` (str, required) — target unit.

- `Tool: wa_matrix` *cmd.wa_matrix*

  Performs matrix operations.

  Parameters: - `op` (str, optional, default `determinant`) — `determinant` | `inverse` | `eigenvalues` | `rank`. - `matrix` (list[list], required) — e.g., `[[1,2],[3,4]]`.

- `Tool: wa_plot` *cmd.wa_plot*

  Generates a function plot as an image via the Simple API and saves it to a file.

  Parameters: - `func` (str, required) — function, e.g., `sin(x)`. - `var` (str, optional, default `x`) - `a` (str, optional) — domain start. - `b` (str, optional) — domain end. - `out` (str, optional) — output path; relative paths go under `data/wolframalpha/`.

X/Twitter

The X/Twitter plugin integrates with the X platform, allowing for comprehensive interactions such as tweeting, retweeting, liking, media uploads, and more. This plugin requires OAuth2 authentication and offers various configuration options to manage API interactions effectively.

- Retrieve user details by providing their username.
- Fetch user information using their unique ID.
- Access recent tweets from a specific user.
- Search for recent tweets using specific keywords or hashtags.
- Create a new tweet and post it on the platform.
- Remove an existing tweet from your profile.
- Reply to a specific tweet with a new comment.
- Quote a tweet while adding your own comments or thoughts.
- Like a tweet to show appreciation or support.
- Remove a like from a previously liked tweet.
- Retweet a tweet to share it with your followers.
- Undo a retweet to remove it from your profile.
- Hide a specific reply to a tweet.
- List all bookmarked tweets for easy access.
- Add a tweet to your bookmarks for later reference.
- Remove a tweet from your bookmarks.
- Upload media files such as images or videos for tweeting.

- Set alternative text for uploaded media for accessibility.

Options

- `API base` *api_base*

  Base API URL. *Default: https://api.x.com*

- `Authorize base` *authorize_base*

  Base URL for OAuth authorization. *Default: https://x.com*

- `HTTP timeout (s)` *http_timeout*

  Requests timeout in seconds. *Default: 30*

OAuth2 PKCE

- `OAuth2 Client ID` *oauth2_client_id*

  Client ID from X Developer Portal. *Secret*

- `OAuth2 Client Secret (optional)` *oauth2_client_secret*

  Only for confidential clients. *Secret*

- `Confidential client (use Basic auth)` *oauth2_confidential*

  Enable if your App is confidential. *Default: False*

- `Redirect URI` *oauth2_redirect_uri*

  Must match one of the callback URLs in your X App. *Default: http://127.0.0.1:8731/callback*

- `Scopes` *oauth2_scopes*

OAuth2 scopes for Authorization Code with PKCE. *Default: tweet.read users.read like.read like.write tweet.write bookmark.read bookmark.write tweet.moderate.write offline.access*

- `(auto) code_verifier` *oauth2_code_verifier*

  Generated by x_oauth_begin. *Secret*

- `(auto) state` *oauth2_state*

  Generated by x_oauth_begin. *Secret*

- `(auto) Access token` *oauth2_access_token*

  Stored user access token. *Secret*

- `(auto) Refresh token` *oauth2_refresh_token*

  Stored user refresh token. *Secret*

- `(auto) Expires at (unix)` *oauth2_expires_at*

  Auto-calculated expiry time.

App-only Bearer (optional for read-only)

- `App-only Bearer token (optional)` *bearer_token*

  Optional app-only bearer for read endpoints. *Secret*

Convenience cache

- `(auto) User ID` *user_id*

  Cached after x_me or oauth exchange.

- `(auto) Username` *username*

Cached after x_me or oauth exchange.

- `Auto-start OAuth when required` *oauth_auto_begin*

  Start PKCE flow automatically if needed. *Default: True*

- `Open browser automatically` *oauth_open_browser*

  Open authorize URL in default browser. *Default: True*

- `Use local server for OAuth` *oauth_local_server*

  Capture redirect using a local server. *Default: True*

- `OAuth local timeout (s)` *oauth_local_timeout*

  Time to wait for redirect with code. *Default: 180*

- `Success HTML` *oauth_success_html*

  HTML displayed on local callback success.

- `Fail HTML` *oauth_fail_html*

  HTML displayed on local callback error.

- `OAuth local port (0=auto)` *oauth_local_port*

  Local HTTP port for callback. *Default: 8731*

- `Allow fallback port if busy` *oauth_allow_port_fallback*

  Use a free port if the preferred port is busy. *Default: True*

Commands

Auth

- `x_oauth_begin`

  Begin OAuth2 PKCE flow.

- `x_oauth_exchange`

  Exchange authorization code for tokens.

- `x_oauth_refresh`

  Refresh access token using refresh_token.

Users

- `x_me`

  Get authorized user information.

- `x_user_by_username`

  Lookup user by username.

- `x_user_by_id`

  Lookup user by ID.

Timelines / Search

- `x_user_tweets`

  Retrieve user Tweet timeline.

- `x_search_recent`

  Perform recent search within the last 7 days.

Tweet CRUD

- `x_tweet_create`

  Create a new Tweet/Post.

- `x_tweet_delete`

  Delete a Tweet by ID.

- `x_tweet_reply`

  Reply to a Tweet.

- `x_tweet_quote`

  Quote a Tweet.

Actions

- `x_like`

  Like a Tweet.

- `x_unlike`

  Unlike a Tweet.

- `x_retweet`

  Retweet a Tweet.

- `x_unretweet`

  Undo a retweet.

- `x_hide_reply`

  Hide or unhide a reply to your Tweet.

Bookmarks

- `x_bookmarks_list`

  List bookmarks.

- `x_bookmark_add`

  Add a bookmark.

- `x_bookmark_remove`

  Remove a bookmark.

Media

- `x_upload_media`

  Upload media and return media_id.

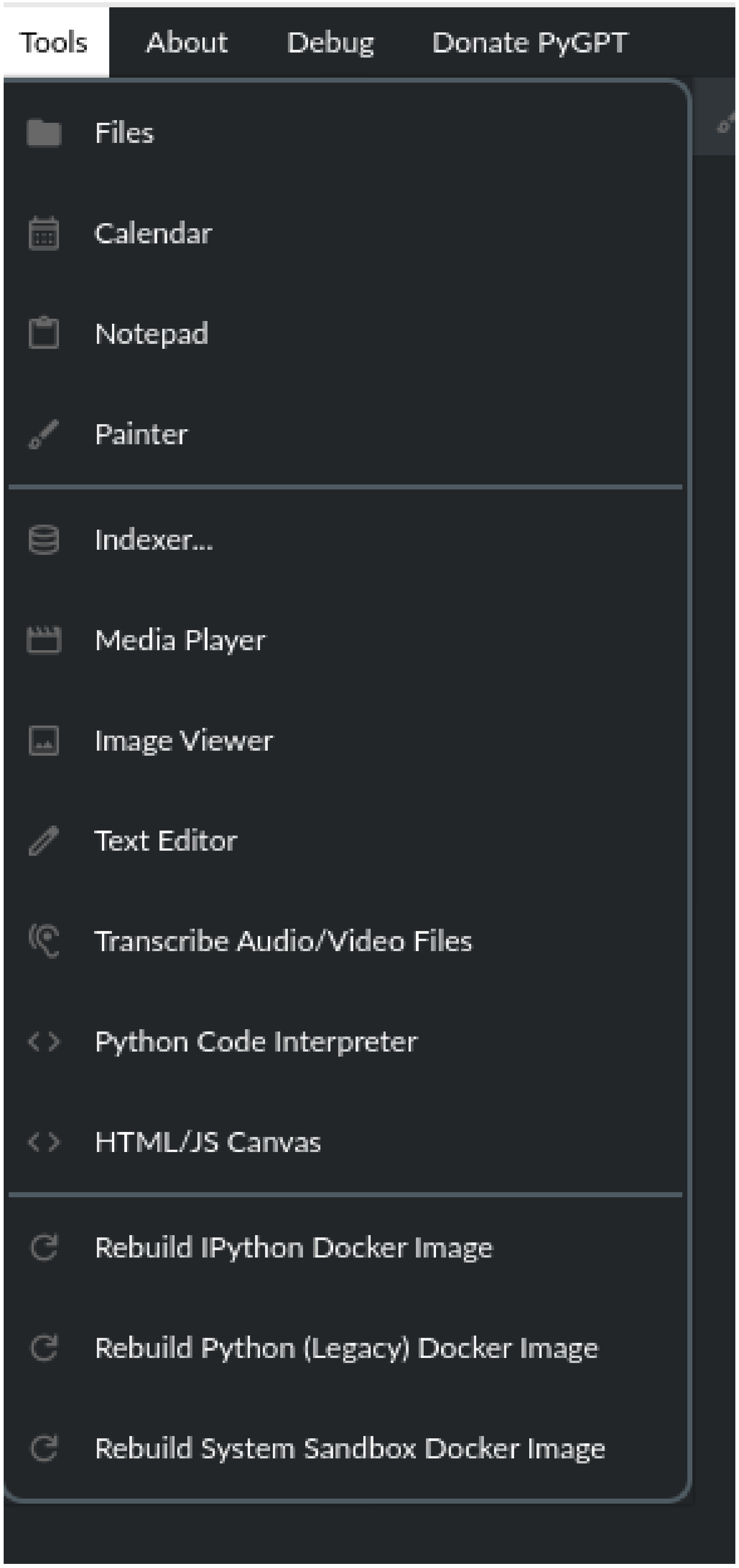
- `x_media_set_alt_text`

  Attach alt text to uploaded media.

Tools

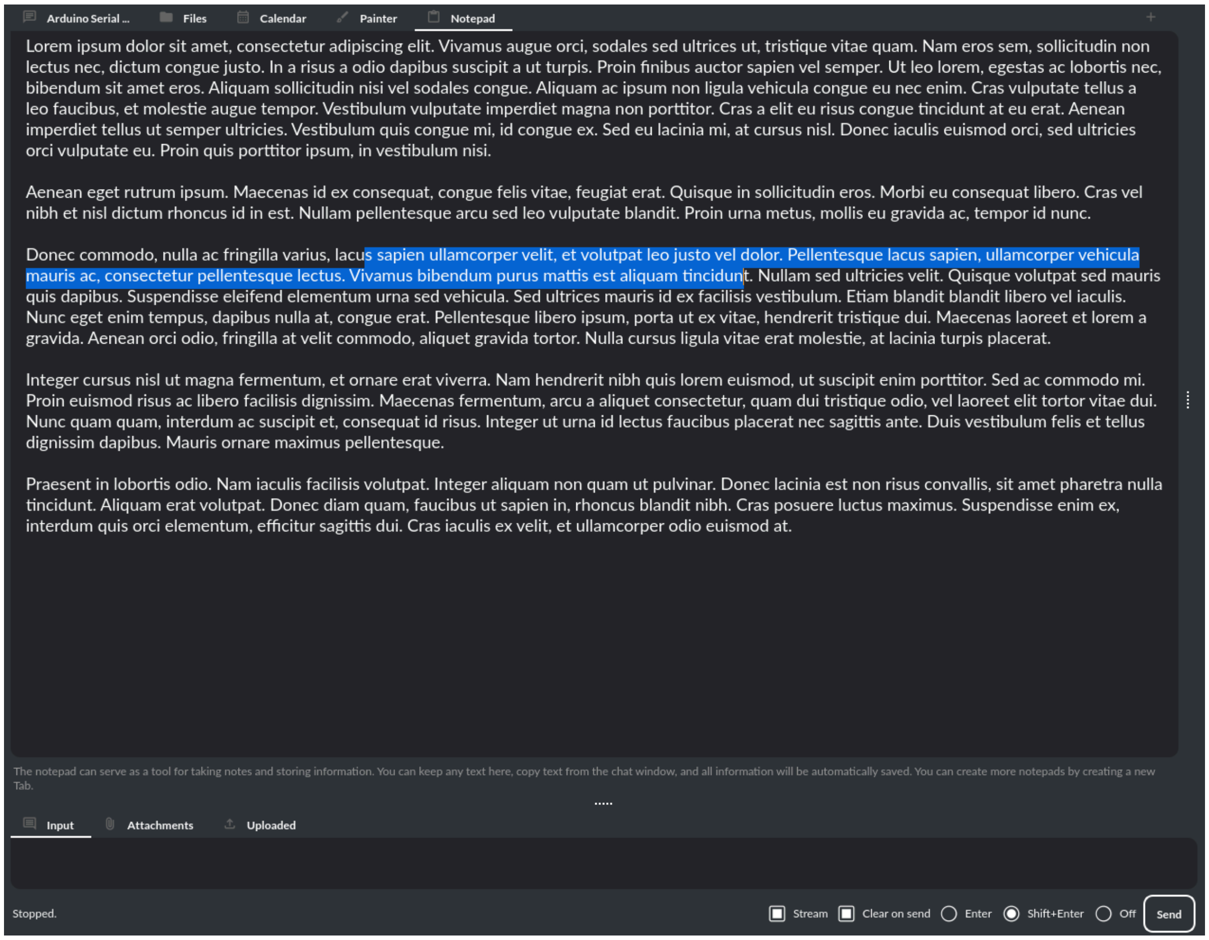
PyGPT features several useful tools, including:

- Notepad
- Calendar
- Painter
- Indexer
- Media Player
- Image Viewer
- Text Eeditor
- Transcribe Audio/Video Files
- Python Code Interpreter
- HTML/JS Canvas (built-in HTML renderer)
- Translator
- Web Browser (Chromium)
- Agents Builder (beta)


Tools
About
Debug
Donate PyGPT
Files
Calendar
Notepad
Painter
Indexer...
Media Player
Image Viewer
Text Editor
Transcribe Audio/Video Files
Python Code Interpreter
HTML/JS Canvas
Rebuild IPython Docker Image
Rebuild Python (Legacy) Docker Image
Rebuild System Sandbox Docker Image

Notepad

The application has a built-in notepad, divided into several tabs. This can be useful for storing information in a convenient way, without the need to open an external text editor. The content of the notepad is automatically saved whenever the content changes.



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus augue orci, sodales sed ultrices ut, tristique vitae quam. Nam eros sem, sollicitudin non lectus nec, dictum congue justo. In a risus a odio dapibus suscipit a ut turpis. Proin finibus auctor sapien vel semper. Ut leo lorem, egestas ac lobortis nec, bibendum sit amet eros. Aliquam sollicitudin nisi vel sodales congue. Aliquam ac ipsum non ligula vehicula congue eu nec enim. Cras vulputate tellus a leo faucibus, et molestie augue tempor. Vestibulum vulputate imperdiet magna non porttitor. Cras a elit eu risus congue tincidunt at eu erat. Aenean imperdiet tellus ut semper ultricies. Vestibulum quis congue mi, id congue ex. Sed eu lacinia mi, at cursus nisl. Donec iaculis euismod orci, sed ultricies orci vulputate eu. Proin quis porttitor ipsum, in vestibulum nisi.

Aenean eget rutrum ipsum. Maecenas id ex consequat, congue felis vitae, feugiat erat. Quisque in sollicitudin eros. Morbi eu consequat libero. Cras vel nibh et nisl dictum rhoncus id in est. Nullam pellentesque arcu sed leo vulputate blandit. Proin urna metus, mollis eu gravida ac, tempor id nunc.

Donec commodo, nulla ac fringilla varius, lacus sapien ullamcorper velit, et volutpat leo justo vel dolor. Pellentesque lacus sapien, ullamcorper vehicula mauris ac, consectetur pellentesque lectus. Vivamus bibendum purus mattis est aliquam tincidunt. Nullam sed ultricies velit. Quisque volutpat sed mauris quis dapibus. Suspendisse eleifend elementum urna sed vehicula. Sed ultrices mauris id ex facilisis vestibulum. Etiam blandit blandit libero vel iaculis. Nunc eget enim tempus, dapibus nulla at, congue erat. Pellentesque libero ipsum, porta ut ex vitae, hendrerit tristique dui. Maecenas laoreet et lorem a gravida. Aenean orci odio, fringilla at velit commodo, aliquet gravida tortor. Nulla cursus ligula vitae erat molestie, at lacinia turpis placerat.

Integer cursus nisl ut magna fermentum, et ornare erat viverra. Nam hendrerit nibh quis lorem euismod, ut suscipit enim porttitor. Sed ac commodo mi. Proin euismod risus ac libero facilisis dignissim. Maecenas fermentum, arcu a aliquet consectetur, quam dui tristique odio, vel laoreet elit tortor vitae dui. Nunc quam quam, interdum ac suscipit et, consequat id risus. Integer ut urna id lectus faucibus placerat nec sagittis ante. Duis vestibulum felis et tellus dignissim dapibus. Mauris ornare maximus pellentesque.

Praesent in lobortis odio. Nam iaculis facilisis volutpat. Integer aliquam non quam ut pulvinar. Donec lacinia est non risus convallis, sit amet pharetra nulla tincidunt. Aliquam erat volutpat. Donec diam quam, faucibus ut sapien in, rhoncus blandit nibh. Cras posuere luctus maximus. Suspendisse enim ex, interdum quis orci elementum, efficitur sagittis dui. Cras iaculis ex velit, et ullamcorper odio euismod at.

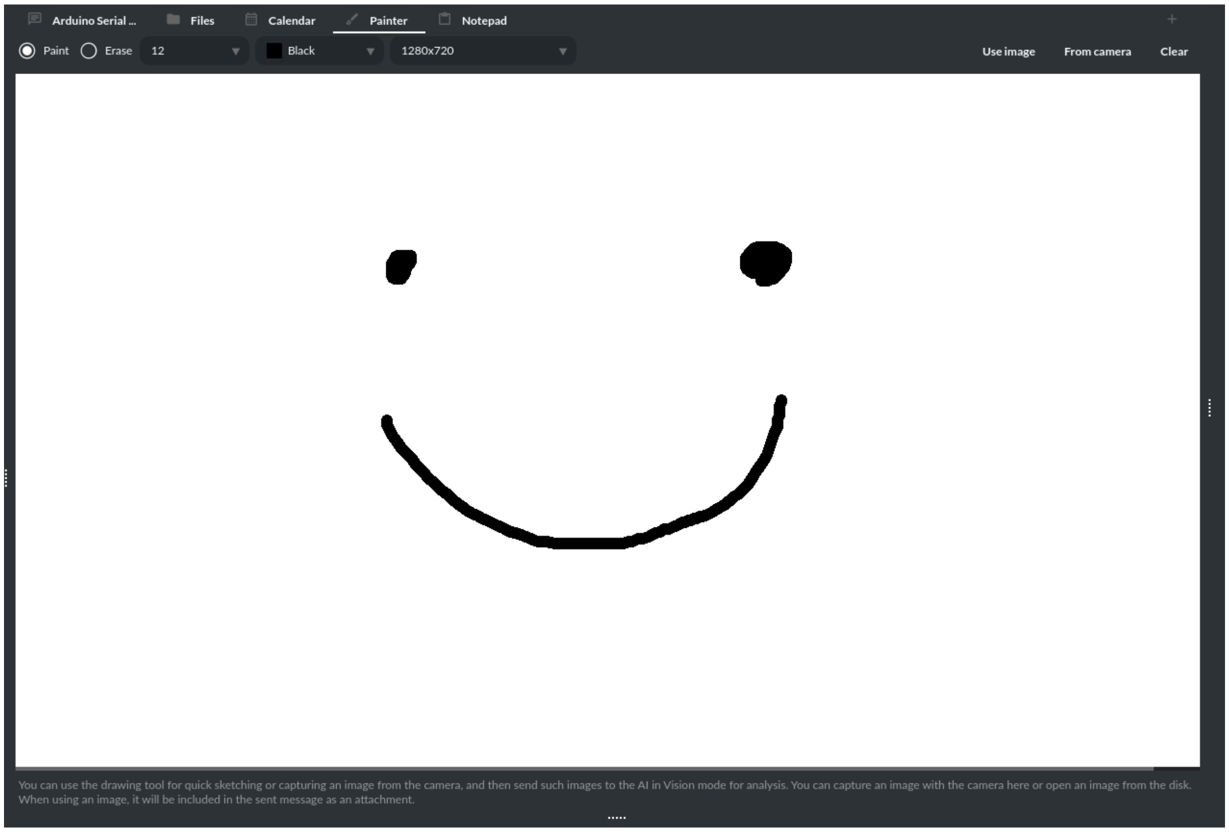
The notepad can serve as a tool for taking notes and storing information. You can keep any text here, copy text from the chat window, and all information will be automatically saved. You can create more notepads by creating a new Tab.

Stopped.

[✓ Stream] [✓ Clear on send] Enter [✓ Shift+Enter] Off

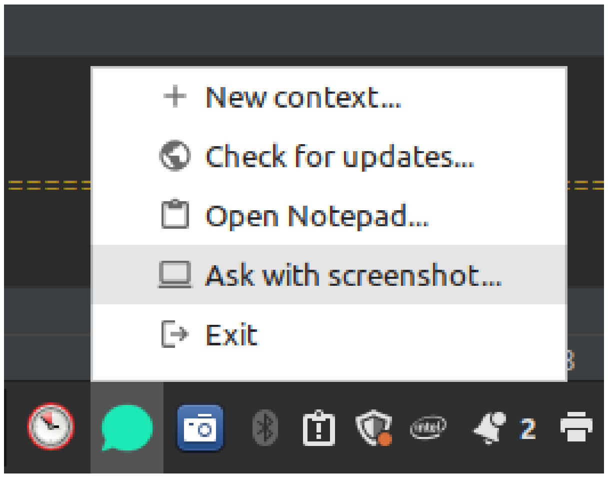
Painter

Using the `Painter` tool, you can create quick sketches and submit them to the model for analysis. You can also edit open or camera-captured images, for example, by adding elements like arrows or outlines to objects. Additionally, you can capture screenshots from the system - the captured image is placed in the drawing tool and attached to the query being sent.



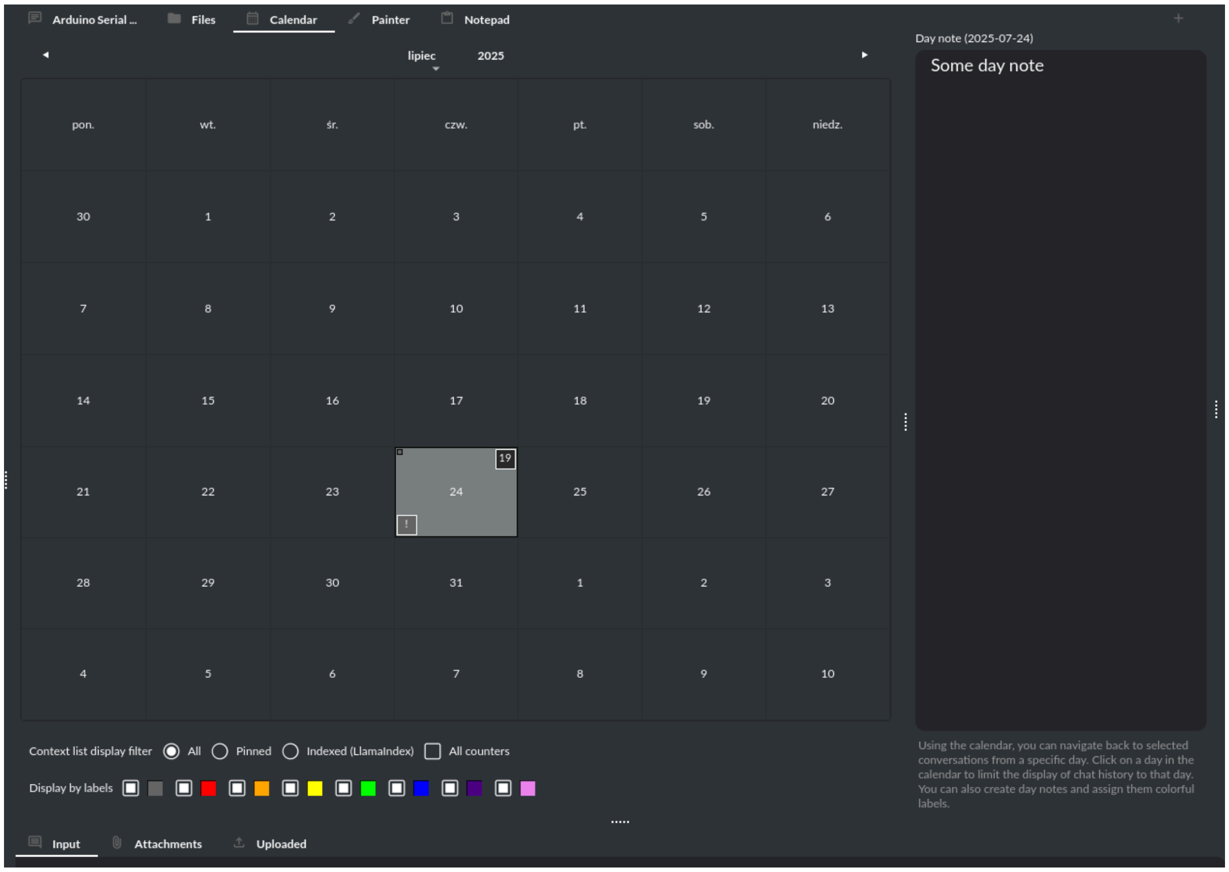
You can use the drawing tool for quick sketching or capturing an image from the camera, and then send such images to the AI in Vision mode for analysis. You can capture an image with the camera here or open an image from the disk. When using an image, it will be included in the sent message as an attachment.

To quick capture the screenshot click on the option `Ask with screenshot` in tray-icon dropdown:



Calendar

Using the calendar, you can go back to selected conversations from a specific day and add daily notes. After adding a note, it will be marked on the list, and you can change the color of its label by right-clicking and selecting `Set label color` option. By clicking on a particular day of the week, conversations from that day will be displayed.



pon.	wt.	śr.	czw.	pt.	sob.	niedz.
30	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24 19 !	25	26	27
28	29	30	31	1	2	3
4	5	6	7	8	9	10

Context list display filter [✓ All] Pinned Indexed (LlamaIndex) All counters

Display by labels ⬛ 🟥 🟧 🟨 🟩 🟦 🟪 🟪

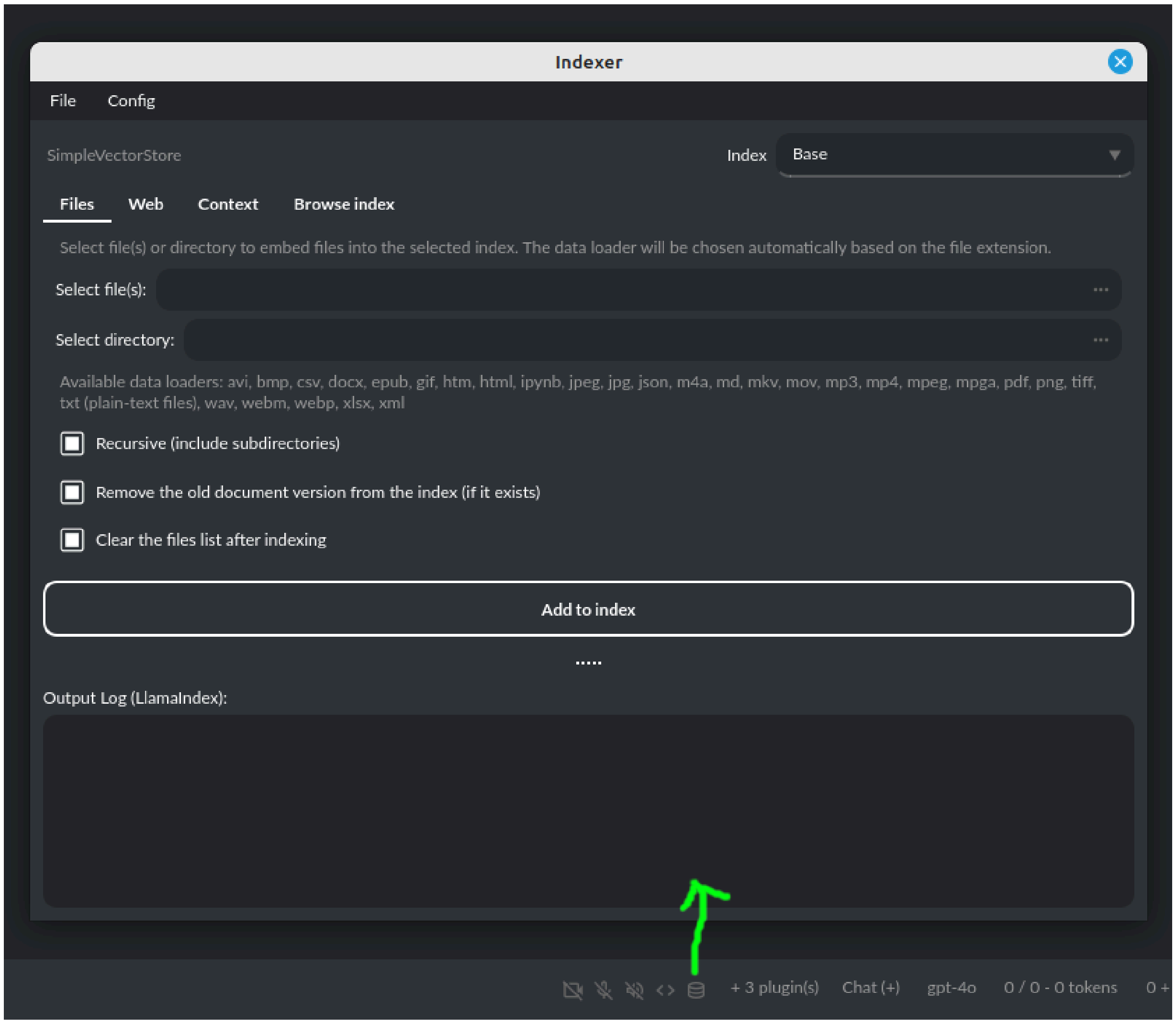
Day note (2025-07-24)

Some day note

Using the calendar, you can navigate back to selected conversations from a specific day. Click on a day in the calendar to limit the display of chat history to that day. You can also create day notes and assign them colorful labels.

Indexer

This tool allows indexing of local files or directories and external web content to a vector database, which can then be used with the `Chat with Files` mode. Using this tool, you can manage local indexes and add new data with built-in `LlamaIndex` integration.



Indexer

File Config

SimpleVectorStore

Index: Base

Files | Web | Context | Browse index

Select file(s) or directory to embed files into the selected index. The data loader will be chosen automatically based on the file extension.

Select file(s):

Select directory:

Available data loaders: avi, bmp, csv, docx, epub, gif, htm, html, ipynb, jpeg, jpg, json, m4a, md, mkv, mov, mp3, mp4, mpeg, mpga, pdf, png, tiff, txt (plain-text files), wav, webm, webp, xlsx, xml

☐ Recursive (include subdirectories)

☐ Remove the old document version from the index (if it exists)

☐ Clear the files list after indexing

Add to index

Output Log (LlamaIndex):

+ 3 plugin(s) Chat (+) gpt-4o 0 / 0 - 0 tokens 0 +

Media Player

A simple video/audio player that allows you to play video files directly from within the app.

Image Viewer

A simple image browser that lets you preview images directly within the app.

Text Editor

A simple text editor that enables you to edit text files directly within the app.

Transcribe Audio/Video Files

An audio transcription tool with which you can prepare a transcript from a video or audio file. It will use a speech recognition plugin to generate the text from the file.

Python Code Interpreter

This tool allows you to run Python code directly from within the app. It is integrated with the `Code Interpreter` plugin, ensuring that code generated by the model is automatically available from the interpreter. In the plugin settings, you can enable the execution of code in a Docker environment.

Important

Executing Python code using IPython in compiled versions requires an enabled sandbox (Docker container). You can connect the Docker container via `Plugins -> Settings`.

HTML/JS Canvas

Allows to render HTML/JS code in HTML Canvas (built-in renderer based on Chromium). To use it, just ask the model to render the HTML/JS code in built-in browser (HTML Canvas). Tool is integrated with the `Code Interpreter` plugin.

Translator

Enables translation between multiple languages using an AI model.

Web Browser (Chromium)

A built-in web browser based on Chromium, allowing you to open webpages directly within the app.

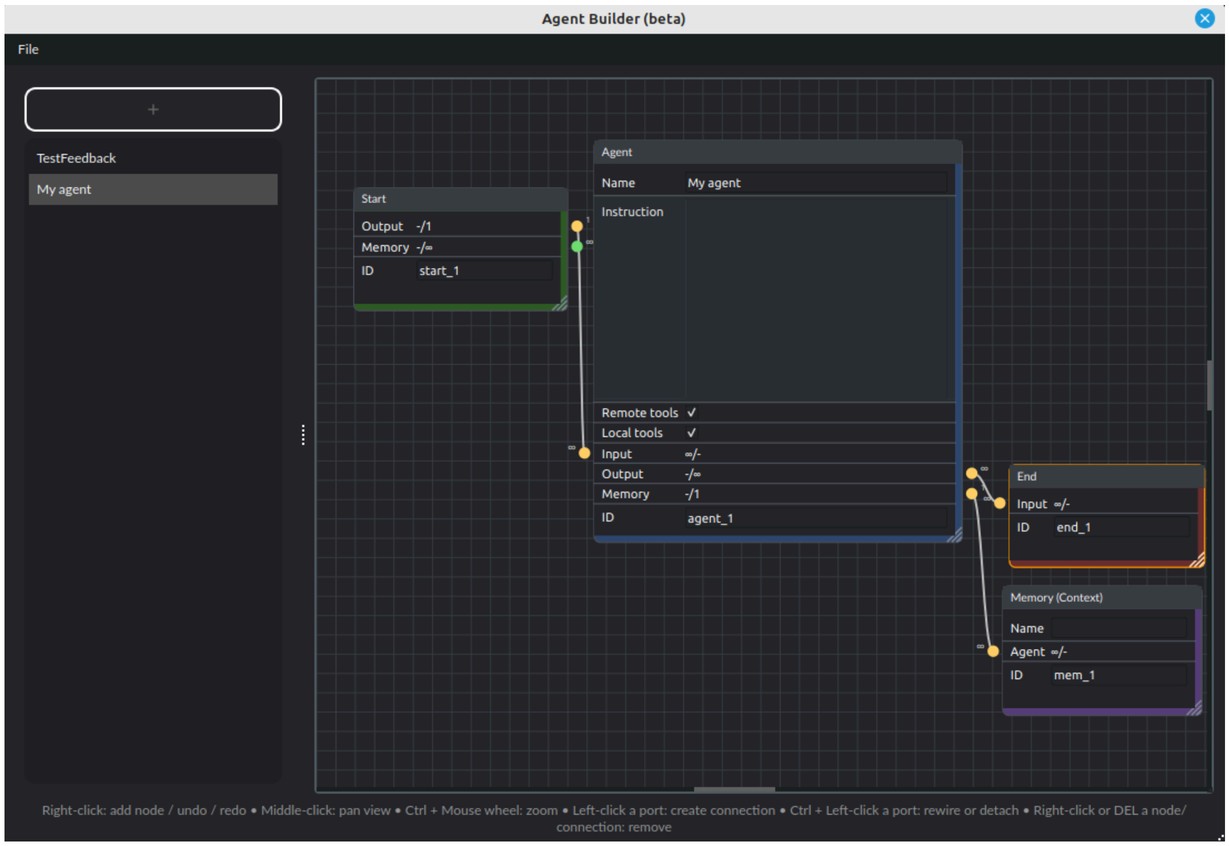
Warning

SECURITY NOTICE: For your protection, avoid using the built-in browser for sensitive or critical tasks. It is intended for basic use only.

Agents Builder (beta)

To launch the Agent Editor, navigate to:

Tools -> Agents Builder



Right-click: add node / undo / redo • Middle-click: pan view • Ctrl + Mouse wheel: zoom • Left-click a port: create connection • Ctrl + Left-click a port: rewire or detach • Right-click or DEL a node/connection: remove

This tool allows you to create workflows for agents using a node editor, without writing any code. You can add a new agent type, and it will appear in the list of presets.

To add a new element, right-click on the editor grid and select `Add` to insert a new node.

Types of Nodes:

- **Start**: The starting point for agents (user input).
- **Agent**: A single agent with customizable default parameters, such as system instructions and tool usage. These settings can be overridden in the preset.
- **Memory**: Shared memory between agents (shared Context).
- **End**: The endpoint, returning control to the user.

Agents with connected shared memory share it among themselves. Agents without shared memory only receive the latest output from the previous agent.

The first agent in the sequence always receives the full context passed by the user.

Connecting agents and memory is done using node connections via slots. To connect slots, simply drag from the input port to the output port (Ctrl + mouse button removes a connection).

Node Editor Navigation:

- **Right-click**: Add node, undo, redo, clear
- **Middle-click + drag**: Pan view
- **Ctrl + Mouse wheel**: Zoom
- **Left-click a port**: Create connection
- **Ctrl + Left-click a port**: Rewire or detach connection
- **Right-click or DELETE a node/connection**: Remove node/connection

Tip

Enable agent debugging in `Settings -> Debug -> Log Agents usage to console` to log the full workflow to the console.

Agents built using this tool are compatible with both OpenAI Agents and LlamaIndex.

Notes:

Routing and system instruction: for every agent that has more than one connection leading to the next agent, a routing instruction is automatically injected just before your system prompt:

```
You are a routing-capable agent in a multi-agent flow.
Your id is: <current_id>, name: <agent_name>.
You MUST respond ONLY with a single JSON object and nothing
```

```
else.
Schema:
{
  "route": "<ID of the next agent from allowed_routes OR the
string 'end'>",
  "content": "<final response text for the user (or tool
result)>"
}
Rules:
- allowed_routes: [<allowed>]
- If you want to finish the flow, set route to "end".
- content must contain the user-facing answer (you may include
structured data as JSON or Markdown inside content).
- Do NOT add any commentary outside of the JSON. No leading or
trailing text.
- If using tools, still return the final JSON with tool results
summarized in content.
- Human-friendly route names: <names>
- Human-friendly route roles (optional): <roles>

<here begins your system instruction>
```

INFO: Agents Builder is in beta.

Functions, commands and tools

Note

Remember to enable the `+ Tools` checkbox to enable execution of tools and commands from plugins.

From version `2.2.20` PyGPT uses native API function calls by default. You can go back to internal syntax (described below) by switching off option `Config -> Settings -> Prompts -> Use native API function calls`. You must also enable `Tool calls` checkbox in model advanced settings to use native function calls with the specified model.

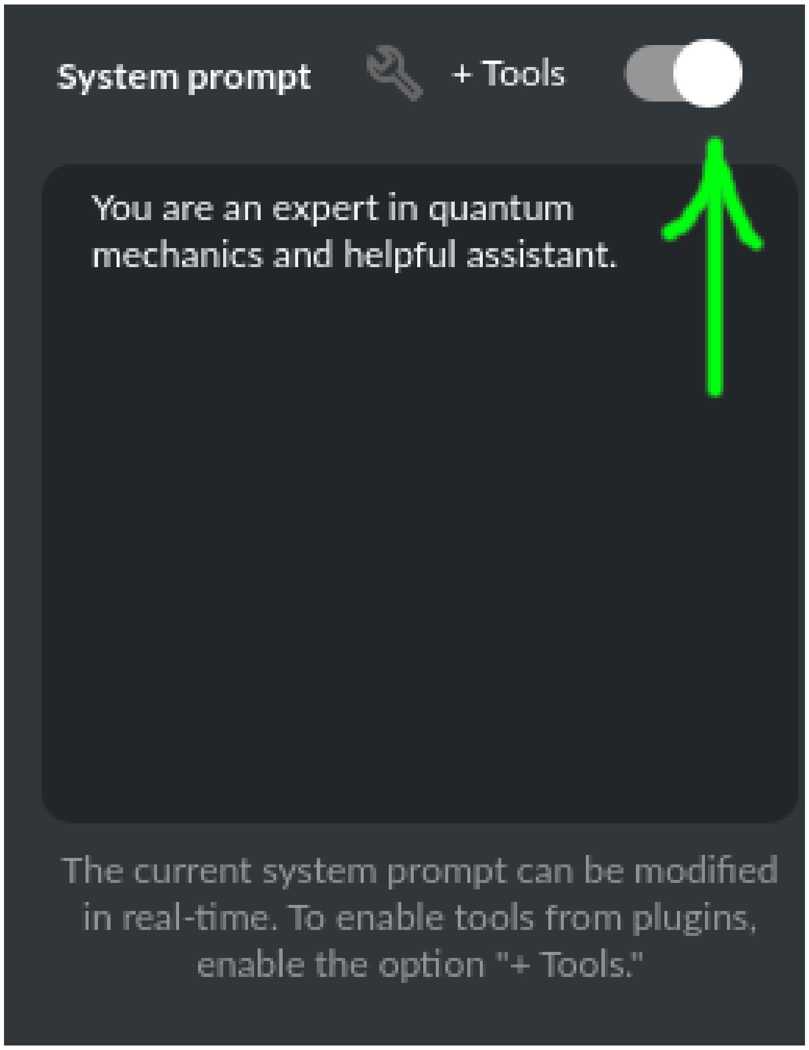
In background, **PyGPT** uses an internal syntax to define commands and their parameters, which can then be used by the model and executed on the application side or even directly in the system. This syntax looks as follows (example command below):

```
<tool>{"cmd": "send_email", "params": {"quote": "Why don't
skeletons fight each other? They don't have the guts!"}}</tool>
```

It is a JSON object wrapped between `<tool>` tags. The application extracts the JSON object from such formatted text and executes the appropriate function based on the provided parameters and command name. Many of these types of commands are defined in plugins (e.g., those used for file operations or internet searches). You can also define your own commands using the `Custom Commands` plugin, or simply by creating your own plugin and adding it to the application.

Tip

The `+ Tools` option checkbox must be enabled to allow the execution of commands from plugins. Disable the option if you do not want to use commands, to prevent additional token usage (as the command execution system prompt consumes additional tokens and may slow down local models).



You are an expert in quantum mechanics and helpful assistant.

The current system prompt can be modified in real-time. To enable tools from plugins, enable the option "+ Tools."

When native API function calls are disabled, a special system prompt responsible for invoking commands is added to the main system prompt if the `+ Tools` option is active.

However, there is an additional possibility to define your own commands and execute them with the help of model. These are functions / tools - defined on the API side and described using JSON objects. You can find a complete guide on how to define functions here:

https://platform.openai.com/docs/guides/function-calling

https://cookbook.openai.com/examples/how_to_call_functions_with_chat_models

PyGPT offers compatibility of these functions with commands (tools) used in the application. All you need to do is define the appropriate functions using the correct JSON schema, and PyGPT will do the rest, translating such syntax on the fly into its own internal format.

Local functions and tools from plugins are available in all modes, except `Assistants`.

To enable local functions for `Assistants` mode (in this mode remote tools are used by default), create a new Assistant, open the Preset edit dialog and import tools from plugins or add a new function using *+ Function* button e.g. with the following content:

Name: `send_email`

Description: `Sends a quote using email`

Params (JSON):

```
{
        "type": "object",
        "properties": {
            "quote": {
                "type": "string",
                "description": "A generated funny quote"
            }
        },
        "required": [
            "quote"
        ]
}
```

Then, in the `Custom Commands` plugin, create a new command with the same name and the same parameters:

Command name: `send_email`

Instruction/prompt: `send mail`

Params list: `quote`

Command to execute: `echo "OK. Email sent: {quote}"`

At next, enable the `+ Tools` option and enable the plugin.

Ask a model:

```
Create a funny quote and email it
```

In response you will receive prepared command, like this:

```
<tool>{"cmd": "send_email", "params": {"quote": "Why do we tell
actors to 'break a leg?' Because every play has a cast!"}}
</tool>
```

After receiving this, PyGPT will execute the system `echo` command with params given from `params` field and replacing `{quote}` placeholder with `quote` param value.

As a result, response like this will be sent to the model:

```
[{"request": {"cmd": "send_email"}, "result": "OK. Email sent:
Why do we tell actors to 'break a leg?' Because every play has
a cast!"}]
```

With this flow you can use both forms - API provider JSON schema and PyGPT schema - to define and execute commands and functions in the application. They will cooperate with each other and you can use them interchangeably.

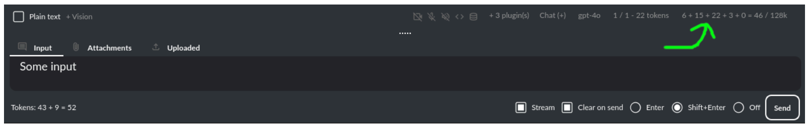
Token usage

Input tokens

The application features a token calculator. It attempts to forecast the number of tokens that a particular query will consume and displays this estimate in real time. This gives you improved control over your token usage. The app provides detailed information about the tokens used for the user’s prompt, the system prompt, any additional data, and those used within the context (the memory of previous entries).

Important

Remember that these are only approximate calculations and do not include, for example, the number of tokens consumed by some plugins. You can find the exact number of tokens used on provider’s website.



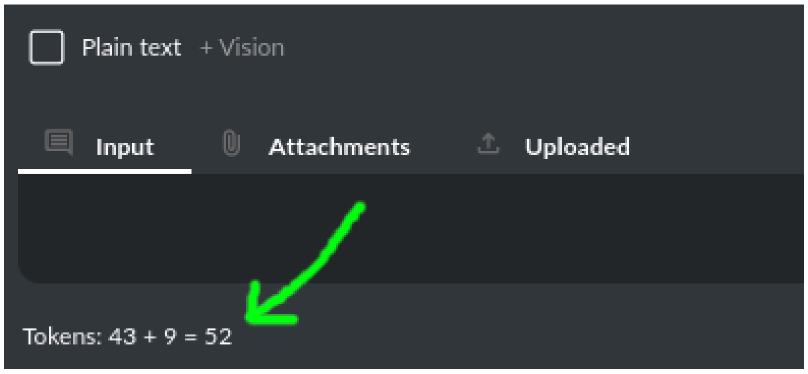
Some input

Tokens: 43 + 9 = 52

[✓ Stream] [✓ Clear on send] Enter [✓ Shift+Enter] Off

Total tokens

After receiving a response from the model, the application displays the actual total number of tokens used for the query (received from the API).



Tokens: 43 + 9 = 52

Accessibility

Since version `2.2.8`, PyGPT has added beta support for disabled people and voice control. This may be very useful for blind people.

In the `Config / Accessibility` menu, you can turn on accessibility features such as:

- activating voice control
- translating actions and events on the screen with audio speech
- setting up keyboard shortcuts for actions.

Using voice control

Voice control can be turned on in two ways: globally, through settings in `Config -> Accessibility`, and by using the `Voice control (inline)` plugin. Both options let you use the same voice commands, but they work a bit differently - the global option allows you to run commands outside of a conversation, anywhere, while the plugin option lets you execute commands directly during a conversation – allowing you to interact with the model and execute commands at the same time, within the conversation.

In the plugin (inline) option, you can also turn on a special trigger word that will be needed for content to be recognized as a voice command. You can set this up by going to `Plugins -> Settings -> Voice Control (inline)`:

```
Magic prefix for voice commands
```

Tip

When the voice control is enabled via a plugin, simply provide commands while providing the content of the conversation by using the standard *Microphone* button.

Enabling voice control globally

Turn on the voice control option in `Config / Accessibility`:

`Enable voice control (using microphone)`

Once you enable this option, an `Voice Control` button will appear at the bottom right corner of the window. When you click on this button, the microphone will start listening; clicking it again stops listening and starts recognizing the voice command you said. You can cancel voice recording at any time with the `ESC` key. You can also set a keyboard shortcut to turn voice recording on/off.

Voice command recognition works based on a model, so you don’t have to worry about saying things perfectly.

Here’s a list of commands you can ask for by voice:

- Get the current application status
- Exit the application
- Enable audio output
- Disable audio output
- Enable audio input
- Disable audio input
- Add a memo to the calendar
- Clear memos from calendar
- Read the calendar memos
- Enable the camera
- Disable the camera
- Capture image from camera
- Create a new context
- Go to the previous context
- Go to the next context
- Go to the latest context

- Focus on the input
- Send the input
- Clear the input
- Get current conversation info
- Get available commands list
- Stop executing current action
- Clear the attachments
- Read the last conversation entry
- Read the whole conversation
- Rename current context
- Search for a conversation
- Clear the search results
- Send the message to input
- Append message to current input without sending it
- Switch to chat mode
- Switch to chat with files (llama-index) mode
- Switch to the next mode
- Switch to the previous mode
- Switch to the next model
- Switch to the previous model
- Add note to notepad
- Clear notepad contents
- Read current notepad contents
- Switch to the next preset
- Switch to the previous preset
- Switch to the chat tab
- Switch to the calendar tab
- Switch to the draw (painter) tab
- Switch to the files tab
- Switch to the notepad tab
- Switch to the next tab
- Switch to the previous tab
- Start listening for voice input
- Stop listening for voice input
- Toggle listening for voice input

More commands coming soon.

Just ask for an action that matches one of the descriptions above. These descriptions are also known to the model, and relevant commands are assigned to them. When you voice a command that fits one of those patterns, the model will trigger the appropriate action.

For convenience, you can enable a short sound to play when voice recording starts and stops. To do this, turn on the option:

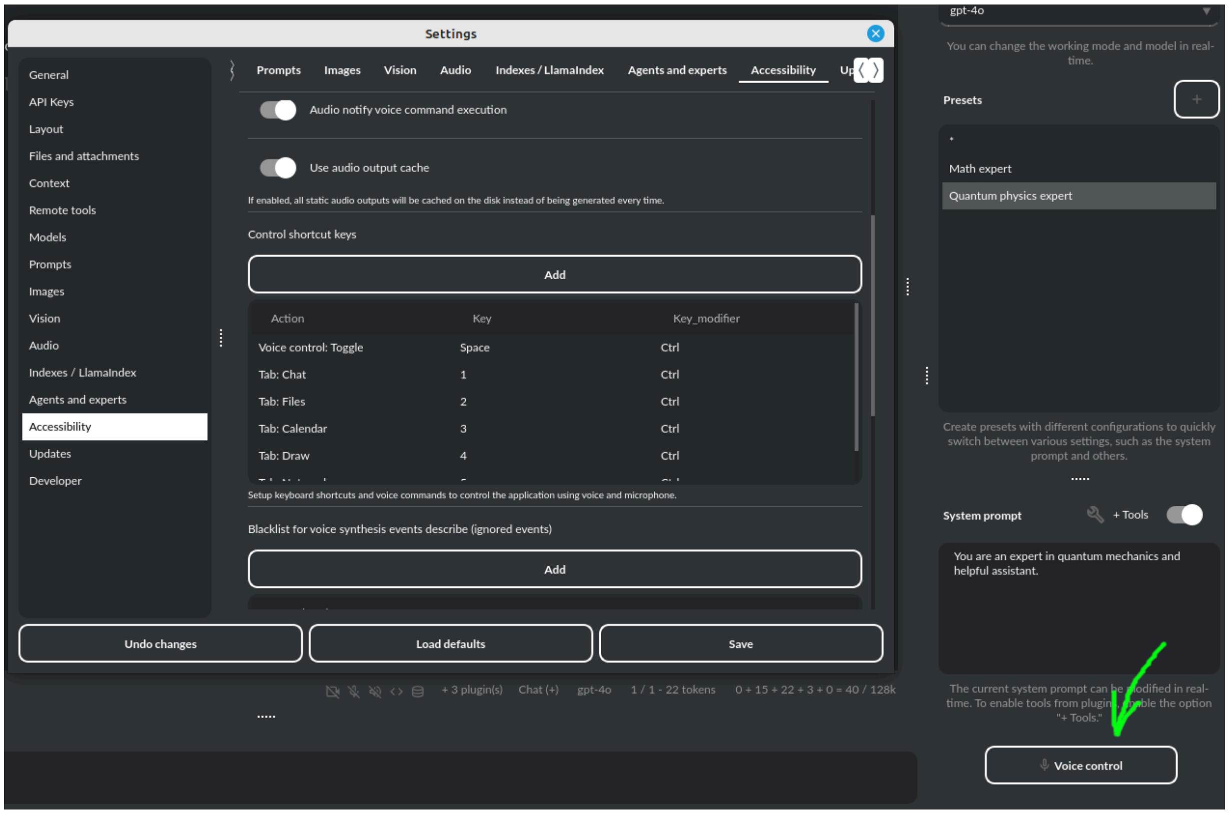
```
Audio notify microphone listening start/stop
```

To enable a sound notification when a voice command is recognized and command execution begins, turn on the option:

```
Audio notify voice command execution
```

For voice translation of on-screen events and information about completed commands via speech synthesis, you can turn on the option:

```
Use voice synthesis to describe events on the screen.
```



Audio notify voice command execution

Use audio output cache

If enabled, all static audio outputs will be cached on the disk instead of being generated every time.

Control shortcut keys

Action	Key	Key_modifier
Voice control: Toggle	Space	Ctrl
Tab: Chat	1	Ctrl
Tab: Files	2	Ctrl
Tab: Calendar	3	Ctrl
Tab: Draw	4	Ctrl

Setup keyboard shortcuts and voice commands to control the application using voice and microphone.

Blacklist for voice synthesis events describe (ignored events)

You can change the working mode and model in real-time.

Create presets with different configurations to quickly switch between various settings, such as the system prompt and others.

You are an expert in quantum mechanics and helpful assistant.

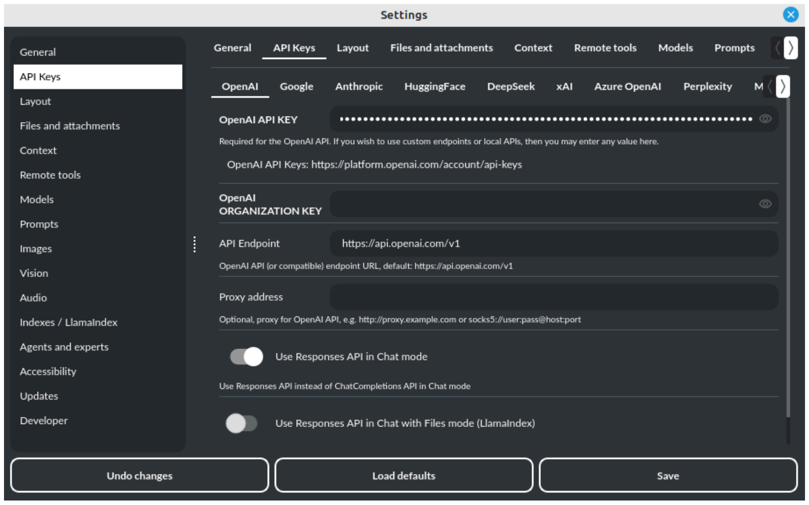
The current system prompt can be modified in real-time. To enable tools from plugins, enable the option "+ Tools."

Configuration

Settings

The following basic options can be modified directly within the application:

```
Config -> Settings...
```



OpenAI API KEY ••

Required for the OpenAI API. If you wish to use custom endpoints or local APIs, then you may enter any value here.

OpenAI API Keys: https://platform.openai.com/account/api-keys

OpenAI ORGANIZATION KEY

API Endpoint https://api.openai.com/v1

OpenAI API (or compatible) endpoint URL, default: https://api.openai.com/v1

Proxy address

Optional, proxy for OpenAI API, e.g. http://proxy.example.com or socks5://user:pass@host:port

Use Responses API in Chat mode

Use Responses API instead of ChatCompletions API in Chat mode

Use Responses API in Chat with Files mode (LlamaIndex)

General

- `Show tray icon`: Show/hide tray icon. Tray icon provides additional features like "Ask with screenshot" or "Open notepad". Restart of the application is required for this option to take effect. Default: True.
- `Minimize to tray on exit`: Minimize to tray icon on exit. Tray icon enabled is required for this option to work. Default: False.
- `Render engine`: chat output render engine: *WebEngine / Chromium* - for full HTML/CSS and *Legacy (markdown)* for legacy, simple markdown CSS output. Default: WebEngine / Chromium.

- `OpenGL hardware acceleration`: enables hardware acceleration in *WebEngine / Chromium* renderer. Default: False.
- `Use proxy`: Enable this option to use a proxy for connections to APIs. Default: False.
- `Proxy address`: Proxy address to be used for connection in API SDKs; supports HTTP/SOCKS, e.g. http://proxy.example.com or socks5://user:pass@host:port
- `Application environment (os.environ)`: Additional environment vars to set on application start.
- `Memory Limit`: Renderer memory limit; set to 0 to disable. If > 0, the app will try to free memory after the limit is reached. Accepted formats: 3.5GB, 2GB, 2048MB, 1_000_000. Minimum: 2GB.

API Keys

- `OpenAI API KEY`: Required for the OpenAI API. If you wish to use custom endpoints or local APIs, then you may enter any value here.
- `OpenAI ORGANIZATION KEY`: The organization's API key, which is optional for use within the application.
- `API Endpoint`: OpenAI API endpoint URL, default: https://api.openai.com/v1.
- `Anthropic API KEY`: Required for the Anthropic API and Claude models.
- `Deepseek API KEY`: Required for the Deepseek API.
- `Google API KEY`: Required for the Google API and Gemini models.
- `HuggingFace API KEY`: Required for the HuggingFace API.
- `Mistral AI API KEY`: Required for the Mistral AI API.
- `Perplexity API KEY`: Required for the Perplexity API and Sonar models.
- `xAI API KEY`: Required for the xAI API and Grok models.
- `OpenAI API version`: Azure OpenAI API version, e.g. 2023-07-01-preview

- `Azure OpenAI API endpoint`: Azure OpenAI API endpoint, https://<your-resource-name>.openai.azure.com/

Layout

- `Style (chat)`: Chat style (Blocks, or ChatGPT-like, or ChatGPT-like Wide. *WebEngine / Chromium* render mode only.
- `Zoom`: Adjusts the zoom in chat window (web render view). *WebEngine / Chromium* render mode only.
- `Font Size (chat window)`: Adjusts the font size in the chat window (plain-text) and notepads.
- `Font Size (input)`: Adjusts the font size in the input window.
- `Font Size (ctx list)`: Adjusts the font size in contexts list.
- `Font Size (toolbox)`: Adjusts the font size in toolbox on right.
- `Layout density`: Adjusts layout elements density. Default: -1.
- `DPI scaling`: Enable/disable DPI scaling. Restart of the application is required for this option to take effect. Default: True.
- `DPI factor`: DPI factor. Restart of the application is required for this option to take effect. Default: 1.0.
- `Auto-collapse user message (px)` - Auto-collapse user message after N pixels of height, set to 0 to disable auto-collapse.
- `Display tips (help descriptions)`: Display help tips, Default: True.
- `Store dialog window positions`: Enable or disable dialogs positions store/restore, Default: True.

Code syntax

- `Code syntax highlight`: Syntax highlight theme in code blocks. *WebEngine / Chromium* render mode only.
- `Disable syntax highlight`: Option to disable syntax highlighting in code blocks. *WebEngine / Chromium* render mode only.
- `Max chars to highlight (static)`: Sets the maximum number of characters to be highlighted in static content. Set to 0 to disable. *WebEngine / Chromium* render mode only.

- `Max lines to highlight (static)`: Sets the maximum number of lines to be highlighted in static content. Set to 0 to disable. *WebEngine / Chromium* render mode only.
- `Max lines to highlight (real-time)`: Sets the maximum number of lines to be highlighted in real-time stream mode. Set to 0 to disable. *WebEngine / Chromium* render mode only.
- `Highlight every N chars (real-time)`: Sets the interval for highlighting every N characters in real-time stream mode. *WebEngine / Chromium* render mode only.
- `Highlight every N line (real-time)`: Sets the interval for highlighting every N lines in real-time stream mode. *WebEngine / Chromium* render mode only.

Files and attachments

- `Store attachments in the workdir upload directory`: Enable to store a local copy of uploaded attachments for future use. Default: True
- `Store images, capture and uploads in data directory`: Enable to store everything in single data directory. Default: False
- `Directory for file downloads`: Subdirectory for downloaded files, e.g. in Assistants mode, inside "data". Default: "download"
- `Model for querying index`: Model to use for preparing query and querying the index when the RAG option is selected.
- `Model for attachment content summary`: Model to use when generating a summary for the content of a file when the Summary option is selected.
- `Use history in RAG query`: When enabled, the content of the entire conversation will be used when preparing a query if mode is RAG or Summary.
- `RAG limit`: Only if the option `Use history in RAG query` is enabled. Specify the limit of how many recent entries in the conversation will be used when generating a query for RAG. 0 = no limit.

Context

- `Context Threshold`: Sets the number of tokens reserved for the model to respond to the next prompt.
- `Limit of last contexts on list to show  (0 = unlimited)`: Limit of the last contexts on list, default: 0 (unlimited).
- `Show context groups on top of the context list`: Display groups on top, default: False
- `Show date separators on the context list`: Show date periods, default: True
- `Show date separators in groups on the context list`: Show date periods in groups, default: True
- `Show date separators in pinned on the context list`: Show date periods in pinned items, default: False
- `Use Context`: Toggles the use of conversation context (memory of previous inputs).
- `Store History`: Toggles conversation history store.
- `Store Time in History`: Chooses whether timestamps are added to the .txt files.
- `Context Auto-summary`: Enables automatic generation of titles for contexts, Default: True.
- `Lock incompatible modes`: If enabled, the app will create a new context when switched to an incompatible mode within an existing context.
- `Search also in conversation content, not only in titles`: When enabled, context search will also consider the content of conversations, not just the titles of conversations.
- `Show LlamaIndex sources`: If enabled, sources utilized will be displayed in the response (if available, it will not work in streamed chat).
- `Show code interpreter output`: If enabled, output from the code interpreter in the Assistant API will be displayed in real-time (in stream mode), Default: True.
- `Use extra context output`: If enabled, plain text output (if available) from command results will be displayed alongside the JSON output, Default: True.

- `Open URLs in built-in browser`: Enable this option to open all URLs in the built-in browser (Chromium) instead of an external browser. Default: False.
- `Model used for auto-summary`: Model used for context auto-summary (generating titles in context list) (default: *gpt-4o-mini*). **Tip:** If you prefer to use local models, you should change the model here as well

Remote tools

Enable/disable remote tools, like Web Search, MCP or Image generation.

Remote tools are available for these providers, and only via their native SDKs:

- Anthropic
- Google
- OpenAI
- xAI

Models

- `Max Output Tokens`: Sets the maximum number of tokens the model can generate for a single response.
- `Max Total Tokens`: Sets the maximum token count that the application can send to the model, including the conversation context.
- `RPM limit`: Sets the limit of maximum requests per minute (RPM), 0 = no limit.
- `Temperature`: Sets the randomness of the conversation. A lower value makes the model's responses more deterministic, while a higher value increases creativity and abstraction.
- `Top-p`: A parameter that influences the model's response diversity, similar to temperature. For more information, please check the OpenAI documentation.

- `Frequency Penalty`: Decreases the likelihood of repetition in the model's responses.
- `Presence Penalty`: Discourages the model from mentioning topics that have already been brought up in the conversation.

Prompts

- `Use native API function calls`: Use API function calls to run commands from plugins instead of using command prompts - disabled in Autonomous and Experts modes, default: True
- `Command execute: instruction`: Prompt for appending command execution instructions. Placeholders: {schema}, {extra}
- `Command execute: extra footer (non-Assistant modes)`: Extra footer to append after commands JSON schema.
- `Command execute: extra footer (Assistant mode only)`: PAdditional instructions to separate local commands from the remote environment that is already configured in the Assistants.
- `Context: auto-summary (system prompt)`: System prompt for context auto-summary.
- `Context: auto-summary (user message)`: User message for context auto-summary. Placeholders: {input}, {output}
- `Agent: evaluation prompt in loop (LlamaIndex) - % complete`: Prompt used for evaluating (by % complete) the response in Agents (LlamaIndex/OpenAI) mode.
- `Agent: evaluation prompt in loop (LlamaIndex) - % score`: Prompt used for evaluating (by % score) the response in Agents (LlamaIndex/OpenAI) mode.
- `Agent: system instruction (Legacy)`: Prompt to instruct how to handle autonomous mode.
- `Agent: continue (Legacy)`: Prompt sent to automatically continue the conversation.
- `Agent: continue (always, more steps) (Legacy)`: Prompt sent to always automatically continue the conversation (more reasoning - "Always continue..." option).

- `Agent: goal update (Legacy)`: Prompt to instruct how to update current goal status.
- `Experts: Master prompt`: Prompt to instruct how to handle experts.
- `Image generate`: Prompt for generating prompts for image generation (if raw-mode is disabled).

Images and video

Image

- `Image size`: The resolution of the generated images. Default: 1024x1024
- `Image quality`: The image quality of the generated images. Default: standard
- `Prompt generation model`: Model used for generating prompts for image generation (if raw-mode is disabled).

Video

- `Aspect ratio`: Specifies the frame aspect ratio (e.g., 16:9, 9:16, 1:1). Availability depends on the selected model.
- `Video duration`: Sets the clip length in seconds; limits may vary by model.
- `FPS`: Determines the frames per second (e.g., 24, 25, 30). Values may be rounded or ignored by the model.
- `Generate audio`: Option to include synthesized background audio if supported by the model.
- `Negative prompt`: Specifies words or phrases to avoid in the output (comma-separated).
- `Prompt enhancement model`: Defines the LLM used to refine your prompt before video generation. This is not the video model.
- `Video resolution`: Sets the target output resolution (e.g., 720p, 1080p). Availability depends on the model.
- `Seed`: Provides an optional random seed for reproducible results; leave empty for random.

Vision and camera

- `Camera Input Device`: Video capture camera index (index of the camera, default: 0).
- `Camera capture width (px)`: Video capture resolution (width).
- `Camera capture height (px)`: Video capture resolution (height).
- `Camera IDX (number)`: Video capture camera index (number of camera).
- `Image capture quality`: Video capture image JPEG quality (%).

Audio

- `Audio Input Backend`: Selects the backend for audio input (Native/QtMultimedia, PyAudio, PyGame)
- `Audio Input Device`: Selects the audio device for Microphone input.
- `Audio Output Backend`: Selects the backend for audio input (Native/QtMultimedia, PyAudio)
- `Audio Output Device`: Selects the audio device for audio output.
- `Channels`: Input channels, default: 1
- `Sampling Rate`: Sampling rate, default: 44100
- `Use cache`: Use cache for generating audio files.
- `Max files to store`: Max files to store on disk for audio cache.
- `Audio notify microphone listening start/stop`: enables audio “tick” notify when microphone listening started/ended.
- `Continuous Audio Recording (Chunks)`: Enable recording in chunks for long audio recordings in notepad (voice notes).
- `VAD prefix padding (in ms)`: VAD prefix padding in ms, default: 300ms (Realtime audio mode)
- `VAD end silence (in ms)`: VAD end silence in ms, default: 2000ms (Realtime audio mode)

Indexes / LlamaIndex

General

- `Indexes`: List of created indexes.

Vector Store

- `Vector Store`: Vector store to use (vector database provided by LlamaIndex).
- `Vector Store (**kwargs)`: Keyword arguments for vector store provider (api_key, index_name, etc.).

Chat

- `Chat mode`: LlamIndex chat mode for use in query engine, default: context
- `Use ReAct agent for Tool calls in Chat with Files mode`: Enable ReAct agent for tool calls in Chat with Files mode.
- `Auto-retrieve additional context`: Enable automatic retrieve of additional context from vector store in every query.

Embeddings

- `Embeddings provider`: Global embeddings provider (for indexing and Chat with Files).
- `Embeddings provider (ENV)`: ENV vars for global embeddings provider (API keys, etc.).
- `Embeddings provider (**kwargs)`: Keyword arguments for global embeddings provider (model_name, etc.).
- `Default embedding providers for attachments`: Define embedding model by provider to use in attachments.
- `RPM limit for embeddings API calls`: Specify the limit of maximum requests per minute (RPM), 0 = no limit.

Indexing

- `Recursive directory indexing`: Enables recursive directory indexing, default is False.

- `Replace old document versions in the index during re-indexing`: If enabled, previous versions of documents will be deleted from the index when the newest versions are indexed, default is True.
- `Excluded file extensions`: File extensions to exclude if no data loader for this extension, separated by comma.
- `Force exclude files`: If enabled, the exclusion list will be applied even when the data loader for the extension is active. Default: False.
- `Stop indexing on error`: If enabled, indexing will stop whenever an error occurs Default: True.
- `Custom metadata to append/replace to indexed documents (files)`: Define custom metadata key => value fields for specified file extensions, separate extensions by comma.nAllowed placeholders: {path}, {relative_path} {filename}, {dirname}, {relative_dir} {ext}, {size}, {mtime}, {date}, {date_time}, {time}, {timestamp}. Use * (asterisk) as extension if you want to apply field to all files. Set empty value to remove field with specified key from metadata.
- `Custom metadata to append/replace to indexed documents (web)`: Define custom metadata key => value fields for specified external data loaders.nAllowed placeholders: {date}, {date_time}, {time}, {timestamp} + {data loader args}

Data Loaders

- `Additional keyword arguments (**kwargs) for data loaders`: Additional keyword arguments, such as settings, API keys, for the data loader. These arguments will be passed to the loader; please refer to the LlamaIndex or LlamaHub loaders reference for a list of allowed arguments for the specified data loader.
- `Use local models in Video/Audio and Image (vision) loaders`: Enables usage of local models in Video/Audio and Image (vision) loaders. If disabled then API models will be used (GPT-4

Vision and Whisper). Note: local models will work only in Python version (not compiled/Snap). Default: False.

Update

- `Auto-index DB in real time`: Enables conversation context auto-indexing in defined modes.
- `ID of index for auto-indexing`: Index to use if auto-indexing of conversation context is enabled.
- `Enable auto-index in modes`: List of modes with enabled context auto-index, separated by comma.
- `DB (ALL), DB (UPDATE), FILES (ALL)`: Index the data – batch indexing is available here.

Agent and experts

General

- `Auto retrieve additional context from RAG`: Auto retrieve additional context from RAG at the beginning if the index is provided.
- `Display a tray notification when the goal is achieved.`: If enabled, a notification will be displayed after goal achieved / finished run.
- `Display full agent output in chat window`: If enabled, a real-time output from agent reasoning will be displayed with the response.

Agents (LlamaIndex / OpenAI)

- `Max steps (per iteration)` - Max steps is one iteration before goal achieved
- `Max evaluation steps in loop` - Maximum evaluation steps to achieve the final result, set 0 to infinity
- `Model for evaluation`: Model used for evaluation with score/percentage (loop). If not selected, then current active

model will be used.

- `Append and compare previous evaluation prompt in next evaluation` - If enabled, previous improvement prompt will be checked in next eval in loop, default: False
- `Split response messages` - Split response messages to separated context items in OpenAI Agents mode.

settings.agent.openai.response.split = Split response messages
settings.agent.openai.response.split.desc = Split re

Autonomous (Legacy agents)

- `Sub-mode for agents`: Sub-mode to use in Agent (Autonomous) mode (chat, llama_index, etc.). Default: chat.
- `Index to use`: Only if sub-mode is llama_index (Chat with files), choose the index to use in both Agent and Expert modes.
- `Use native API function calls`: Use API function calls to run tools from plugins instead of using command prompts - Autonomous mode only, default: False
- `Use Responses API in Agent mode`: Use Responses API instead of ChatCompletions API in Agent (autonomous) mode. OpenAI models only. Default: False

Experts

- `Sub-mode for experts`: Sub-mode to use in Experts mode (chat, llama_index, etc.). Default: chat.
- `Use agent for expert reasoning`: If enabled, the ReAct agent will be used for expert calls and expert reasoning. Default: True
- `Use native API function calls`: Use API function calls to run tools from plugins instead of using command prompts - Experts only, default: False
- `Use Responses API in Experts mode (master)`: Use Responses API instead of ChatCompletions API in Experts (master model). OpenAI models only. Default: False

- `Use Responses API in Experts (slaves)`: Use Responses API instead of ChatCompletions API for Expert instances (slave models). OpenAI models only. Default: False

Accessibility

- `Enable voice control (using microphone)`: enables voice control (using microphone and defined commands).
- `Model`: model used for voice command recognition.
- `Use voice synthesis to describe events on the screen.`: enables audio description of on-screen events.
- `Use audio output cache`: If enabled, all static audio outputs will be cached on the disk instead of being generated every time. Default: True.
- `Audio notify voice command execution`: enables audio "tick" notify when voice command is executed.
- `Control shortcut keys`: configuration for keyboard shortcuts for a specified actions.
- `Blacklist for voice synthesis events describe (ignored events)`: list of muted events for 'Use voice synthesis to describe event' option.
- `Voice control actions blacklist`: Disable actions in voice control; add actions to the blacklist to prevent execution through voice commands.

Personalize

- `About You`: Provide information about yourself, e.g., "My name is... I'm 30 years old, I'm interested in..." This will be included in the model's system prompt.

Warning

Please do not use AI as a “friend”. Real-life friendship is better than using an AI as a friendship replacement. DO NOT become emotionally involved in interactions with an AI.

- `Enable in Modes`: Select the modes where the personalized “about” prompt will be used.

Updates

- `Check for updates on start`: Enables checking for updates on start. Default: True.
- `Check for updates in background`: Enables checking for updates in background (checking every 5 minutes). Default: True.

Debug

- `Show debug menu`: Enables debug (developer) menu.
- `Log level`: toggle log level (ERROR|WARNING|INFO|DEBUG)
- `Log and debug context`: Enables logging of context input/output.
- `Log and debug events`: Enables logging of event dispatch.
- `Log plugin usage to console`: Enables logging of plugin usage to console.
- `Log DALL-E usage to console`: Enables logging of DALL-E usage to console.
- `Log attachments usage to console`: Enables logging of attachments usage to console.
- `Log Agents usage to console`: Enables logging of Agents usage to console.
- `Log LlamaIndex usage to console`: Enables logging of LlamaIndex usage to console.
- `Log Assistants usage to console`: Enables logging of Assistants API usage to console.

JSON files

The configuration is stored in JSON files for easy manual modification outside of the application. These configuration files are located in the user's work directory within the following subdirectory:

```
{HOME_DIR}/.config/pygpt-net/
```

Manual configuration

You can manually edit the configuration files in this directory (this is your work directory):

```
{HOME_DIR}/.config/pygpt-net/
```

- `assistants.json` - stores the list of assistants.
- `attachments.json` - stores the list of current attachments.
- `config.json` - stores the main configuration settings.
- `models.json` - stores models configurations.
- `cache` - a directory for audio cache.
- `capture` - a directory for captured images from camera and screenshots
- `css` - a directory for CSS stylesheets (user override)
- `history` - a directory for context history in `.txt` format.
- `idx` - `LlamaIndex` indexes
- `img` - a directory for images generated with `DALL-E 3` and `DALL-E 2`, saved as `.png` files.
- `locale` - a directory for locales (user override)
- `data` - a directory for data files and files downloaded/generated by models.
- `presets` - a directory for presets stored as `.json` files.
- `upload` - a directory for local copies of attachments coming from outside the workdir
- `db.sqlite` - a database with contexts, notepads and indexes data records
- `app.log` - a file with error and debug log

Setting the Working Directory Using Command Line Arguments

To set the current working directory using a command-line argument, use:

```
python3 ./run.py --workdir="/path/to/workdir"
```

or, for the binary version:

```
pygpt.exe --workdir="/path/to/workdir"
```

Translations / locale

Locale *.ini* files are located in the directory:

```
./data/locale
```

This directory is automatically scanned when the application launches. To add a new translation, create and save the file with the appropriate name, for example:

```
locale.es.ini
```

This will add Spanish as a selectable language in the application’s language menu.

Overwriting CSS and locales with Your Own Files:

You can also overwrite files in the `locale` and `css` app directories with your own files in the user directory. This allows you to overwrite language files or CSS styles in a very simple way - by just creating files in your working directory.

```
{HOME_DIR}/.config/pygpt-net/
```

- *locale* - a directory for locales in `.ini` format.
- *css* - a directory for CSS styles

Adding Your Own Fonts

You can add your own fonts and use them in CSS files. To load your own fonts, you should place them in the `%workdir%/fonts` directory. Supported font types include: `otf`, `ttf`. You can see the list of loaded fonts in `Debug / Config`.

Example:

```
%workdir%
|_css
|_data
|_fonts
   |_MyFont
     |_MyFont-Regular.ttf
     |_MyFont-Bold.ttf
     |...
```

```
pre {{
    font-family: 'MyFont';
}}
```

Data Loaders

Configuring data loaders

In the `Settings -> LlamaIndex -> Data loaders` section you can define the additional keyword arguments to pass into data loader instance.

In most cases, an internal LlamaIndex loaders are used internally. You can check these base loaders e.g. here:

Files loaders: https://github.com/run-llama/llama_index/tree/main/llama-index-integrations/readers/llama-index-readers-file/llama_index/readers/file

Web loaders: https://github.com/run-llama/llama_index/tree/main/llama-index-integrations/readers/llama-index-readers-web

Tip

To index an external data or data from the Web just ask for it, by using `Web Search` plugin, e.g. you can ask the model with `Please index the youtube video: URL to video`, etc. Data loader for a specified content will be choosen automatically.

Allowed additional keyword arguments for built-in data loaders (files):

CSV Files (file_csv)

- `concat_rows` - bool, default: `True`
- `encoding` - str, default: `utf-8`

HTML Files (file_html)

- `tag` - str, default: `section`
- `ignore_no_id` - bool, default: `False`

Image (vision) (file_image_vision)

This loader can operate in two modes: local model and API. If the local mode is enabled, then the local model will be used. The local mode requires a Python/PyPi version of the application and is not available in the compiled or Snap versions. If the API mode (default) is selected, then the OpenAI API and the standard vision model will be used.

Note

Usage of API mode consumes additional tokens in OpenAI API (for `GPT-4 Vision` model)!

Local mode requires `torch`, `transformers`, `sentencepiece` and `Pillow` to be installed and uses the `Salesforce/blip2-opt-2.7b` model to describing images.

- `keep_image` - bool, default: `False`
- `local_prompt` - str, default: `Question: describe what you see in this image. Answer:`
- `api_prompt` - str, default: `Describe what you see in this image` - Prompt to use in API
- `api_model` - str, default: `gpt-4-vision-preview` - Model to use in API
- `api_tokens` - int, default: `1000` - Max output tokens in API

IPYNB Notebook files (file_ipynb)

- `parser_config` - dict, default: `None`
- `concatenate` - bool, default: `False`

Markdown files (file_md)

- `remove_hyperlinks` - bool, default: `True`
- `remove_images` - bool, default: `True`

PDF documents (file_pdf)

- `return_full_document` - bool, default: `False`

Video/Audio (file_video_audio)

This loader can operate in two modes: local model and API. If the local mode is enabled, then the local `Whisper` model will be used. The local mode requires a Python/PyPi version of the application and is not available in the compiled or Snap versions. If the API mode (default) is selected, then the currently selected provider in `Audio Input` plugin will be used. If the `OpenAI Whisper` is chosen then the OpenAI API and the API Whisper model will be used.

Note: Usage of Whisper via API consumes additional tokens in OpenAI API (for `Whisper` model)!

Local mode requires `torch` and `openai-whisper` to be installed and uses the `Whisper` model locally to transcribing video and audio.

- `model_version` - str, default: `base` - Whisper model to use, available models: https://github.com/openai/whisper

XML files (file_xml)

- `tree_level_split` - int, default: `0`

Allowed additional keyword arguments for built-in data loaders (Web and external content):

Bitbucket (web_bitbucket)

- `username` - str, default: *None*
- `api_key` - str, default: *None*
- `extensions_to_skip` - list, default: *[]*

ChatGPT Retrieval (web_chatgpt_retrieval)

- `endpoint_url` - str, default: *None*
- `bearer_token` - str, default: *None*
- `retries` - int, default: *None*
- `batch_size` - int, default: *100*

Google Calendar (web_google_calendar)

- `credentials_path` - str, default: *credentials.json*
- `token_path` - str, default: *token.json*

Google Docs (web_google_docs)

- `credentials_path` - str, default: *credentials.json*
- `token_path` - str, default: *token.json*

Google Drive (web_google_drive)

- `credentials_path` - str, default: *credentials.json*
- `token_path` - str, default: *token.json*
- `pydrive_creds_path` - str, default: *creds.txt*

Google Gmail (web_google_gmail)

- `credentials_path` - str, default: *credentials.json*
- `token_path` - str, default: *token.json*
- `use_iterative_parser` - bool, default: *False*
- `max_results` - int, default: *10*
- `results_per_page` - int, default: *None*

Google Keep (web_google_keep)

- `credentials_path` - str, default: *keep_credentials.json*

Google Sheets (web_google_sheets)

- `credentials_path` - str, default: *credentials.json*
- `token_path` - str, default: *token.json*

GitHub Issues (web_github_issues)

- `token` - str, default: *None*
- `verbose` - bool, default: *False*

GitHub Repository (web_github_repository)

- `token` - str, default: *None*
- `verbose` - bool, default: *False*
- `concurrent_requests` - int, default: *5*
- `timeout` - int, default: *5*
- `retries` - int, default: *0*
- `filter_dirs_include` - list, default: *None*
- `filter_dirs_exclude` - list, default: *None*
- `filter_file_ext_include` - list, default: *None*
- `filter_file_ext_exclude` - list, default: *None*

Microsoft OneDrive (web_microsoft_onedrive)

- `client_id` - str, default: *None*
- `client_secret` - str, default: *None*
- `tenant_id` - str, default: *consumers*

Sitemap (XML) (web_sitemap)

- `html_to_text` - bool, default: *False*
- `limit` - int, default: *10*

SQL Database (web_database)

- `uri` - str, default: *None*

You can provide a single URI in the form of: `{scheme}://{user}:{password}@{host}:{port}/{dbname}`, or you can provide each field manually:

- `scheme` - str, default: *None*
- `host` - str, default: *None*
- `port` - str, default: *None*
- `user` - str, default: *None*
- `password` - str, default: *None*

- `dbname` - str, default: *None*

Twitter/X posts (web_twitter)

- `bearer_token` - str, default: *None*
- `num_tweets` - int, default: *100*

Vector stores

Available vector stores (provided by `LlamaIndex`):

- ChromaVectorStore
- ElasticsearchStore
- PinecodeVectorStore
- RedisVectorStore
- SimpleVectorStore

You can configure selected vector store by providing config options like `api_key`, etc. in `Settings -> LlamaIndex` window.

Arguments provided here (on list: `Vector Store (**kwargs)` in `Advanced settings` will be passed to selected vector store provider. You can check keyword arguments needed by selected provider on LlamaIndex API reference page:

https://docs.llamaindex.ai/en/stable/api_reference/storage/vector_store.html

Which keyword arguments are passed to providers?

For `ChromaVectorStore` and `SimpleVectorStore` all arguments are set by PyGPT and passed internally (you do not need to configure anything). For other providers you can provide these arguments:

ElasticsearchStore

Keyword arguments for ElasticsearchStore(`**kwargs`):

- `index_name` (default: current index ID, already set, not required)
- any other keyword arguments provided on list

PinecodeVectorStore

Keyword arguments for Pinecone(`**kwargs`):

- `api_key`
- index_name (default: current index ID, already set, not required)

RedisVectorStore

Keyword arguments for RedisVectorStore(`**kwargs`):

- `index_name` (default: current index ID, already set, not required)
- any other keyword arguments provided on list

You can extend list of available providers by creating custom provider and registering it on app launch.

By default, you are using chat-based mode when using `Chat with Files`. If you want to only query index (without chat) you can enable `Query index only (without chat)` option.

Adding custom vector stores and offline data loaders

You can create a custom vector store provider or data loader for your data and develop a custom launcher for the application.

See the section `Extending PyGPT / Adding a custom Vector Store provider` for more details.

Updates

Updating PyGPT

PyGPT comes with an integrated update notification system. When a new version with additional features is released, you’ll receive an alert within the app.

To get the new version, simply download it and start using it in place of the old one. All your custom settings like configuration, presets, indexes, and past conversations will be kept and ready to use right away in the new version.

Debugging and Logging

In `Settings -> Developer` dialog, you can enable the `Show debug menu` option to turn on the debugging menu. The menu allows you to inspect the status of application elements. In the debugging menu, there is a `Logger` option that opens a log window. In the window, the program's operation is displayed in real-time.

Logging levels:

By default, all errors and exceptions are logged to the file:

```
{HOME_DIR}/.config/pygpt-net/app.log
```

To increase the logging level (`ERROR` level is default), run the application with `--debug` argument:

```
python3 run.py --debug=1
```

or

```
python3 run.py --debug=2
```

- The value `1` enables the `INFO` logging level.
- The value `2` enables the `DEBUG` logging level (most information).

Compatibility (legacy) mode

If you have a problems with *WebEngine / Chromium* renderer you can force the legacy mode by launching the app with command line arguments:

```
python3 run.py --legacy=1
```

and to force disable OpenGL hardware acceleration:

```
python3 run.py --disable-gpu=1
```

You can also manualy enable legacy mode by editing config file - open the `%WORKDIR%/config.json` config file in editor and set the following options:

```
"render.engine": "legacy",
"render.open_gl": false,
```

Extending PyGPT

Quick start

You can create your own extension for **PyGPT** at any time.

PyGPT can be extended with:

- custom models
- custom plugins
- custom LLM wrappers
- custom vector store providers
- custom data loaders
- custom audio input providers
- custom audio output providers
- custom web search engine providers
- custom agents (LlamaIndex or OpenAI Agents)

Examples (tutorial files)

See the `examples` directory in this repository with examples of custom launcher, plugin, vector store, LLM provider and data loader:

- `examples/custom_launcher.py`
- `examples/example_audio_input.py`
- `examples/example_audio_output.py`
- `examples/example_data_loader.py`
- `examples/example_llm.py`
- `examples/example_plugin.py`
- `examples/example_vector_store.py`
- `examples/example_web_search.py`

These example files can be used as a starting point for creating your own extensions for **PyGPT**.

Extending PyGPT with custom plugins, LLMs wrappers and vector stores:

- You can pass custom plugin instances, LLMs wrappers and vector store providers to the launcher.
- This is useful if you want to extend PyGPT with your own plugins, vectors storage and LLMs.

To register custom plugins:

- Pass a list with the plugin instances as `plugins` keyword argument.

To register custom LLMs wrappers:

- Pass a list with the LLMs wrappers instances as `llms` keyword argument.

To register custom vector store providers:

- Pass a list with the vector store provider instances as `vector_stores` keyword argument.

To register custom data loaders:

- Pass a list with the data loader instances as `loaders` keyword argument.

To register custom audio input providers:

- Pass a list with the audio input provider instances as `audio_input` keyword argument.

To register custom audio output providers:

- Pass a list with the audio output provider instances as `audio_output` keyword argument.

To register custom web providers:

- Pass a list with the web provider instances as `web` keyword argument.

To register an agent:

- Pass a list with the agent instances as `agents` keyword argument.

Adding a custom model

To add a new model using the OpenAI API, or LlamaIndex wrapper, use the editor in `Config -> Models` or manually edit the `models.json` file by inserting the model's configuration details. If you are adding a model via LlamaIndex, ensure to include the model's name, its supported modes (either `chat`, `completion`, or both), the LLM provider (such as `OpenAI` or `HuggingFace`), and, if you are using an external API-based model, an optional `API KEY` along with any other necessary environment settings.

Example of models configuration - `%WORKDIR%/models.json`:

```
"gpt-3.5-turbo": {
    "id": "gpt-3.5-turbo",
    "name": "gpt-3.5-turbo",
    "mode": [
        "chat",
        "assistant",
        "langchain",
        "llama_index"
    ],
    "provider": "openai",
    "llama_index": {
        "args": [
            {
                "name": "model",
                "value": "gpt-3.5-turbo",
                "type": "str"
            }
        ],
        "env": [
            {
                "name": "OPENAI_API_KEY",
                "value": "{api_key}"
            }
        ]
    },
    "ctx": 4096,
    "tokens": 4096,
    "default": false
},
```

Tip

`{api_key}` in `models.json` is a placeholder for the main OpenAI API KEY from the settings. It will be replaced by the configured key value.

There is built-in support for those LLM providers:

- `Anthropic`
- `Azure OpenAI`
- `Deepseek API`
- `Google`
- `HuggingFace`
- `Local models` (OpenAI API compatible)
- `Ollama`
- `OpenAI`
- `OpenRouter`
- `Perplexity`

- `xAI`

Adding a custom plugin

Creating Your Own Plugin

You can create your own plugin for **PyGPT**. The plugin can be written in Python and then registered with the application just before launching it. All plugins included with the app are stored in the `plugin` directory - you can use them as coding examples for your own plugins.

Examples (tutorial files)

See the example plugin in this `examples` directory:

- `examples/example_plugin.py`

These example file can be used as a starting point for creating your own plugin for **PyGPT**.

To register a custom plugin:

- Create a custom launcher for the app.
- Pass a list with the custom plugin instances as `plugins` keyword argument.

Example of a custom launcher:

```
# custom_launcher.py

from pygpt_net.app import run
from plugins import CustomPlugin, OtherCustomPlugin
from llms import CustomLLM
from vector_stores import CustomVectorStore

plugins = [
    CustomPlugin(),
    OtherCustomPlugin(),
]
llms = [
    CustomLLM(),
]
vector_stores = [
    CustomVectorStore(),
]

run(
    plugins=plugins,
    llms=llms,
    vector_stores=vector_stores,
)
```

Handling events

In the plugin, you can receive and modify dispatched events. To do this, create a method named `handle(self, event, *args, **kwargs)` and handle the received events like here:

```
# custom_plugin.py

from pygpt_net.core.events import Event


def handle(self, event: Event, *args, **kwargs):
    """
    Handle dispatched events

    :param event: event object
    """
    name = event.name
    data = event.data
    ctx = event.ctx

    if name == Event.INPUT_BEFORE:
        self.some_method(data['value'])
    elif name == Event.CTX_BEGIN:
        self.some_other_method(ctx)
    else:
        # ...
```

List of Events

Event names are defined in `Event` class in `pygpt_net.core.events`.

Syntax: `event name` - triggered on, `event data` *(data type)*:

- `AI_NAME` - when preparing an AI name, `data['value']` *(string, name of the AI assistant)*
- `AGENT_PROMPT` - on agent prompt in eval mode, `data['value']` *(string, prompt)*
- `AUDIO_INPUT_RECORD_START` - start audio input recording
- `AUDIO_INPUT_RECORD_STOP` - stop audio input recording
- `AUDIO_INPUT_RECORD_TOGGLE` - toggle audio input recording
- `AUDIO_INPUT_TRANSCRIBE` - on audio file transcribe, `data['path']` *(string, path to audio file)*
- `AUDIO_INPUT_STOP` - force stop audio input
- `AUDIO_INPUT_TOGGLE` - when speech input is enabled or disabled, `data['value']` *(bool, True/False)*
- `AUDIO_OUTPUT_STOP` - force stop audio output
- `AUDIO_OUTPUT_TOGGLE` - when speech output is enabled or disabled, `data['value']` *(bool, True/False)*
- `AUDIO_READ_TEXT` - on text read using speech synthesis, `data['text']` *(str, text to read)*
- `CMD_EXECUTE` - when a command is executed, `data['commands']` *(list, commands and arguments)*
- `CMD_INLINE` - when an inline command is executed, `data['commands']` *(list, commands and arguments)*
- `CMD_SYNTAX` - when appending syntax for commands, `data['prompt'], data['syntax']` *(string, list, prompt and list with commands usage syntax)*

- `CMD_SYNTAX_INLINE` - when appending syntax for commands (inline mode), `data['prompt'], data['syntax']` *(string, list, prompt and list with commands usage syntax)*
- `CTX_AFTER` - after the context item is sent, `ctx`
- `CTX_BEFORE` - before the context item is sent, `ctx`
- `CTX_BEGIN` - when context item create, `ctx`
- `CTX_END` - when context item handling is finished, `ctx`
- `CTX_SELECT` - when context is selected on list, `data['value']` *(int, ctx meta ID)*
- `DISABLE` - when the plugin is disabled, `data['value']` *(string, plugin ID)*
- `ENABLE` - when the plugin is enabled, `data['value']` *(string, plugin ID)*
- `FORCE_STOP` - on force stop plugins
- `INPUT_BEFORE` - upon receiving input from the textarea, `data['value']` *(string, text to be sent)*
- `MODE_BEFORE` - before the mode is selected `data['value'], data['prompt']` *(string, string, mode ID)*
- `MODE_SELECT` - on mode select `data['value']` *(string, mode ID)*
- `MODEL_BEFORE` - before the model is selected `data['value']` *(string, model ID)*
- `MODEL_SELECT` - on model select `data['value']` *(string, model ID)*
- `PLUGIN_SETTINGS_CHANGED` - on plugin settings update (saving settings)
- `PLUGIN_OPTION_GET` - on request for plugin option value `data['name'], data['value']` *(string, any, name of requested option, value)*
- `POST_PROMPT` - after preparing a system prompt, `data['value']` *(string, system prompt)*
- `POST_PROMPT_ASYNC` - after preparing a system prompt, just before request in async thread, `data['value']` *(string, system prompt)*
- `POST_PROMPT_END` - after preparing a system prompt, just before request in async thread, at the very end `data['value']` *(string,*

system prompt)

- `PRE_PROMPT` - before preparing a system prompt, `data['value']` *(string, system prompt)*
- `SYSTEM_PROMPT` - when preparing a system prompt, `data['value']` *(string, system prompt)*
- `TOOL_OUTPUT_RENDER` - when rendering extra content from tools from plugins, `data['content']` *(string, content)*
- `UI_ATTACHMENTS` - when the attachment upload elements are rendered, `data['value']` *(bool, show True/False)*
- `UI_VISION` - when the vision elements are rendered, `data['value']` *(bool, show True/False)*
- `USER_NAME` - when preparing a user's name, `data['value']` *(string, name of the user)*
- `USER_SEND` - just before the input text is sent, `data['value']` *(string, input text)*

You can stop the propagation of a received event at any time by setting `stop` to *True*:

```
event.stop = True
```

Events flow can be debugged by enabling the option `Config -> Settings -> Developer -> Log and debug events`.

Adding a custom LLM provider

Handling LLMs with LlamaIndex is implemented through separated wrappers. This allows for the addition of support for any provider and model available via LlamaIndex. All built-in wrappers for the models and its providers are placed in the `pygpt_net.provider.llms`.

These wrappers are loaded into the application during startup using `launcher.add_llm()` method:

```
# app.py

from pygpt_net.provider.api.openai import OpenAILLM
from pygpt_net.provider.llms.azure_openai import AzureOpenAILLM
from pygpt_net.provider.llms.anthropic import AnthropicLLM
from pygpt_net.provider.llms.hugging_face import HuggingFaceLLM
from pygpt_net.provider.llms.ollama import OllamaLLM
from pygpt_net.provider.llms.google import GoogleLLM

def run(**kwargs):
    """Runs the app."""
    # Initialize the app
    launcher = Launcher()
    launcher.init()

    # Register plugins
    ...

    # Register langchain and llama-index LLMs wrappers
    launcher.add_llm(OpenAILLM())
    launcher.add_llm(AzureOpenAILLM())
    launcher.add_llm(AnthropicLLM())
    launcher.add_llm(HuggingFaceLLM())
    launcher.add_llm(OllamaLLM())
    launcher.add_llm(GoogleLLM())

    # Launch the app
    launcher.run()
```

To add support for providers not included by default, you can create your own wrapper that returns a custom model to the application and then pass this custom wrapper to the launcher.

Extending PyGPT with custom plugins and LLM wrappers is straightforward:

- Pass instances of custom plugins and LLM wrappers directly to the launcher.

To register custom LLM wrappers:

- Provide a list of LLM wrapper instances as the `llms` keyword argument when initializing the custom app launcher.

Example:

```
# custom_launcher.py

from pygpt_net.app import run
from plugins import CustomPlugin, OtherCustomPlugin
from llms import CustomLLM

plugins = [
    CustomPlugin(),
    OtherCustomPlugin(),
]
llms = [
    CustomLLM(),  # <--- custom LLM provider (wrapper)
]
vector_stores = []

run(
    plugins=plugins,
    llms=llms,
    vector_stores=vector_stores,
)
```

Examples (tutorial files)

See the `examples` directory in this repository with examples of custom launcher, plugin, vector store, LLM provider and data loader:

- `examples/custom_launcher.py`
- `examples/example_audio_input.py`
- `examples/example_audio_output.py`
- `examples/example_data_loader.py`
- `examples/example_llm.py` <– use it as an example
- `examples/example_plugin.py`
- `examples/example_vector_store.py`
- `examples/example_web_search.py`

These example files can be used as a starting point for creating your own extensions for **PyGPT**.

To integrate your own model or provider into **PyGPT**, you can also reference the classes located in the `pygpt_net.provider.llms`. These samples can act as an more complex example for your custom class. Ensure that your custom wrapper class includes two essential methods: `chat` and `completion`. These methods should return the respective objects required for the model to operate in `chat` and `completion` modes.

Every single LLM provider (wrapper) inherits from `BaseLLM` class and can provide 2 components: provider for LlamaIndex, and provider for Embeddings.

Adding a custom vector store provider

You can create a custom vector store provider or data loader for your data and develop a custom launcher for the application. To register your custom vector store provider or data loader, simply register it by passing the vector store provider instance to `vector_stores` keyword argument and loader instance in the `loaders` keyword argument:

```
# app.py

# vector stores
from pygpt_net.provider.vector_stores.chroma import
ChromaProvider
from pygpt_net.provider.vector_stores.elasticsearch import
ElasticsearchProvider
from pygpt_net.provider.vector_stores.pinecode import
PinecodeProvider
from pygpt_net.provider.vector_stores.redis import
RedisProvider
```

```
from pygpt_net.provider.vector_stores.simple import
SimpleProvider

def run(**kwargs):
    # ...
    # register base vector store providers (llama-index)
    launcher.add_vector_store(ChromaProvider())
    launcher.add_vector_store(ElasticsearchProvider())
    launcher.add_vector_store(PinecodeProvider())
    launcher.add_vector_store(RedisProvider())
    launcher.add_vector_store(SimpleProvider())

    # register custom vector store providers (llama-index)
    vector_stores = kwargs.get('vector_stores', None)
    if isinstance(vector_stores, list):
        for store in vector_stores:
            launcher.add_vector_store(store)

    # ...
```

To register your custom vector store provider just register it by passing provider instance in `vector_stores` keyword argument:

```
# custom_launcher.py

from pygpt_net.app import run
from plugins import CustomPlugin, OtherCustomPlugin
from llms import CustomLLM
from vector_stores import CustomVectorStore

plugins = [
    CustomPlugin(),
    OtherCustomPlugin(),
]
llms = [
    CustomLLM(),
]
vector_stores = [
    CustomVectorStore(),  # <--- custom vector store provider
]
```

```
run(
    plugins=plugins,
    llms=llms,
    vector_stores=vector_stores,
)
```

The vector store provider must be an instance of `pygpt_net.provider.vector_stores.base.BaseStore`. You can review the code of the built-in providers in `pygpt_net.provider.vector_stores` and use them as examples when creating a custom provider.

Adding a custom data loader

```
# custom_launcher.py

from pygpt_net.app import run
from plugins import CustomPlugin, OtherCustomPlugin
from llms import CustomLLM
from vector_stores import CustomVectorStore
from loaders import CustomLoader

plugins = [
    CustomPlugin(),
    OtherCustomPlugin(),
]
llms = [
    CustomLLM(),
]
vector_stores = [
    CustomVectorStore(),
]
loaders = [
    CustomLoader(),  # <---- custom data loader
]

run(
    plugins=plugins,
    llms=llms,
    vector_stores=vector_stores,  # <--- list with custom
vector store providers
    loaders=loaders  # <--- list with custom data loaders
)
```

The data loader must be an instance of `pygpt_net.provider.loaders.base.BaseLoader`. You can review the code of the built-in loaders in `pygpt_net.provider.loaders` and use them as examples when creating a custom loader.

Security

Security Policy

We take the security of our project seriously. While we strive to keep our code and dependencies secure, there may be occasional security vulnerabilities.

Reporting a Vulnerability

If you believe you have found a security issue in our project, we encourage you to report it to us. We welcome any responsible disclosure of potential security vulnerabilities.

Please send us an email at info@pygpt.net with the following information:

- A clear description of the issue you have identified.
- Any steps to reproduce the issue or a proof-of-concept (if possible).

We will review your report and work to understand and address any potential issues as quickly as possible.

External Libraries

Our project uses external libraries, and we try to keep them up-to-date. If you discover a security vulnerability in any of the libraries that our project depends on, please report it directly to the maintainers of that library, and also let us know.

Thank you for helping us maintain the security of our project.

Credits

Links

Official website: https://pygpt.net

Support and donate: https://pygpt.net/#donate

GitHub: https://github.com/szczyglis-dev/py-gpt

Discord: https://pygpt.net/discord

Snap Store: https://snapcraft.io/pygpt

Microsoft Store: https://apps.microsoft.com/detail/XP99R4MX3X65VQ

PyPI: https://pypi.org/project/pygpt-net

Author: Marcin Szczygliński (Poland, EU)

Contact: info@pygpt.net

License: MIT License

Special thanks

GitHub's community:

- **BillionShields**
- **gfsysa**
- **glinkot**
- **kaneda2004**

- **KingOfTheCastle**
- **linnflux**
- **lukasz-pekala**
- **moritz-t-w**
- **oleksii-honchar**
- **yf007**

Third-party libraries

Full list of external libraries used in this project is located in the **requirements.txt** file in the main folder of the repository.

All used SVG icons are from `Material Design Icons` provided by Google:

https://github.com/google/material-design-icons

https://fonts.google.com/icons

Code of the LlamaIndex offline loaders integrated into app is taken from LlamaHub: https://llamahub.ai

Awesome ChatGPT Prompts (used in templates): https://github.com/f/awesome-chatgpt-prompts/

Index