Contributing

Setting-up

Preferred method - VSCode Dev Container

VSCode dev containers are a great way to containerize not only the necessary requirements but al

To use our dev container:

1. Download Docker Desktop
2. Download VSCode
3. Within VSCode, install the Remote Development extension (ms-vscode-remote.vscode-remote-e
4. Open the VSCode command palette (cmd / ctrl + shift + p)
5. Select `Dev Containers: Rebuild and Reopen in container`. A new VSCode window should open
   - Be sure the correct VSCode Python Interpreter is selected. Additional instructions are provided
6. To test that your environment is set up correctly, open a new terminal and run the command `pyt
7. After the initial build, you should now be able to leave and re-enter the container any time withou

NOTE: If you use this method, your default shell will be the poetry shell which will contain all the ne

Alternative method

To run the tests, you need to first clone the repository.

Then install the package through poetry:
Note - You may need to install poetry. You likely will just need to run `pip install poetry`. See [here](

After installing poetry, use it to install our development requirements.

```bash
poetry install --with dev
```

Testing

To run the all tests, or a specific test suite, use the following commands:

```bash
poetry run pytest
poetry run pytest tests/PATH_TO_TEST_SUITE
```

If you are changing CI actions, you can use the [act](https://nektosact.com/introduction.html) tool to

Example for testing the push action:
You may need the `--container-architecture linux/amd64` flag if you are on an M1/2 mac.

```
black==24.2.0
pre-commit==3.7.0
pytest==8.2.1
pytest-env==1.1.3
pytest-mock==3.12.0
ruff==0.3.0
torch==2.2.1
transformers==4.38.2
```

backoff
datasets
joblib<=1.3.2
openai>=0.28.1,<2.0.0
optuna
pandas
pydantic~=2.0
regex
requests
structlog
tqdm
ujson

```
from setuptools import find_packages, setup

# Read the content of the README file
with open("README.md", encoding="utf-8") as f:
    long_description = f.read()

# Read the content of the requirements.txt file
with open("requirements.txt", encoding="utf-8") as f:
    requirements = f.read().splitlines()

setup(
    #replace_package_name_marker
    name="dspy-ai",
    #replace_package_version_marker
    version="2.4.10",
    description="DSPy",
    long_description=long_description,
    long_description_content_type='text/markdown',
    url="https://github.com/stanfordnlp/dsp",
    author="Omar Khattab",
    author_email="okhattab@stanford.edu",
    license="MIT License",
    packages=find_packages(include=['dsp.*', 'dspy.*', 'dsp', 'dspy']),
    python_requires='>=3.9',
    install_requires=requirements,

    extras_require={
        "chromadb": ["chromadb~=0.4.14"],
        "qdrant": ["qdrant-client", "fastembed"],
        "marqo": ["marqo~=3.1.0"],
        "mongodb": ["pymongo~=3.12.0"],
        "pinecone": ["pinecone-client~=2.2.4"],
        "weaviate": ["weaviate-client~=4.6.5"],
        "faiss-cpu": ["sentence_transformers", "faiss-cpu"],
        "milvus": ["pymilvus~=2.3.7"],
        "google-vertex-ai": ["google-cloud-aiplatform==1.43.0"],
        "myscale":["clickhouse-connect"],
        "snowflake": ["snowflake-snowpark-python"],
        "fastembed": ["fastembed"],
        "google-vertex-ai": ["google-cloud-aiplatform==1.43.0"],
        "myscale":["clickhouse-connect"],
        "groq": ["groq~=0.8.0"],
    },
    classifiers=[
        "Development Status :: 3 - Alpha",
        "Intended Audience :: Science/Research",
        "License :: OSI Approved :: MIT License",
```

Getting Started with Create React App

This project was bootstrapped with [Create React App](https://github.com/facebook/create-react-ap

Available Scripts

In the project directory, you can run:

`npm start`

Runs the app in the development mode.\
Open http://localhost:3000 to view it in your browser.

The page will reload when you make changes.\
You may also see any lint errors in the console.

`npm test`

Launches the test runner in the interactive watch mode.\
See the section about [running tests](https://facebook.github.io/create-react-app/docs/running-tests

`npm run build`

Builds the app for production to the `build` folder.\
It correctly bundles React in production mode and optimizes the build for the best performance.

The build is minified and the filenames include the hashes.\
Your app is ready to be deployed!

See the section about [deployment](https://facebook.github.io/create-react-app/docs/deployment) fo

`npm run eject`

Note: this is a one-way operation. Once you `eject`, you can't go back!

If you aren't satisfied with the build tool and configuration choices, you can `eject` at any time. This

Instead, it will copy all the configuration files and the transitive dependencies (webpack, Babel, ESL

You don't have to ever use `eject`. The curated feature set is suitable for small and middle deploym

Learn More

You can learn more in the [Create React App documentation](https://facebook.github.io/create-reac

To learn React, check out the [React documentation](https://reactjs.org/).

```
{
  "name": "dsp-app",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@emotion/react": "^11.10.5",
    "@emotion/styled": "^11.10.5",
    "@mui/icons-material": "^5.11.9",
    "@mui/material": "^5.11.9",
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "axios": "^1.3.3",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

```
{
  "name": "dsp-app",
  "version": "0.1.0",
  "lockfileVersion": 2,
  "requires": true,
  "packages": {
    "": {
      "name": "dsp-app",
      "version": "0.1.0",
      "dependencies": {
        "@emotion/react": "^11.10.5",
        "@emotion/styled": "^11.10.5",
        "@mui/icons-material": "^5.11.9",
        "@mui/material": "^5.11.9",
        "@testing-library/jest-dom": "^5.16.5",
        "@testing-library/react": "^13.4.0",
        "@testing-library/user-event": "^13.5.0",
        "axios": "^1.3.3",
        "react": "^18.2.0",
        "react-dom": "^18.2.0",
        "react-scripts": "5.0.1",
        "web-vitals": "^2.1.4"
      }
    },
    "node_modules/@adobe/css-tools": {
      "version": "4.1.0",
      "resolved": "https://registry.npmjs.org/@adobe/css-tools/-/css-tools-4.1.0.tgz",
      "integrity": "sha512-mMVJ/j/GbZ/De4ZHWbQAQO1J6iVnjtZLc9WEdkUQb8S/Bu2cAF2bETXUg
    },
    "node_modules/@ampproject/remapping": {
      "version": "2.2.0",
      "resolved": "https://registry.npmjs.org/@ampproject/remapping/-/remapping-2.2.0.tgz",
      "integrity": "sha512-qRmjj8nj9qmLTQXXmaR1cck3UXSRMPrbsLJAasZpF+t3riI71BXed5ebIOY
      "dependencies": {
        "@jridgewell/gen-mapping": "^0.1.0",
        "@jridgewell/trace-mapping": "^0.3.9"
      },
      "engines": {
        "node": ">=6.0.0"
      }
    },
    "node_modules/@babel/code-frame": {
      "version": "7.18.6",
      "resolved": "https://registry.npmjs.org/@babel/code-frame/-/code-frame-7.18.6.tgz",
      "integrity": "sha512-TDCmlK5eOvH+eH7cdAFlNXeVJqWIQ7gW9tY1GJIpUtFb6CmjVyq2VM3u
      "dependencies": {
        "@babel/highlight": "^7.18.6"
```

```
module.exports = {
  presets: [require.resolve('@docusaurus/core/lib/babel/preset')],
};
```

```
// Importing necessary modules from Docusaurus and Prism for syntax highlighting
import { themes as prismThemes } from 'prism-react-renderer';
import type { Config } from '@docusaurus/types';

// Config object type declaration for TypeScript
const config: Config = {
  title: 'DSPy',
  tagline: 'Programming·not prompting·Language Models',
  favicon: 'img/logo.png',

  // The URL and base URL for your project
  url: 'https://dspy.ai',
  baseUrl: '/',

  // GitHub configuration for deployment and organization information
  organizationName: 'stanfordnlp',
  projectName: 'dspy',

  // Handling of broken links and markdown links
  onBrokenLinks: 'throw',
  onBrokenMarkdownLinks: 'warn',

  // Internationalization configuration, with 'en' as the default language
  i18n: {
    defaultLocale: 'en',
    locales: ['en'],
  },

  // Preset configuration using the 'classic' preset
  presets: [
    [
      'classic',
      {
        // Configuration for the docs portion of the site
        docs: {
          path: './docs', // Path to the documentation markdown files
          routeBasePath: '/docs/', // URL route for the documentation section
          sidebarPath: require.resolve('./sidebars.ts'), // Path to the sidebar configuration
          // URL for the "edit this page" feature
          // editUrl: 'https://github.com/facebook/docusaurus/tree/main/packages/create-docusaurus/te
        },
        // Configuration for the blog portion of the site
        blog: {
          showReadingTime: true, // Shows estimated reading time for blog posts
          // URL for the "edit this page" feature for blog posts
          // editUrl: 'https://github.com/facebook/docusaurus/tree/main/packages/create-docusaurus/te
        },
```

```
import type {SidebarsConfig} from '@docusaurus/plugin-content-docs';

/**
 * Creating a sidebar enables you to:
 - create an ordered group of docs
 - render a sidebar for each doc of that group
 - provide next/previous navigation

 The sidebars can be generated from the filesystem, or explicitly defined here.

 Create as many sidebars as you want.
 */
const sidebars: SidebarsConfig = {
  // By default, Docusaurus generates a sidebar from the docs folder structure
  tutorialSidebar: [{type: 'autogenerated', dirName: '.'}],

  // But you can create a sidebar manually
  /*
  tutorialSidebar: [
    'intro',
    'hello',
    {
      type: 'category',
      label: 'Tutorial',
      items: ['tutorial-basics/create-a-document'],
    },
  ],
   */
};

export default sidebars;
```

Getting Started with Create React App

This project was bootstrapped with [Create React App](https://github.com/facebook/create-react-ap

Available Scripts

In the project directory, you can run:

`npm start`

Runs the app in the development mode.\
Open http://localhost:3000 to view it in your browser.

The page will reload when you make changes.\
You may also see any lint errors in the console.

`npm test`

Launches the test runner in the interactive watch mode.\
See the section about [running tests](https://facebook.github.io/create-react-app/docs/running-tests

`npm run build`

Builds the app for production to the `build` folder.\
It correctly bundles React in production mode and optimizes the build for the best performance.

The build is minified and the filenames include the hashes.\
Your app is ready to be deployed!

See the section about [deployment](https://facebook.github.io/create-react-app/docs/deployment) fo

`npm run eject`

Note: this is a one-way operation. Once you `eject`, you can't go back!

If you aren't satisfied with the build tool and configuration choices, you can `eject` at any time. This

Instead, it will copy all the configuration files and the transitive dependencies (webpack, Babel, ESL

You don't have to ever use `eject`. The curated feature set is suitable for small and middle deploym

Learn More

You can learn more in the [Create React App documentation](https://facebook.github.io/create-reac

To learn React, check out the [React documentation](https://reactjs.org/).

```
{
  // This file is not used in compilation. It is here just for a nice editor experience.
  "extends": "@docusaurus/tsconfig",
  "compilerOptions": {
    "baseUrl": "."
  }
}
```

---
sidebar_position: 999

DSPy Cheatsheet

This page will contain snippets for frequent usage patterns.

DSPy DataLoaders

Import and initializing a DataLoader Object:

```python
import dspy
from dspy.datasets import DataLoader

dl = DataLoader()
```

Loading from HuggingFace Datasets

```python
code_alpaca = dl.from_huggingface("HuggingFaceH4/CodeAlpaca_20K")
```

You can access the dataset of the splits by calling key of the corresponding split:

```python
train_dataset = code_alpaca['train']
test_dataset = code_alpaca['test']
```

Loading specific splits from HuggingFace

You can also manually specify splits you want to include as a parameters and it'll return a dictionary

```python
code_alpaca = dl.from_huggingface(
    "HuggingFaceH4/CodeAlpaca_20K",
    split = ["train", "test"],
)

print(f"Splits in dataset: {code_alpaca.keys()}")
```

If you specify a single split then dataloader will return a List of `dspy.Example` instead of dictionary

---
sidebar_position: 1

API References

Welcome to the API References for DSPy! This is where you'll find easy-to-understand information

---
sidebar_position: 998

FAQs

Is DSPy right for me? DSPy vs. other frameworks

The **DSPy** philosophy and abstraction differ significantly from other libraries and frameworks, so

DSPy vs. thin wrappers for prompts (OpenAI API, MiniChain, basic templating) In other words: _

DSPy vs. application development libraries like LangChain, LlamaIndex LangChain and LlamaIn

DSPy vs. generation control libraries like Guidance, LMQL, RELM, Outlines These are all excitin

Basic Usage

How should I use DSPy for my task? We wrote a [eight-step guide](https://dspy-docs.vercel.app

How do I convert my complex prompt into a DSPy pipeline? See the same answer above.

What do DSPy optimizers tune? Or, _what does compiling actually do?_ Each optimizer is differ

Other FAQs. We welcome PRs to add formal answers to each of these here. You will find the answ

- **How do I get multiple outputs?**

You can specify multiple output fields. For the short-form signature, you can list multiple outputs as

- **How can I work with long responses?**

You can specify the generation of long responses as a `dspy.OutputField`. To ensure comprehensi

- **How can I ensure that DSPy doesn't strip new line characters from my inputs or outputs?**

DSPy uses [Signatures](https://dspy-docs.vercel.app/docs/deep-dive/signature/understanding-sign

```python
class UnstrippedSignature(dspy.Signature):
    """Enter some information for the model here."""

    title = dspy.InputField()
    object = dspy.InputField(format=str)
    result = dspy.OutputField(format=str)
```

---
sidebar_position: 2

Signatures

When we assign tasks to LMs in DSPy, we specify the behavior we need as a Signature.

A signature is a declarative specification of input/output behavior of a DSPy module. Signatures

You're probably familiar with function signatures, which specify the input and output arguments and

- While typical function signatures just _describe_ things, DSPy Signatures _define and control the

- The field names matter in DSPy Signatures. You express semantic roles in plain English: a `quest

Why should I use a DSPy Signature?

tl;dr For modular and clean code, in which LM calls can be optimized into high-quality prompts (

Long Answer: Most people coerce LMs to do tasks by hacking long, brittle prompts. Or by collec

Writing signatures is far more modular, adaptive, and reproducible than hacking at prompts or finet

Inline DSPy Signatures

Signatures can be defined as a short string, with argument names that define semantic roles for inp

1. Question Answering: `"question -> answer"`

2. Sentiment Classification: `"sentence -> sentiment"`

3. Summarization: `"document -> summary"`

Your signatures can also have multiple input/output fields.

4. Retrieval-Augmented Question Answering: `"context, question -> answer"`

5. Multiple-Choice Question Answering with Reasoning: `"question, choices -> reasoning, selection

Tip: For fields, any valid variable names work! Field names should be semantically meaningful,

---
sidebar_position: 5

Data

DSPy is a machine learning framework, so working in it involves training sets, development sets, a

For each example in your data, we distinguish typically between three types of values: the inputs, t

How much data do I need and how do I collect data for my task?

Concretely, you can use DSPy optimizers usefully with as few as 10 example inputs, but having 50

How can you get examples like these? If your task is extremely unusual, please invest in preparing

However, chances are that your task is not actually that unique. You can almost always find somew

If there's data whose licenses are permissive enough, we suggest you use them. Otherwise, you ca

DSPy `Example` objects

The core data type for data in DSPy is `Example`. You will use **Examples** to represent items in y

DSPy **Examples** are similar to Python `dict`s but have a few useful utilities. Your DSPy modules

When you use DSPy, you will do a lot of evaluation and optimization runs. Your individual datapoin

```python
qa_pair = dspy.Example(question="This is a question?", answer="This is an answer.")

print(qa_pair)
print(qa_pair.question)
print(qa_pair.answer)
```
Output:
```text
Example({'question': 'This is a question?', 'answer': 'This is an answer.'}) (input_keys=None)
This is a question?
This is an answer.
```

Examples can have any field keys and any value types, though usually values are strings.

```text
object = Example(field1=value1, field2=value2, field3=value3, ...)
```

---
sidebar_position: 5

Metrics

DSPy is a machine learning framework, so you must think about your **automatic metrics** for eva

What is a metric and how do I define a metric for my task?

A metric is just a function that will take examples from your data and take the output of your system

For simple tasks, this could be just "accuracy" or "exact match" or "F1 score". This may be the case

However, for most applications, your system will output long-form outputs. There, your metric shou

Getting this right on the first try is unlikely, but you should start with something simple and iterate.

Simple metrics

A DSPy metric is just a function in Python that takes `example` (e.g., from your training or dev set)

Your metric should also accept an optional third argument called `trace`. You can ignore this for a n

Here's a simple example of a metric that's comparing `example.answer` and `pred.answer`. This pa

```python
def validate_answer(example, pred, trace=None):
    return example.answer.lower() == pred.answer.lower()
```

Some people find these utilities (built-in) convenient:

- `dspy.evaluate.metrics.answer_exact_match`
- `dspy.evaluate.metrics.answer_passage_match`

Your metrics could be more complex, e.g. check for multiple properties. The metric below will return

```python
def validate_context_and_answer(example, pred, trace=None):
    # check the gold label and the predicted answer are the same
    answer_match = example.answer.lower() == pred.answer.lower()

    # check the predicted answer comes from one of the retrieved contexts
    context_match = any((pred.answer.lower() in c) for c in pred.context)
```

```
{
  "label": "Local Language Model Clients",
  "position": 6,
  "link": {
    "type": "generated-index",
    "description": "DSPy supports various methods including `built-in wrappers`, `server integration
  }
}
```

---
sidebar_position: 1

Using DSPy in 8 Steps

Using DSPy well for solving a new task is just doing good machine learning with LMs.

What this means is that it's an iterative process. You make some initial choices, which will be sub-c

As we discuss below, you will define your task and the metrics you want to maximize, and prepare

1) Define your task.

You cannot use DSPy well if you haven't defined the problem you're trying to solve.

Expected Input/Output Behavior: Are you trying to build a chatbot over your data? A code assis

It's often useful to come up with just 3-4 examples of the inputs and outputs of your program (e.g.,

If you need help thinking about your task, we recently created a [Discord server](https://discord.gg/

Quality and Cost Specs: You probably don't have infinite budget. Your final system can't be too

Take this as an opportunity to guess what kind of language model you'd like to use. Maybe GPT-3.

2) Define your pipeline.

What should your DSPy program do? Can it just be a simple chain-of-thought step? Or do you nee

Is there a typical workflow for solving your problem in multiple well-defined steps? Or do you want a

Think about this space but always start simple. Almost every task should probably start with just a s

Then write your (initial) DSPy program. Again: start simple, and let the next few steps guide any co

3) Explore a few examples.

By this point, you probably have a few examples of the task you're trying to solve.

Run them through your pipeline. Consider using a large and powerful LM at this point, or a couple

At this point, you're still using your pipeline zero-shot, so it will be far from perfect. DSPy will help y

Record the interesting (both easy and hard) examples you try: even if you don't have labels, simply

---
sidebar_position: 2

Language Models

The most powerful features in DSPy revolve around algorithmically optimizing the prompts (or weig

Let's first make sure you can set up your language model. DSPy support clients for many remote a

Setting up the LM client.

You can just call the constructor that connects to the LM. Then, use `dspy.configure` to declare this

For example, to use OpenAI language models, you can do it as follows.

```python
gpt3_turbo = dspy.OpenAI(model='gpt-3.5-turbo-1106', max_tokens=300)
dspy.configure(lm=gpt3_turbo)
```

Directly calling the LM.

You can simply call the LM with a string to give it a raw prompt, i.e. a string.

```python
gpt3_turbo("hello! this is a raw prompt to GPT-3.5")
```

Output:
```text
['Hello! How can I assist you today?']
```

This is almost never the recommended way to interact with LMs in DSPy, but it is allowed.

Using the LM with DSPy signatures.

You can also use the LM via DSPy [`signature` (input/output spec)](https://dspy-docs.vercel.app/do

```python
# Define a module (ChainOfThought) and assign it a signature (return an answer, given a question)
qa = dspy.ChainOfThought('question -> answer')

# Run with the default LM configured with `dspy.configure` above.
response = qa(question="How many floors are in the castle David Gregory inherited?")
print(response.answer)
```

DSPy Assertions

Introduction

Language models (LMs) have transformed how we interact with machine learning, offering vast cap

To address this, we introduce DSPy Assertions, a feature within the DSPy framework designed to a

dspy.Assert and dspy.Suggest API

We introduce two primary constructs within DSPy Assertions:

- **`dspy.Assert`**:
  - **Parameters**:
    - `constraint (bool)`: Outcome of Python-defined boolean validation check.
    - `msg (Optional[str])`: User-defined error message providing feedback or correction guidance.
    - `backtrack (Optional[module])`: Specifies target module for retry attempts upon constraint failur
  - **Behavior**: Initiates retry upon failure, dynamically adjusting the pipeline's execution. If failure

- **`dspy.Suggest`**:
  - **Parameters**: Similar to `dspy.Assert`.
  - **Behavior**: Encourages self-refinement through retries without enforcing hard stops. Logs failu

- **dspy.Assert vs. Python Assertions**: Unlike conventional Python `assert` statements that termin

Specifically, when a constraint is not met:

- Backtracking Mechanism: An under-the-hood backtracking is initiated, offering the model a chanc
- Dynamic Signature Modification: internally modifying your DSPy program·s Signature by adding th
    - Past Output: your model's past output that did not pass the validation_fn
    - Instruction: your user-defined feedback message on what went wrong and what possibly to fix

If the error continues past the `max_backtracking_attempts`, then `dspy.Assert` will halt the pipelin

- **dspy.Suggest vs. dspy.Assert**: `dspy.Suggest` on the other hand offers a softer approach. It m

- **`dspy.Suggest`** are best utilized as "helpers" during the evaluation phase, offering guidance an
- **`dspy.Assert`** are recommended during the development stage as "checkers" to ensure the LM

Use Case: Including Assertions in DSPy Programs

We start with using an example of a multi-hop QA SimplifiedBaleen pipeline as defined in the intro

```python
class SimplifiedBaleen(dspy.Module):
    def __init__(self, passages_per_hop=2, max_hops=2):
```

Typed Predictors

In DSPy Signatures, we have `InputField` and `OutputField` that define the nature of inputs and ou

Pydantic `BaseModel` is a great way to enforce type constraints on the fields, but it is not directly c

Executing Typed Predictors

Using Typed Predictors is not too different than any other module with the minor additions of type h

Defining Input and Output Models

Let's take a simple task as an example i.e. given the `context` and `query`, the LLM should return a

```python
from pydantic import BaseModel, Field

class Input(BaseModel):
    context: str = Field(description="The context for the question")
    query: str = Field(description="The question to be answered")

class Output(BaseModel):
    answer: str = Field(description="The answer for the question")
    confidence: float = Field(ge=0, le=1, description="The confidence score for the answer")
```

As you can see, we can describe the attributes by defining a simple Signature that takes in the inpu

Creating Typed Predictor

A Typed Predictor needs a Typed Signature, which extends a `dspy.Signature` with the addition of

```python
class QASignature(dspy.Signature):
    """Answer the question based on the context and query provided, and on the scale of 10 tell how

    input: Input = dspy.InputField()
    output: Output = dspy.OutputField()
```

Now that we have the `QASignature`, let's define a Typed Predictor that executes this Signature wh

```python
predictor = dspy.TypedPredictor(QASignature)
```

Similar to other modules, we pass the `QASignature` to `dspy.TypedPredictor` which enforces the t

---
sidebar_position: 3

Modules

A **DSPy module** is a building block for programs that use LMs.

- Each built-in module abstracts a **prompting technique** (like chain of thought or ReAct). Crucial

- A DSPy module has **learnable parameters** (i.e., the little pieces comprising the prompt and the

- Multiple modules can be composed into bigger modules (programs). DSPy modules are inspired

How do I use a built-in module, like `dspy.Predict` or `dspy.ChainOfThought`?

Let's start with the most fundamental module, `dspy.Predict`. Internally, all other DSPy modules are

We'll assume you are already at least a little familiar with [DSPy signatures](https://dspy-docs.verce

To use a module, we first **declare** it by giving it a signature. Then we **call** the module with the

```python
sentence = "it's a charming and often affecting journey."  # example from the SST-2 dataset.

# 1) Declare with a signature.
classify = dspy.Predict('sentence -> sentiment')

# 2) Call with input argument(s).
response = classify(sentence=sentence)

# 3) Access the output.
print(response.sentiment)
```

Output:

```text
Positive
```

When we declare a module, we can pass configuration keys to it.

Below, we'll pass `n=5` to request five completions. We can also pass `temperature` or `max_len`,

Let's use `dspy.ChainOfThought`. In many cases, simply swapping `dspy.ChainOfThought` in place

```python
```

---
sidebar_position: 6

Optimizers (formerly Teleprompters)

A **DSPy optimizer** is an algorithm that can tune the parameters of a DSPy program (i.e., the pro

There are many built-in optimizers in DSPy, which apply vastly different strategies. A typical DSPy

- Your **DSPy program**. This may be a single module (e.g., `dspy.Predict`) or a complex multi-mo

- Your **metric**. This is a function that evaluates the output of your program, and assigns it a scor

- A few **training inputs**. This may be very small (i.e., only 5 or 10 examples) and incomplete (onl

If you happen to have a lot of data, DSPy can leverage that. But you can start small and get strong

Note: Formerly called **DSPy Teleprompters**. We are making an official name update, which w

What does a DSPy Optimizer tune? **How** does it tune them?

Traditional deep neural networks (DNNs) can be optimized with gradient descent, given a loss func

DSPy programs consist of multiple calls to LMs, stacked together as [DSPy modules]. Each DSPy m

Given a metric, DSPy can optimize all of these three with multi-stage optimization algorithms. Thes

In many cases, we found that compiling leads to better prompts than human writing. Not because D

What DSPy Optimizers are currently available?

<!-- The following diagram was generated by: -->
<!-- 1. Running symilar on the teleprompter module to extract the python hierarchy as a Graphviz d
<!-- 2. Hand-editing the resulting dot file to remove classes that are not teleprompters/optimizers (e
<!-- 3. Using dot to compile the `.dot` file into a PNG -->
<!-- Robert Goldman [2024/05/11:rpg] -->

[Subclasses of Teleprompter](figures/teleprompter-classes.png)

All of these can be accessed via `from dspy.teleprompt import *`.

Automatic Few-Shot Learning

These optimizers extend the signature by automatically generating and including **optimized** exa

```
{
  "label": "Local Language Model Clients",
  "position": 6,
  "link": {
    "type": "generated-index",
    "description": "DSPy supports various methods including `built-in wrappers`, `server integration
  }
}
```

Build & Release Workflow Implementation

The [build_and_release](https://github.com/stanfordnlp/dspy/blob/main/.github/workflows/build_and

Overview

At a high level, the workflow works as follows:

1. Maintainer of the repo pushes a tag following [semver](https://semver.org/) versioning for the new
2. This triggers the github action which extracts the tag (the version)
3. Builds and publishes a release on [test-pypi](https://test.pypi.org/project/dspy-ai-test/)
4. Uses the test-pypi release to run build_utils/tests/intro.py with the new release as an integration t
5. Assuming the test runs successfully, it pushes a release to [pypi](https://pypi.org/project/dspy-ai/
6. (Currently manual) the user creates a release and includes release notes, as described in docs/d

Implementation Details

The workflow executes a series of jobs in sequence:
- extract-tag
- build-and-publish-test-pypi
- test-intro-script
- build-and-publish-pypi

extract-tag
Extracts the tag pushed to the commit. This tag is expected to be the version of the new deploymen

build-and-publish-test-pypi
Builds and publishes the package to test-pypi.
1. Determines the version that should be deployed to test-pypi. There may be an existing deployme
    1. Load the releases on test-pypi
    1. Check if there is a release matching our current tag
        1. If not, create a release with the current tag
        1. If it exists, oad the latest published version (this will either be the version with the tag itself,
1. Updates the version placeholder in [setup.py](https://github.com/stanfordnlp/dspy/blob/main/setu
1. Updates the version placeholder in [pyproject.toml](https://github.com/stanfordnlp/dspy/blob/mai
1. Updates the package name placeholder in [setup.py](https://github.com/stanfordnlp/dspy/blob/m
1. Updates the package name placeholder in [pyproject.toml](https://github.com/stanfordnlp/dspy/b
1. Builds the binary wheel
1. Publishes the package to test-pypi.

test-intro-script
Runs the pytest containing the intro script as an integration test using the package published to tes
1. Uses a loop to install the version just published to test-pypi as sometimes there is a race conditio
2. Runs the test to ensure the package is working as expected.
3. If this fails, the workflow fails and the maintainer needs to make a fix and delete and then recreat

Release Checklist

* [ ] On `main` Create a git tag with pattern X.Y.Z where X, Y, and Z follow the [semver pattern](http
    * ```bash
      git tag X.Y.Z
      git push origin --tags
      ```
    * This will trigger the github action to build and release the package.
* [ ] Confirm the tests pass and the package has been published to pypi.
    * If the tests fail, you can remove the tag from your local and github repo using:
    ```bash
    git push origin --delete X.Y.Z # Delete on Github
    git tag -d X.Y.Z # Delete locally
    ```
    * Fix the errors and then repeat the steps above to recreate the tag locally and push to Github to
    * Note that the github action takes care of incrementing the release version on test-pypi automat
* [ ] [Create a release](https://docs.github.com/en/repositories/releasing-projects-on-github/managi
* [ ] Add release notes. You can make use of [automatically generated release notes](https://docs.g
* If creating a new release for major or minor version:
    * [ ] Create a new release branch with the last commit and name it 'release/X.Y`
    * [ ] [Update the default branch](https://docs.github.com/en/organizations/managing-organization

Prerequisites

The automation requires a [trusted publisher](https://docs.pypi.org/trusted-publishers/) to be set up
    ```

```
{
  "label": "Local Language Model Clients",
  "position": 6,
  "link": {
    "type": "generated-index",
    "description": "DSPy supports various methods including `built-in wrappers`, `server integration
  }
}
```

```
{
  "label": "Local Language Model Clients",
  "position": 6,
  "link": {
    "type": "generated-index",
    "description": "DSPy supports various methods including `built-in wrappers`, `server integration
  }
}
```

```
{
  "label": "Local Language Model Clients",
  "position": 6,
  "link": {
    "type": "generated-index",
    "description": "DSPy supports various methods including `built-in wrappers`, `server integration
  }
}
```

```
{
  "label": "Local Language Model Clients",
  "position": 6,
  "link": {
    "type": "generated-index",
    "description": "DSPy supports various methods including `built-in wrappers`, `server integration
  }
}
```

```
{
  "label": "Local Language Model Clients",
  "position": 6,
  "link": {
    "type": "generated-index",
    "description": "DSPy supports various methods including `built-in wrappers`, `server integration
  }
}
```

```
{
  "label": "Local Language Model Clients",
  "position": 6,
  "link": {
    "type": "generated-index",
    "description": "DSPy supports various methods including `built-in wrappers`, `server integration
  }
}
```

```
{
  "label": "Local Language Model Clients",
  "position": 6,
  "link": {
    "type": "generated-index",
    "description": "DSPy supports various methods including `built-in wrappers`, `server integration
  }
}
```

Understanding Typed Predictors

Why use a Typed Predictor?

How to use a Typed Predictor?

Prompt of Typed Predictors

How Typed Predictors work?

```
{
  "label": "Local Language Model Clients",
  "position": 6,
  "link": {
    "type": "generated-index",
    "description": "DSPy supports various methods including `built-in wrappers`, `server integration
  }
}
```

Functional Typed Predictors

Typed Predictors as Decorators

Functional Typed Predictors in `dspy.Module`

How Functional Typed Predictors work?

```
{
  "label": "Local Language Model Clients",
  "position": 6,
  "link": {
    "type": "generated-index",
    "description": "DSPy supports various methods including `built-in wrappers`, `server integration
  }
}
```

---
sidebar_position: 99998

Community Examples

The DSPy team believes complexity has to be justified. We take this seriously: we never release a

There's a bunch of examples in the `examples/` directory and in the top-level directory. We welcom

You can find other examples tweeted by [@lateinteraction](https://twitter.com/lateinteraction) on Tw

Some other examples (not exhaustive, feel free to add more via PR):

- Applying DSPy Assertions
  - [Long-form Answer Generation with Citations, by Arnav Singhvi](https://colab.research.google.co
  - [Generating Answer Choices for Quiz Questions, by Arnav Singhvi](https://colab.research.google
  - [Generating Tweets for QA, by Arnav Singhvi](https://colab.research.google.com/github/stanford
- [Compiling LCEL runnables from LangChain in DSPy](https://github.com/stanfordnlp/dspy/blob/ma
- [AI feedback, or writing LM-based metrics in DSPy](https://github.com/stanfordnlp/dspy/blob/main
- [DSPy Optimizers Benchmark on a bunch of different tasks, by Michael Ryan](https://github.com/s
- [Indian Languages NLI with gains due to compiling by Saiful Haq](https://github.com/saifulhaq95/D
- [Sophisticated Extreme Multi-Class Classification, IReRa, by Karel D'Oosterlinck](https://github.co
- [DSPy on BIG-Bench Hard Example, by Chris Levy](https://drchrislevy.github.io/posts/dspy/dspy.h
- [Using Ollama with DSPy for Mistral (quantized) by @jrknox1977](https://gist.github.com/jrknox19
- [Using DSPy, "The Unreasonable Effectiveness of Eccentric Automatic Prompts" (paper) by VMw

There are also recent cool examples at [Weaviate's DSPy cookbook](https://github.com/weaviate/re

---
sidebar_position: 2
---

[02] Multi-Hop Question Answering

A single search query is often not enough for complex QA tasks. For instance, an example within `H

The standard approach for this challenge in retrieval-augmented NLP literature is to build multi-hop

Configuring LM and RM

We'll start by setting up the language model (LM) and retrieval model (RM), which **DSPy** suppor

In this notebook, we'll work with GPT-3.5 (`gpt-3.5-turbo`) and the `ColBERTv2` retriever (a free se

```python
import dspy

turbo = dspy.OpenAI(model='gpt-3.5-turbo')
colbertv2_wiki17_abstracts = dspy.ColBERTv2(url='http://20.102.90.50:2017/wiki17_abstracts')

dspy.settings.configure(lm=turbo, rm=colbertv2_wiki17_abstracts)
```

Loading the Dataset

For this tutorial, we make use of the mentioned `HotPotQA` dataset, a collection of complex questio

```python
from dspy.datasets import HotPotQA

# Load the dataset.
dataset = HotPotQA(train_seed=1, train_size=20, eval_seed=2023, dev_size=50, test_size=0)

# Tell DSPy that the 'question' field is the input. Any other fields are labels and/or metadata.
trainset = [x.with_inputs('question') for x in dataset.train]
devset = [x.with_inputs('question') for x in dataset.dev]

len(trainset), len(devset)
```
Output:
```text
(20, 50)
```

Building Signature

---
sidebar_position: 1
---

[01] RAG: Retrieval-Augmented Generation

Retrieval-augmented generation (RAG) is an approach that allows LLMs to tap into a large corpus

RAG ensures LLMs can dynamically utilize real-time knowledge even if not originally trained on the

Configuring LM and RM

We'll start by setting up the language model (LM) and retrieval model (RM), which **DSPy** suppor

In this notebook, we'll work with GPT-3.5 (`gpt-3.5-turbo`) and the `ColBERTv2` retriever (a free se

```python
import dspy

turbo = dspy.OpenAI(model='gpt-3.5-turbo')
colbertv2_wiki17_abstracts = dspy.ColBERTv2(url='http://20.102.90.50:2017/wiki17_abstracts')

dspy.settings.configure(lm=turbo, rm=colbertv2_wiki17_abstracts)
```

Loading the Dataset

For this tutorial, we make use of the `HotPotQA` dataset, a collection of complex question-answer p

```python
from dspy.datasets import HotPotQA

# Load the dataset.
dataset = HotPotQA(train_seed=1, train_size=20, eval_seed=2023, dev_size=50, test_size=0)

# Tell DSPy that the 'question' field is the input. Any other fields are labels and/or metadata.
trainset = [x.with_inputs('question') for x in dataset.train]
devset = [x.with_inputs('question') for x in dataset.dev]

len(trainset), len(devset)
```
Output:
```text
(20, 50)
```

```
{
  "label": "Local Language Model Clients",
  "position": 6,
  "link": {
    "type": "generated-index",
    "description": "DSPy supports various methods including `built-in wrappers`, `server integration
  }
}
```

---
sidebar_position: 99999

Additional Resources

Tutorials

Level	**Tutorial**	**Run in Colab**	**Description**
Beginner	[**Getting Started**](https://github.com/stanfordnlp/dspy/blob/main/intro.ipynb)	[<img
Beginner	[**Minimal Working Example**](https://dspy-docs.vercel.app/docs/quick-start/minimal-e
Beginner	[**Compiling for Tricky Tasks**](https://github.com/stanfordnlp/dspy/blob/main/example
Beginner	[**Local Models & Custom Datasets**](https://github.com/stanfordnlp/dspy/blob/main/sl
Intermediate	[**The DSPy Paper**](https://arxiv.org/abs/2310.03714)	N/A	Sections 3, 5, 6, and
Intermediate	[**DSPy Assertions**](https://arxiv.org/abs/2312.13382)	[<img align="center" src="
Intermediate	[**Finetuning for Complex Programs**](https://twitter.com/lateinteraction/status/171
Advanced	[**Information Extraction**](https://twitter.com/KarelDoostrlnck/status/1724991014207

Resources

- [DSPy talk at ScaleByTheBay Nov 2023](https://www.youtube.com/watch?v=Dt3H2ninoeY).
- [DSPy webinar with MLOps Learners](https://www.youtube.com/watch?v=im7bCLW2aM4), a bit l
- Hands-on Overviews of DSPy by the community: [DSPy Explained! by Connor Shorten](https://ww
- Interviews: [Weaviate Podcast in-person](https://www.youtube.com/watch?v=CDung1LnLbY), and
- **Tracing in DSPy** with Arize Phoenix: [Tutorial for tracing your prompts and the steps of your DS
- **Tracing & Optimization Tracking in DSPy** with Parea AI: [Tutorial on tracing & evaluating a DS

---
sidebar_position: 1

API References

Welcome to the API References for DSPy! This is where you'll find easy-to-understand information

```
import inspect
import uuid
from typing import Any

import dsp
import dspy

###################### Assertion Helpers ######################


def _build_error_msg(feedback_msgs):
    """Build an error message from a list of feedback messages."""
    return "\n".join([msg for msg in feedback_msgs])


###################### Assertion Exceptions ######################


class DSPyAssertionError(AssertionError):
    """Custom exception raised when a DSPy `Assert` fails."""

    def __init__(
        self,
        id: str,
        msg: str,
        target_module: Any = None,
        state: Any = None,
        is_metric: bool = False,
    ) -> None:
        super().__init__(msg)
        self.id = id
        self.msg = msg
        self.target_module = target_module
        self.state = state
        self.is_metric = is_metric


class DSPySuggestionError(AssertionError):
    """Custom exception raised when a DSPy `Suggest` fails."""

    def __init__(
        self,
        id: str,
        msg: str,
        target_module: Any = None,
        state: Any = None,
        is_metric: bool = False,
```

dspy.ChainOfThoughtWithHint

Constructor

The constructor initializes the `ChainOfThoughtWithHint` class and sets up its attributes, inheriting

```python
class ChainOfThoughtWithHint(Predict):
    def __init__(self, signature, rationale_type=None, activated=True, **config):
        super().__init__(signature, **config)
        self.activated = activated
        signature = self.signature

        *keys, last_key = signature.fields.keys()
        rationale_type = rationale_type or dspy.OutputField(
            prefix="Reasoning: Let's think step by step in order to",
            desc="${produce the " + last_key + "}. We ...",
        )
        self.extended_signature1 = self.signature.insert(-2, "rationale", rationale_type, type_=str)

        DEFAULT_HINT_TYPE = dspy.OutputField()
        self.extended_signature2 = self.extended_signature1.insert(-2, "hint", DEFAULT_HINT_TYPE
```

Parameters:
- `signature` (_Any_): Signature of predictive model.
- `rationale_type` (_dspy.OutputField_, _optional_): Rationale type for reasoning steps. Defaults to
- `activated` (_bool_, _optional_): Flag for activated chain of thought processing. Defaults to `True`.
- `**config` (_dict_): Additional configuration parameters for model.

Method

`forward(self, **kwargs)`

This method extends the parent `Predict` class's forward pass, updating the signature dynamically

Parameters:
- `**kwargs`: Keyword arguments required for prediction.

Returns:
- The result of the `forward` method in the parent `Predict` class.

Examples

```python
#Define a simple signature for basic question answering
class BasicQA(dspy.Signature):
```

dspy.ProgramOfThought

Constructor

The constructor initializes the `ProgramOfThought` class and sets up its attributes. It is designed to

```python
import dsp
import dspy
from ..primitives.program import Module
from ..primitives.python_interpreter import CodePrompt, PythonInterpreter
import re

class ProgramOfThought(Module):
    def __init__(self, signature, max_iters=3):
        ...
```

Parameters:
- `signature` (_dspy.Signature_): Signature defining the input and output fields for the program.
- `max_iters` (_int_, _optional_): Maximum number of iterations for refining the generated code. De

Methods

`_generate_signature(self, mode)`

Generates a signature dict for different modes: `generate`, `regenerate`, and `answer`.

The `generate` mode serves as an initial generation of Python code with the signature (`question ->
The `regenerate` mode serves as a refining generation of Python code, accounting for the past ger
The `answer` mode serves to execute the last stored generated code and output the final answer to

Parameters:
- `mode` (_str_): Mode of operation of Program of Thought.

Returns:
- A dictionary representing the signature for specified mode.

`_generate_instruction(self, mode)`

Generates instructions for code generation based on the mode. This ensures the signature accoun

The instructional modes mirror the signature modes: `generate`, `regenerate`, `answer`

Parameters:
- `mode` (_str_): Mode of operation.

dspy.MultiChainComparison

Constructor

The constructor initializes the `MultiChainComparison` class and sets up its attributes. It inherits fro

The class incorporates multiple student attempt reasonings and concludes with the selected best re

```python
from .predict import Predict
from ..primitives.program import Module

import dsp

class MultiChainComparison(Module):
    def __init__(self, signature, M=3, temperature=0.7, **config):
        super().__init__()

        self.M = M
        signature = Predict(signature).signature
        *keys, last_key = signature.kwargs.keys()

        extended_kwargs = {key: signature.kwargs[key] for key in keys}

        for idx in range(M):
            candidate_type = dsp.Type(prefix=f"Student Attempt #{idx+1}:", desc="${reasoning attempt
            extended_kwargs.update({f'reasoning_attempt_{idx+1}': candidate_type})

        rationale_type = dsp.Type(prefix="Accurate Reasoning: Thank you everyone. Let's now holisti
        extended_kwargs.update({'rationale': rationale_type, last_key: signature.kwargs[last_key]})

        signature = dsp.Template(signature.instructions, **extended_kwargs)
        self.predict = Predict(signature, temperature=temperature, **config)
        self.last_key = last_key
```

Parameters:
- `signature` (_Any_): Signature of predictive model.
- `M` (_int_, _optional_): Number of student reasoning attempts. Defaults to `3`.
- `temperature` (_float_, _optional_): Temperature parameter for prediction. Defaults to `0.7`.
- `**config` (_dict_): Additional configuration parameters for model.

Method

`forward(self, completions, **kwargs)`

This method aggregates all the student reasoning attempts and calls the predict method with exten

dspy.ChainOfThought

Constructor

The constructor initializes the `ChainOfThought` class and sets up its attributes. It inherits from the

Internally, the class initializes the `activated` attribute to indicate if chain of thought processing has

```python
class ChainOfThought(Predict):
    def __init__(self, signature, rationale_type=None, activated=True, **config):
        super().__init__(signature, **config)

        self.activated = activated

        signature = ensure_signature(self.signature)
        *_keys, last_key = signature.output_fields.keys()

        rationale_type = rationale_type or dspy.OutputField(
            prefix="Reasoning: Let's think step by step in order to",
            desc="${produce the " + last_key + "}. We ...",
        )

        self.extended_signature = signature.prepend("rationale", rationale_type, type_=str)
```

Parameters:
- `signature` (_Any_): Signature of predictive model.
- `rationale_type` (_dspy.OutputField_, _optional_): Rationale type for reasoning steps. Defaults to
- `activated` (_bool_, _optional_): Flag for activated chain of thought processing. Defaults to `True`.
- `**config` (_dict_): Additional configuration parameters for model.

Method

`forward(self, **kwargs)`

This method extends the parent `Predict` class' forward pass while updating the signature when ch

Parameters:
- `**kwargs`: Keyword arguments required for prediction.

Returns:
- The result of the `forward` method.

Examples

```python
```

dspy.ReAct

Constructor

The constructor initializes the `ReAct` class and sets up its attributes. It is specifically designed to c

```python
import dsp
import dspy
from ..primitives.program import Module
from .predict import Predict

class ReAct(Module):
    def __init__(self, signature, max_iters=5, num_results=3, tools=None):
        ...
```

Parameters:
- `signature` (_Any_): Signature of the predictive model.
- `max_iters` (_int_, _optional_): Maximum number of iterations for the Thought-Action-Observation
- `num_results` (_int_, _optional_): Number of results to retrieve in the action step. Defaults to `3`.
- `tools` (_List[dspy.Tool]_, _optional_): List of tools available for actions. If none is provided, a defa

Methods

`_generate_signature(self, iters)`

Generates a signature for the Thought-Action-Observation cycle based on the number of iterations

Parameters:
- `iters` (_int_): Number of iterations.

Returns:
- A dictionary representation of the signature.

***

`act(self, output, hop)`

Processes an action and returns the observation or final answer.

Parameters:
- `output` (_dict_): Current output from the Thought.
- `hop` (_int_): Current iteration number.

dspy.Predict

Constructor

The constructor initializes the `Predict` class and sets up its attributes, taking in the `signature` and

```python
class Predict(Parameter):
    def __init__(self, signature, **config):
        self.stage = random.randbytes(8).hex()
        self.signature = signature
        self.config = config
        self.reset()

        if isinstance(signature, str):
            inputs, outputs = signature.split("->")
            inputs, outputs = inputs.split(","), outputs.split(",")
            inputs, outputs = [field.strip() for field in inputs], [field.strip() for field in outputs]

            assert all(len(field.split()) == 1 for field in (inputs + outputs))

            inputs_ = ', '.join([f"`{field}`" for field in inputs])
            outputs_ = ', '.join([f"`{field}`" for field in outputs])

            instructions = f"""Given the fields {inputs_}, produce the fields {outputs_}."""

            inputs = {k: InputField() for k in inputs}
            outputs = {k: OutputField() for k in outputs}

            for k, v in inputs.items():
                v.finalize(k, infer_prefix(k))

            for k, v in outputs.items():
                v.finalize(k, infer_prefix(k))

            self.signature = dsp.Template(instructions, **inputs, **outputs)
```

Parameters:
- `signature` (_Any_): Signature of predictive model.
- `**config` (_dict_): Additional configuration parameters for model.

Method

`__call__(self, **kwargs)`

This method serves as a wrapper for the `forward` method. It allows making predictions using the `

```
{
  "label": "Local Language Model Clients",
  "position": 6,
  "link": {
    "type": "generated-index",
    "description": "DSPy supports various methods including `built-in wrappers`, `server integration
  }
}
```

dspy.Retrieve

Constructor

The constructor initializes the `Retrieve` class and sets up its attributes, taking in `k` number of retr

```python
class Retrieve(Parameter):
    def __init__(self, k=3):
        self.stage = random.randbytes(8).hex()
        self.k = k
```

Parameters:
- `k` (_Any_): Number of retrieval responses

Method

`__call__(self, *args, **kwargs):`

This method serves as a wrapper for the `forward` method. It allows making retrievals on an input q

Parameters:
- `**args`: Arguments required for retrieval.
- `**kwargs`: Keyword arguments required for retrieval.

Returns:
- The result of the `forward` method.

Examples

```python
query='When was the first FIFA World Cup held?'

# Call the retriever on a particular query.
retrieve = dspy.Retrieve(k=3)
topK_passages = retrieve(query).passages

print(f"Top {retrieve.k} passages for question: {query} \n", '-' * 30, '\n')

for idx, passage in enumerate(topK_passages):
    print(f'{idx+1}]', passage, '\n')
```

---
sidebar_position: 1
---

teleprompt.LabeledFewShot

Constructor

The constructor initializes the `LabeledFewShot` class and sets up its attributes, particularly definin

```python
class LabeledFewShot(Teleprompter):
    def __init__(self, k=16):
        self.k = k
```

Parameters:
- `k` (_int_): Number of samples to be used for each predictor. Defaults to 16.

Method

`compile(self, student, *, trainset)`

This method compiles the `LabeledFewShot` instance by configuring the `student` predictor. It assi

Parameters:
- `student` (_Teleprompter_): Student predictor to be compiled.
- `trainset` (_list_): Training dataset for compiling with student predictor.

Returns:
- The compiled `student` predictor with assigned training samples for each predictor or the original

Example

```python
import dspy

#Assume defined trainset
class RAG(dspy.Module):
    def __init__(self, num_passages=3):
        super().__init__()

        #declare retrieval and predictor modules
        self.retrieve = dspy.Retrieve(k=num_passages)
        self.generate_answer = dspy.ChainOfThought(GenerateAnswer)

    #flow for answering questions using predictor and retrieval modules
```

---
sidebar_position: 2
---

teleprompt.BootstrapFewShot

Constructor

The constructor initializes the `BootstrapFewShot` class and sets up parameters for bootstrapping.

```python
class BootstrapFewShot(Teleprompter):
    def __init__(self, metric=None, metric_threshold=None, teacher_settings={}, max_bootstrapped_
        self.metric = metric
        self.teacher_settings = teacher_settings

        self.max_bootstrapped_demos = max_bootstrapped_demos
        self.max_labeled_demos = max_labeled_demos
        self.max_rounds = max_rounds
```

Parameters:
- `metric` (_callable_, _optional_): Metric function to evaluate examples during bootstrapping. Defau
- `metric_threshold` (_float_, _optional_): Score threshold for metric to determine successful examp
- `teacher_settings` (_dict_, _optional_): Settings for teacher predictor. Defaults to empty dictionary
- `max_bootstrapped_demos` (_int_, _optional_): Maximum number of bootstrapped demonstration
- `max_labeled_demos` (_int_, _optional_): Maximum number of labeled demonstrations per predic
- `max_rounds` (_int_, _optional_): Maximum number of bootstrapping rounds. Defaults to 1.
- `max_errors` (_int_): Maximum errors permitted during evaluation. Halts run with the latest error m

Method

`compile(self, student, *, teacher=None, trainset, valset=None)`

This method compiles the BootstrapFewShot instance by performing bootstrapping to refine the stu

This process includes preparing the student and teacher predictors, which involves creating predict

The next stage involves preparing predictor mappings by validating that both the student and teach

The final stage is performing the bootstrapping iterations.

Parameters:
- `student` (_Teleprompter_): Student predictor to be compiled.
- `teacher` (_Teleprompter_, _optional_): Teacher predictor used for bootstrapping. Defaults to `No
- `trainset` (_list_): Training dataset used in bootstrapping.
- `valset` (_list_, _optional_): Validation dataset used in compilation. Defaults to `None`.

---
sidebar_position: 5
---

teleprompt.BootstrapFinetune

Constructor

`__init__(self, metric=None, teacher_settings={}, multitask=True)`

The constructor initializes a `BootstrapFinetune` instance and sets up its attributes. It defines the te

```python
class BootstrapFinetune(Teleprompter):
    def __init__(self, metric=None, teacher_settings={}, multitask=True):
```

Parameters:
- `metric` (_callable_, _optional_): Metric function to evaluate examples during bootstrapping. Defau
- `teacher_settings` (_dict_, _optional_): Settings for teacher predictor. Defaults to empty dictionary
- `multitask` (_bool_, _optional_): Enable multitask fine-tuning. Defaults to `True`.

Method

`compile(self, student, *, teacher=None, trainset, valset=None, target='t5-large', bsize=12, ac

This method first compiles for bootstrapping with the `BootstrapFewShot` teleprompter. It then prep

Parameters:
- `student` (_Predict_): Student predictor to be fine-tuned.
- `teacher` (_Predict_, _optional_): Teacher predictor to help with fine-tuning. Defaults to `None`.
- `trainset` (_list_): Training dataset for fine-tuning.
- `valset` (_list_, _optional_): Validation dataset for fine-tuning. Defaults to `None`.
- `target` (_str_, _optional_): Target model for fine-tuning. Defaults to `'t5-large'`.
- `bsize` (_int_, _optional_): Batch size for training. Defaults to `12`.
- `accumsteps` (_int_, _optional_): Gradient accumulation steps. Defaults to `1`.
- `lr` (_float_, _optional_): Learning rate for fine-tuning. Defaults to `5e-5`.
- `epochs` (_int_, _optional_): Number of training epochs. Defaults to `1`.
- `bf16` (_bool_, _optional_): Enable mixed-precision training with BF16. Defaults to `False`.

Returns:
- `compiled2` (_Predict_): A compiled and fine-tuned `Predict` instance.

Example

```python
#Assume defined trainset
```

```
{
  "label": "Local Language Model Clients",
  "position": 6,
  "link": {
    "type": "generated-index",
    "description": "DSPy supports various methods including `built-in wrappers`, `server integration
  }
}
```

---
sidebar_position: 4
---

teleprompt.BootstrapFewShotWithRandomSearch

Constructor

The constructor initializes the `BootstrapFewShotWithRandomSearch` class and sets up its attribut

```python
class BootstrapFewShotWithRandomSearch(BootstrapFewShot):
    def __init__(self, metric, teacher_settings={}, max_bootstrapped_demos=4, max_labeled_demos
        self.metric = metric
        self.teacher_settings = teacher_settings
        self.max_rounds = max_rounds
        self.num_threads = num_threads
        self.stop_at_score = stop_at_score
        self.metric_threshold = metric_threshold
        self.min_num_samples = 1
        self.max_num_samples = max_bootstrapped_demos
        self.max_errors = max_errors
        self.num_candidate_sets = num_candidate_programs
        self.max_num_traces = 1 + int(max_bootstrapped_demos / 2.0 * self.num_candidate_sets)

        self.max_bootstrapped_demos = self.max_num_traces
        self.max_labeled_demos = max_labeled_demos

        print("Going to sample between", self.min_num_samples, "and", self.max_num_samples, "trac
        print("Going to sample", self.max_num_traces, "traces in total.")
        print("Will attempt to train", self.num_candidate_sets, "candidate sets.")
```

Parameters:
- `metric` (_callable_, _optional_): Metric function to evaluate examples during bootstrapping. Defau
- `teacher_settings` (_dict_, _optional_): Settings for teacher predictor. Defaults to an empty diction
- `max_bootstrapped_demos` (_int_, _optional_): Maximum number of bootstrapped demonstration
- `max_labeled_demos` (_int_, _optional_): Maximum number of labeled demonstrations per predic
- `max_rounds` (_int_, _optional_): Maximum number of bootstrapping rounds. Defaults to 1.
- `num_candidate_programs` (_int_): Number of candidate programs to generate during random se
- `num_threads` (_int_): Number of threads used for evaluation during random search. Defaults to 6
- `max_errors` (_int_): Maximum errors permitted during evaluation. Halts run with the latest error m
- `stop_at_score` (_float_, _optional_): Score threshold for random search to stop early. Defaults to
- `metric_threshold` (_float_, _optional_): Score threshold for metric to determine successful examp

Method

---
sidebar_position: 3
---

teleprompt.Ensemble

Constructor

The constructor initializes the `Ensemble` class and sets up its attributes. This teleprompter is desig

```python
class Ensemble(Teleprompter):
    def __init__(self, *, reduce_fn=None, size=None, deterministic=False):
```

Parameters:
- `reduce_fn` (_callable_, _optional_): Function used to reduce multiple outputs from different progr
- `size` (_int_, _optional_): Number of programs to randomly select for ensembling. If not specified,
- `deterministic` (_bool_, _optional_): Specifies whether ensemble should operate deterministically.

Method

`compile(self, programs)`

This method compiles an ensemble of programs into a single program that when run, can either ran

Parameters:
- `programs` (_list_): List of programs to be ensembled.

Returns:
- `EnsembledProgram` (_Module_): An ensembled version of the input programs.

Example

```python
import dspy
from dspy.teleprompt import Ensemble

# Assume a list of programs
programs = [program1, program2, program3, ...]

# Define Ensemble teleprompter
teleprompter = Ensemble(reduce_fn=dspy.majority, size=2)

# Compile to get the EnsembledProgram
ensembled_program = teleprompter.compile(programs)
```

---
sidebar_position: 6
---

dspy.Anyscale

Usage

```python
lm = dspy.Anyscale(model="mistralai/Mistral-7B-Instruct-v0.1")
```

Constructor

The constructor initializes the base class `LM` and verifies the `api_key` for using Anyscale API.
We expect the following environment variables to be set:
- `ANYSCALE_API_KEY`: API key for Together.
- `ANYSCALE_API_BASE`: API base URL for Together.


```python
class Anyscale(HFModel):
    def __init__(self, model, **kwargs):
```

Parameters:
- `model` (_str_): models hosted on Together.

Methods

Refer to [`dspy.OpenAI`](https://dspy-docs.vercel.app/api/language_model_clients/OpenAI) docum

---
sidebar_position: 11
---

dspy.GoogleVertexAI

This guide provides instructions on how to use the `GoogleVertexAI` class to interact with Google V

Requirements

- Python 3.10 or higher.
- The `vertexai` package installed, which can be installed via pip.
- A Google Cloud account and a configured project with access to Vertex AI.

Installation

Ensure you have installed the `vertexai` package along with other necessary dependencies:

```bash
pip install dspy-ai[google-vertex-ai]
```

Configuration

Before using the `GoogleVertexAI` class, you need to set up access to Google Cloud:

1. Create a project in Google Cloud Platform (GCP).
2. Enable the Vertex AI API for your project.
3. Create authentication credentials and save them in a JSON file.

Usage

Here's an example of how to instantiate the `GoogleVertexAI` class and send a text generation req

```python
from dsp.modules import GoogleVertexAI  # Import the GoogleVertexAI class

# Initialize the class with the model name and parameters for Vertex AI
vertex_ai = GoogleVertexAI(
    model_name="text-bison@002",
    project="your-google-cloud-project-id",
    location="us-central1",
    credentials="path-to-your-service-account-file.json"
)
```

Customizing Requests

---
sidebar_position: 10
---

dspy.CloudflareAI

Usage

```python
lm = dspy.CloudflareAI(model="@hf/meta-llama/meta-llama-3-8b-instruct")
```

Constructor

The constructor initializes the base class `LM` and verifies the `api_key` and `account_id` for using
The following environment variables are expected to be set or passed as arguments:

- `CLOUDFLARE_ACCOUNT_ID`: Account ID for Cloudflare.
- `CLOUDFLARE_API_KEY`: API key for Cloudflare.

```python
class CloudflareAI(LM):
  def __init__(
        self,
        model: str = "@hf/meta-llama/meta-llama-3-8b-instruct",
        account_id: Optional[str] = None,
        api_key: Optional[str] = None,
        system_prompt: Optional[str] = None,
        **kwargs,
    ):
```

Parameters:

- `model` (_str_): Model hosted on Cloudflare. Defaults to `@hf/meta-llama/meta-llama-3-8b-instruc
- `account_id` (_Optional[str]_, _optional_): Account ID for Cloudflare. Defaults to None. Reads fro
- `api_key` (_Optional[str]_, _optional_): API key for Cloudflare. Defaults to None. Reads from envi
- `system_prompt` (_Optional[str]_, _optional_): System prompt to use for generation.

Methods

Refer to [`dspy.OpenAI`](https://dspy-docs.vercel.app/api/language_model_clients/OpenAI) docum

---
sidebar_position: 12
---

dspy.Watsonx

This guide provides instructions on how to use the `Watsonx` class to interact with IBM Watsonx.a

Requirements

- Python 3.10 or higher.
- The `ibm-watsonx-ai` package installed, which can be installed via pip.
- An IBM Cloud account and a Watsonx configured project.

Installation

Ensure you have installed the `ibm-watsonx-ai` package along with other necessary dependencies

Configuration

Before using the `Watsonx` class, you need to set up access to IBM Cloud:

1. Create an IBM Cloud account
2. Enable a Watsonx service from the catalog
3. Create a new project and associate a Watson Machine Learning service instance.
4. Create an IAM authentication credentials and save them in a JSON file.

Usage

Here's an example of how to instantiate the `Watsonx` class and send a generation request:

```python
import dspy

''' Initialize the class with the model name and parameters for Watsonx.ai
    You can choose between many different models:
    * (Mistral) mistralai/mixtral-8x7b-instruct-v01
    * (Meta) meta-llama/llama-3-70b-instruct
    * (IBM) ibm/granite-13b-instruct-v2
    * and many others.
'''
watsonx=dspy.Watsonx(
    model='mistralai/mixtral-8x7b-instruct-v01',
    credentials={
        "apikey": "your-api-key",
        "url": "https://us-south.ml.cloud.ibm.com"
    },
```

---
sidebar_position: 2
---

dspy.AzureOpenAI

Usage

```python
lm = dspy.AzureOpenAI(api_base='...', api_version='2023-12-01-preview', model='gpt-3.5-turbo')
```

Constructor

The constructor initializes the base class `LM` and verifies the provided arguments like the `api_pro

Azure requires that the deployment id of the Azure deployment to be also provided using the argum

```python
class AzureOpenAI(LM):
    def __init__(
        self,
        api_base: str,
        api_version: str,
        model: str = "gpt-3.5-turbo-instruct",
        api_key: Optional[str] = None,
        model_type: Literal["chat", "text"] = None,
        **kwargs,
    ):
```

Parameters:

- `api_base` (str): Azure Base URL.
- `api_version` (str): Version identifier for Azure OpenAI API.
- `api_key` (_Optional[str]_, _optional_): API provider authentication token. Retrieves from `AZURE
- `model_type` (_Literal["chat", "text"]_): Specified model type to use, defaults to 'chat'.
- `azure_ad_token_provider` (_Optional[AzureADTokenProvider]_, _optional_): Pass the bearer tok
- `**kwargs`: Additional language model arguments to pass to the API provider.

Methods

`__call__(self, prompt: str, only_completed: bool = True, return_sorted: bool = False, **kwargs

Retrieves completions from Azure OpenAI Endpoints by calling `request`.

Internally, the method handles the specifics of preparing the request prompt and corresponding pay

---
sidebar_position: 1
---

dspy.OpenAI

Usage

```python
lm = dspy.OpenAI(model='gpt-3.5-turbo')
```

Constructor

The constructor initializes the base class `LM` and verifies the provided arguments like the `api_pro

```python
class OpenAI(LM):
    def __init__(
        self,
        model: str = "text-davinci-002",
        api_key: Optional[str] = None,
        api_provider: Literal["openai"] = "openai",
        model_type: Literal["chat", "text"] = None,
        **kwargs,
    ):
```

Parameters:
- `api_key` (_Optional[str]_, _optional_): API provider authentication token. Defaults to None.
- `api_provider` (_Literal["openai"]_, _optional_): API provider to use. Defaults to "openai".
- `model_type` (_Literal["chat", "text"]_): Specified model type to use.
- `**kwargs`: Additional language model arguments to pass to the API provider.

Methods

`__call__(self, prompt: str, only_completed: bool = True, return_sorted: bool = False, **kwarg

Retrieves completions from OpenAI by calling `request`.

Internally, the method handles the specifics of preparing the request prompt and corresponding pay

After generation, the completions are post-processed based on the `model_type` parameter. If the

Parameters:

```
"""AWS models for LMs."""

from __future__ import annotations

import json
import logging
from abc import abstractmethod
from typing import Any

from dsp.modules.aws_providers import AWSProvider, Bedrock, Sagemaker
from dsp.modules.lm import LM

# Heuristic translating number of chars to tokens
# ~4 chars = 1 token
CHARS2TOKENS: int = 4


class AWSModel(LM):
    """This class adds support for an AWS model.

    It is an abstract class and should not be instantiated directly.
    Instead, use one of the subclasses - AWSMistral, AWSAnthropic, or AWSMeta.
    The subclasses implement the abstract methods _create_body and _call_model
    and work in conjunction with the AWSProvider classes Bedrock and Sagemaker.
    Usage Example:
        bedrock = dspy.Bedrock(region_name="us-west-2")
        bedrock_mixtral = dspy.AWSMistral(bedrock, "mistral.mixtral-8x7b-instruct-v0:1", **kwargs)
        bedrock_haiku = dspy.AWSAnthropic(bedrock, "anthropic.claude-3-haiku-20240307-v1:0", **k
        bedrock_llama2 = dspy.AWSMeta(bedrock, "meta.llama2-13b-chat-v1", **kwargs)

        sagemaker = dspy.Sagemaker(region_name="us-west-2")
        sagemaker_model = dspy.AWSMistral(sagemaker, "<YOUR_ENDPOINT_NAME>", **kwargs)
    """

    def __init__(
        self,
        model: str,
        max_context_size: int,
        max_new_tokens: int,
        **kwargs,
    ) -> None:
        """_summary_.

        Args:
            model (str, optional): An LM name, e.g., a bedrock name or an AWS endpoint.
            max_context_size (int): The maximum context size in tokens.
            max_new_tokens (int): The maximum number of tokens to be sampled from the LM.
```

---
sidebar_position: 12
---

dspy.Snowflake

Usage

```python
import dspy
import os

connection_parameters = {

    "account": os.getenv('SNOWFLAKE_ACCOUNT'),
    "user": os.getenv('SNOWFLAKE_USER'),
    "password": os.getenv('SNOWFLAKE_PASSWORD'),
    "role": os.getenv('SNOWFLAKE_ROLE'),
    "warehouse": os.getenv('SNOWFLAKE_WAREHOUSE'),
    "database": os.getenv('SNOWFLAKE_DATABASE'),
    "schema": os.getenv('SNOWFLAKE_SCHEMA')}

lm = dspy.Snowflake(model="mixtral-8x7b",credentials=connection_parameters)
```

Constructor

The constructor inherits from the base class `LM` and verifies the `credentials` for using Snowflake

```python
class Snowflake(LM):
    def __init__(
        self,
        model,
        credentials,
        **kwargs):
```

Parameters:
- `model` (_str_): model hosted by [Snowflake Cortex](https://docs.snowflake.com/en/user-guide/sn
- `credentials` (_dict_): connection parameters required to initialize a [snowflake snowpark session

Methods

Refer to [`dspy.Snowflake`](https://dspy-docs.vercel.app/api/language_model_clients/Snowflake) d

---
sidebar_position: 5
---

dspy.HFClientVLLM

Usage

```python
lm = dspy.HFClientVLLM(model="meta-llama/Llama-2-7b-hf", port=8080, url="http://localhost")
```

Prerequisites

Refer to the [vLLM Server](https://dspy-docs.vercel.app/api/language_model_clients/HFClientVLLM

Constructor

Refer to [`dspy.TGI`](https://dspy-docs.vercel.app/api/language_model_clients/TGI) documentation

Methods

Refer to [`dspy.OpenAI`](https://dspy-docs.vercel.app/api/language_model_clients/OpenAI) docum

---
sidebar_position: 5
---

dspy.PremAI

[PremAI](https://app.premai.io) is an all-in-one platform that simplifies the process of creating robus

Prerequisites

Refer to the [quick start](https://docs.premai.io/introduction) guide to getting started with the PremA

Usage

Please make sure you have premai python sdk installed. Otherwise you can do it using this comma

```bash
pip install -U premai
```

Here is a quick example on how to use premai python sdk with DSPy

```python
from dspy import PremAI

llm = PremAI(model='mistral-tiny', project_id=123, api_key="your-premai-api-key")
print(llm("what is a large language model"))
```

> Please note: Project ID 123 is just an example. You can find your project ID inside our platform u

Constructor

The constructor initializes the base class `LM` and verifies the `api_key` provided or defined throug

```python
class PremAI(LM):
    def __init__(
        self,
        model: str,
        project_id: int,
        api_key: str,
        base_url: Optional[str] = None,
        session_id: Optional[int] = None,
        **kwargs,
    ) -> None:
```

```
{
  "label": "Local Language Model Clients",
  "position": 6,
  "link": {
    "type": "generated-index",
    "description": "DSPy supports various methods including `built-in wrappers`, `server integration
  }
}
```

---
sidebar_position: 13
---

dspy.You
Wrapper around [You.com's conversational Smart and Research APIs](https://documentation.you.c

Each API endpoint is designed to generate conversational
responses to a variety of query types, including inline citations
and web results when relevant.

Smart Mode:
- Quick, reliable answers for a variety of questions
- Cites the entire web page URL

Research Mode:
- In-depth answers with extensive citations for a variety of questions
- Cites the specific web page snippet relevant to the claim

For more information, check out the documentations at
https://documentation.you.com/api-reference/.

Constructor
```python
You(
    endpoint: Literal["smart", "research"] = "smart",
    ydc_api_key: Optional[str] = None,
)
```

Parameters:
- `endpoint`: You.com conversational endpoints. Choose from "smart" or "research"
- `ydc_api_key`: You.com API key, if `YDC_API_KEY` is not set in the environment

Usage
Obtain a You.com API key from https://api.you.com/.

Export this key to an environment variable `YDC_API_KEY`.

```python
import dspy

# The API key is inferred from the `YDC_API_KEY` environment variable
lm = dspy.You(endpoint="smart")
```

---
sidebar_position: 8
---

dspy.Databricks

Usage
```python
lm = dspy.Databricks(model="databricks-mpt-30b-instruct")
```

Constructor

The constructor inherits from the `GPT3` class and verifies the Databricks authentication credential
We expect the following environment variables to be set:
- `openai.api_key`: Databricks API key.
- `openai.base_url`: Databricks Model Endpoint url

The `kwargs` attribute is initialized with default values for relevant text generation parameters need

```python
class Databricks(GPT3):
    def __init__(
        self,
        model: str,
        api_key: Optional[str] = None,
        api_base: Optional[str] = None,
        model_type: Literal["chat", "text"] = None,
        **kwargs,
    ):
```

Parameters:
- `model` (_str_): models hosted on Databricks.
- `stop` (_List[str]_, _optional_): List of stopping tokens to end generation.
- `api_key` (_Optional[str]_): Databricks API key. Defaults to None
- `api_base` (_Optional[str]_): Databricks Model Endpoint url Defaults to None.
- `model_type` (_Literal["chat", "text", "embeddings"]_): Specified model type to use.
- `**kwargs`: Additional language model arguments to pass to the API provider.

Methods

Refer to [`dspy.OpenAI`](https://dspy-docs.vercel.app/api/language_model_clients/OpenAI) docum

```
"""AWS providers for LMs."""

from abc import ABC, abstractmethod
from typing import Any, Optional

import backoff

try:
    import boto3
    from botocore.exceptions import ClientError
    ERRORS = (ClientError,)

except ImportError:
    ERRORS = (Exception,)


def backoff_hdlr(details):
    """Handler from https://pypi.org/project/backoff/."""
    print(
        "Backing off {wait:0.1f} seconds after {tries} tries "
        "calling function {target} with kwargs "
        "{kwargs}".format(**details),
    )


def giveup_hdlr(details):
    """Wrapper function that decides when to give up on retry."""
    if "max retries" in details.args[0]:
        return False
    return True

class AWSProvider(ABC):
    """This abstract class adds support for AWS model providers such as Bedrock and SageMaker.
    The subclasses such as Bedrock and Sagemaker implement the abstract method _call_model a
    Usage Example:
        bedrock = dspy.Bedrock(region_name="us-west-2")
        bedrock_mixtral = dspy.AWSMistral(bedrock, "mistral.mixtral-8x7b-instruct-v0:1", **kwargs)
        bedrock_haiku = dspy.AWSAnthropic(bedrock, "anthropic.claude-3-haiku-20240307-v1:0", **k
        bedrock_llama2 = dspy.AWSMeta(bedrock, "meta.llama2-13b-chat-v1", **kwargs)

        sagemaker = dspy.Sagemaker(region_name="us-west-2")
        sagemaker_model = dspy.AWSMistral(sagemaker, "<YOUR_ENDPOINT_NAME>", **kwargs)
    """

    def __init__(
        self,
        region_name: str,
```

---
sidebar_position: 7
---

dspy.Together

Usage

```python
lm = dspy.Together(model="mistralai/Mistral-7B-v0.1")
```

Constructor

The constructor initializes the base class `LM` and verifies the `api_key` for using Together API.
We expect the following environment variables to be set:
- `TOGETHER_API_KEY`: API key for Together.
- `TOGETHER_API_BASE`: API base URL for Together.

```python
class Together(HFModel):
    def __init__(self, model, **kwargs):
```

Parameters:
- `model` (_str_): models hosted on Together.
- `stop` (_List[str]_, _optional_): List of stopping tokens to end generation.

Methods

Refer to [`dspy.OpenAI`](https://dspy-docs.vercel.app/api/language_model_clients/OpenAI) docum

dspy.HFClientTGI

Prerequisites

- Docker must be installed on your system. If you don't have Docker installed, you can get it from [h

Setting up the Text-Generation-Inference Server

1. Clone the Text-Generation-Inference repository from GitHub by executing the following comman

   ```
   git clone https://github.com/huggingface/text-generation-inference.git
   ```

2. Change into the cloned repository directory:

   ```
   cd text-generation-inference
   ```

3. Execute the Docker command under the "Get Started" section to run the server:

   ```
   model=meta-llama/Llama-2-7b-hf # set to the specific Hugging Face model ID you wish to use.
   num_shard=2 # set to the number of shards you wish to use.
   volume=$PWD/data # share a volume with the Docker container to avoid downloading weights ev

   docker run --gpus all --shm-size 1g -p 8080:80 -v $volume:/data ghcr.io/huggingface/text-generat
   ```

   This command will start the server and make it accessible at `http://localhost:8080`.

If you want to connect to [Meta Llama 2 models](https://huggingface.co/meta-llama), make sure to

```
   docker run --gpus all --shm-size 1g -p 8080:80 -v $volume:/data -e HUGGING_FACE_HUB_TOK
```

Sending requests to the server

After setting up the text-generation-inference server and ensuring that it displays "Connected" when

Initialize the `HFClientTGI` within your program with the desired parameters. Here is an example ca

   ```python
   lm = dspy.HFClientTGI(model="meta-llama/Llama-2-7b-hf", port=8080, url="http://localhost")
   ```

---
sidebar_position: 9
---

dspy.GROQ

Usage

```python
lm = dspy.GROQ(model='mixtral-8x7b-32768', api_key ="gsk_***" )
```

Constructor

The constructor initializes the base class `LM` and verifies the provided arguments like the `api_key

```python
class GroqLM(LM):
    def __init__(
        self,
        api_key: str,
        model: str = "mixtral-8x7b-32768",
        **kwargs,
    ):
```

Parameters:
- `api_key` str: API provider authentication token. Defaults to None.
- `model` str: model name. Defaults to "mixtral-8x7b-32768' options: ['llama2-70b-4096', 'gemma-7b
- `**kwargs`: Additional language model arguments to pass to the API provider.

Methods

`def __call__(self, prompt: str, only_completed: bool = True, return_sorted: bool = False, **k

Retrieves completions from GROQ by calling `request`.

Internally, the method handles the specifics of preparing the request prompt and corresponding pay

After generation, the generated content look like `choice["message"]["content"]`.

Parameters:
- `prompt` (_str_): Prompt to send to GROQ.
- `only_completed` (_bool_, _optional_): Flag to return only completed responses and ignore compl
- `return_sorted` (_bool_, _optional_): Flag to sort the completion choices using the returned averag

---
sidebar_position: 3
---

dspy.Cohere

Usage

```python
lm = dspy.Cohere(model='command-nightly')
```

Constructor

The constructor initializes the base class `LM` and verifies the `api_key` to set up Cohere request r

```python
class Cohere(LM):
    def __init__(
        self,
        model: str = "command-nightly",
        api_key: Optional[str] = None,
        stop_sequences: List[str] = [],
    ):
```

Parameters:
- `model` (_str_): Cohere pretrained models. Defaults to `command-nightly`.
- `api_key` (_Optional[str]_, _optional_): API provider from Cohere. Defaults to None.
- `stop_sequences` (_List[str]_, _optional_): List of stopping tokens to end generation.

Methods

Refer to [`dspy.OpenAI`](https://dspy-docs.vercel.app/api/language_model_clients/OpenAI) docum

---
sidebar_position: 9
---

dspy.Mistral

Usage

```python
lm = dspy.Mistral(model='mistral-medium-latest', api_key="your-mistralai-api-key")
```

Constructor

The constructor initializes the base class `LM` and verifies the `api_key` provided or defined throug

```python
class Mistral(LM):
    def __init__(
        self,
        model: str = "mistral-medium-latest",
        api_key: Optional[str] = None,
        **kwargs,
    ):
```

Parameters:
- `model` (_str_): Mistral AI pretrained models. Defaults to `mistral-medium-latest`.
- `api_key` (_Optional[str]_, _optional_): API provider from Mistral AI. Defaults to None.
- `**kwargs`: Additional language model arguments to pass to the API provider.

Methods

Refer to [`dspy.OpenAI`](https://dspy-docs.vercel.app/api/language_model_clients/OpenAI) docum

---
sidebar_position: 9
---

retrieve.SnowflakeRM

Constructor

Initialize an instance of the `SnowflakeRM` class, with the option to use `e5-base-v2` or `snowflake

```python
SnowflakeRM(
    snowflake_table_name: str,
    snowflake_credentials: dict,
    k: int = 3,
    embeddings_field: str,
    embeddings_text_field:str,
    embeddings_model: str = "e5-base-v2",
)
```

Parameters:

- `snowflake_table_name (str)`: The name of the Snowflake table containing embeddings.
- `snowflake_credentials (dict)`: The connection parameters needed to initialize a Snowflake Snowp
- `k (int, optional)`: The number of top passages to retrieve. Defaults to 3.
- `embeddings_field (str)`: The name of the column in the Snowflake table containing the embeddin
- `embeddings_text_field (str)`: The name of the column in the Snowflake table containing the pass
- `embeddings_model (str)`: The model to be used to convert text to embeddings

Methods

`forward(self, query_or_queries: Union[str, List[str]], k: Optional[int] = None) -> dspy.Predictio

Search the Snowflake table for the top `k` passages matching the given query or queries, using em

Parameters:

- `query_or_queries` (_Union[str, List[str]]_): The query or list of queries to search for.
- `k` (_Optional[int]_, _optional_): The number of results to retrieve. If not specified, defaults to the

Returns:

- `dspy.Prediction`: Contains the retrieved passages, each represented as a `dotdict` with schema

Quickstart

---
sidebar_position: 3
---

retrieve.ChromadbRM

Constructor

Initialize an instance of the `ChromadbRM` class, with the option to use OpenAI's embeddings or a

```python
ChromadbRM(
    collection_name: str,
    persist_directory: str,
    embedding_function: Optional[EmbeddingFunction[Embeddable]] = OpenAIEmbeddingFunction(
    k: int = 7,
)
```

Parameters:
- `collection_name` (_str_): The name of the chromadb collection.
- `persist_directory` (_str_): Path to the directory where chromadb data is persisted.
- `embedding_function` (_Optional[EmbeddingFunction[Embeddable]]_, _optional_): The function u
- `k` (_int_, _optional_): The number of top passages to retrieve. Defaults to 7.

Methods

`forward(self, query_or_queries: Union[str, List[str]], k: Optional[int] = None) -> dspy.Predictio

Search the chromadb collection for the top `k` passages matching the given query or queries, using

Parameters:
- `query_or_queries` (_Union[str, List[str]]_): The query or list of queries to search for.
- `k` (_Optional[int]_, _optional_): The number of results to retrieve. If not specified, defaults to the v

Returns:
- `dspy.Prediction`: Contains the retrieved passages, each represented as a `dotdict` with schema

Quickstart with OpenAI Embeddings

ChromadbRM have the flexibility from a variety of embedding functions as outlined in the [chromad

```python
from dspy.retrieve.chromadb_rm import ChromadbRM
import os
import openai
from chromadb.utils.embedding_functions import OpenAIEmbeddingFunction
```

---
sidebar_position: 4
---

retrieve.FaissRM

Constructor

Initialize an instance of FaissRM by providing it with a vectorizer and a list of strings

```python
FaissRM(
    document_chunks: List[str],
    vectorizer: dsp.modules.sentence_vectorizer.BaseSentenceVectorizer,
    k: int = 3
)
```

Parameters:
- `document_chunks` (_List[str]_): a list of strings that comprises the corpus to search. You cannot
- `vectorizer` (_dsp.modules.sentence_vectorizer.BaseSentenceVectorizer_, _optional_): If not pro
- `k` (_int_, _optional_): The number of top passages to retrieve. Defaults to 3.

Methods

`forward(self, query_or_queries: Union[str, List[str]]) -> dspy.Prediction`

Search the FaissRM vector database for the top `k` passages matching the given query or queries,

Parameters:
- `query_or_queries` (_Union[str, List[str]]_): The query or list of queries to search for.

Returns:
- `dspy.Prediction`: Contains the retrieved passages, each represented as a `dotdict` with a `long_t

Quickstart with the default vectorizer

The **FaissRM** module provides a retriever that uses an in-memory Faiss vector database. This r

```python
import dspy
from dspy.retrieve.faiss_rm import FaissRM

document_chunks = [
    "The superbowl this year was played between the San Francisco 49ers and the Kanasas City Ch
    "Pop corn is often served in a bowl",
```

---
sidebar_position: 7
---

retrieve.neo4j_rm

Constructor

Initialize an instance of the `Neo4jRM` class.

```python
Neo4jRM(
    index_name: str,
    text_node_property: str,
    k: int = 5,
    retrieval_query: str = None,
    embedding_provider: str = "openai",
    embedding_model: str = "text-embedding-ada-002",
)
```

Environment Variables:

You need to define the credentials as environment variables:

- `NEO4J_USERNAME` (_str_): Specifies the username required for authenticating with the Neo4j

- `NEO4J_PASSWORD` (_str_): Defines the password associated with the `NEO4J_USERNAME`

- `NEO4J_URI` (_str_): Indicates the Uniform Resource Identifier (URI) used to connect to the Neo

- `NEO4J_DATABASE` (_str_, optional): Specifies the name of the database to connect to within th

- `OPENAI_API_KEY` (_str_): Specifies the API key required for authenticiating with OpenAI's serv

Parameters:
- `index_name` (_str_): Specifies the name of the vector index to be used within Neo4j for organizin
- `text_node_property` (_str_, _optional_): Defines the specific property of nodes that will be returne
- `k` (_int_, _optional_): The number of top results to return from the retrieval operation. It defaults t
- `retrieval_query` (_str_, _optional_): A custom query string provided for retrieving data. If not prov
- `embedding_provider` (_str_, _optional_): The name of the service provider for generating embed
- `embedding_model` (_str_, _optional_): The specific embedding model to use from the provider. E

Methods

`forward(self, query: [str], k: Optional[int] = None) -> dspy.Prediction`

---
sidebar_position: 10
---

retrieve.WatsonDiscoveryRM

Constructor

The constructor initializes the `WatsonDiscoveryRM` class instance and sets up the request param

```python
class WatsonDiscoveryRM:
    def __init__(
        self,
        apikey: str,
        url:str,
        version:str,
        project_id:str,
        collection_ids:list=[],
        k: int = 7,
    ):
```

Parameters:
- `apikey` (str): apikey for authentication purposes,
- `url` (str): endpoint URL that includes the service instance ID
- `version` (str): Release date of the version of the API you want to use. Specify dates in YYYY-MM
- `project_id` (str): The Universally Unique Identifier (UUID) of the project.
- `collection_ids` (list): An array containing the collections on which the search will be executed.
- `k` (int, optional): The number of top passages to retrieve. Defaults to 7.

Methods

`forward(self, query_or_queries: Union[str, list[str]], k: Optional[int]= None) -> dspy.Prediction

Search the Watson Discovery collection for the top `k` passages matching the given query or queri

Parameters:
- `query_or_queries` (_Union[str, list[str]]_): The query or list of queries to search for.
- `k` (_Optional[int]_, _optional_): The number of results to retrieve. If not specified, defaults to the

Returns:
- `dspy.Prediction`: Contains the retrieved passages, each represented as a `dotdict` with schema

Quickstart

```python
```

```
{
  "label": "Local Language Model Clients",
  "position": 6,
  "link": {
    "type": "generated-index",
    "description": "DSPy supports various methods including `built-in wrappers`, `server integration
  }
}
```

---
sidebar_position: 2
---

retrieve.AzureCognitiveSearch

Constructor

The constructor initializes an instance of the `AzureCognitiveSearch` class and sets up parameters

```python
class AzureCognitiveSearch:
    def __init__(
        self,
        search_service_name: str,
        search_api_key: str,
        search_index_name: str,
        field_text: str,
        field_score: str, # required field to map with "score" field in dsp framework
    ):
```

Parameters:

- `search_service_name` (_str_): Name of Azure Cognitive Search server.
- `search_api_key` (_str_): API Authentication token for accessing Azure Cognitive Search server.
- `search_index_name` (_str_): Name of search index in the Azure Cognitive Search server.
- `field_text` (_str_): Field name that maps to DSP "content" field.
- `field_score` (_str_): Field name that maps to DSP "score" field.

Methods

Refer to [ColBERTv2](/api/retrieval_model_clients/ColBERTv2) documentation. Keep in mind there

AzureCognitiveSearch supports sending queries and processing the received results, mapping con

Deprecation Notice

This module is scheduled for removal in future releases. Please use the AzureAISearchRM class fr

```
import functools
import os
from typing import List, Optional

import openai

import dspy
from dsp.modules.cache_utils import CacheMemory, NotebookCacheMemory, cache_turn_on
from dsp.utils import dotdict

# Check for necessary libraries and suggest installation if not found.
try:
    import clickhouse_connect
except ImportError:
    raise ImportError(
        "The 'myscale' extra is required to use MyScaleRM. Install it with `pip install dspy-ai[myscale]`
    )

# Verify the compatibility of the OpenAI library version installed.
try:
    major, minor, _ = map(int, openai.__version__.split('.'))
    OPENAI_VERSION_COMPATIBLE = major >= 1 and minor >= 16
except Exception:
    OPENAI_VERSION_COMPATIBLE = False

if not OPENAI_VERSION_COMPATIBLE:
    raise ImportError(
        "An incompatible OpenAI library version is installed. Ensure you have version 1.16.1 or later.",
    )

# Attempt to handle specific OpenAI errors; fallback to general ones if necessary.
try:
    import openai.error
    ERRORS = (openai.error.RateLimitError, openai.error.ServiceUnavailableError, openai.error.AP
except Exception:
    ERRORS = (openai.RateLimitError, openai.APIError)


class MyScaleRM(dspy.Retrieve):
    """
    A retrieval module that uses MyScaleDB to return the top passages for a given query.

    MyScaleDB is a fork of ClickHouse that focuses on vector similarity search and full
    text search.  MyScaleRM is designed to facilitate easy retrieval of information from
    MyScaleDB using embeddings.  It supports embedding generation through either a local
    model or the OpenAI API. This class abstracts away the complexities of connecting to
    MyScaleDB, managing API keys, and processing queries to return semantically
```

---
sidebar_position: 8
---

retrieve.RAGatouilleRM

Constructor

The constructor initializes the `RAGatouille` class instance and sets up the required parameters for

```python
class RAGatouilleRM(dspy.Retrieve):
    def __init__(
        self,
        index_root: str,
        index_name: str,
        k: int = 3,
    ):
```

Parameters:
- `index_root` (_str_): Folder path where your index is stored.
- `index_name` (_str_): Name of the index you want to retrieve from.
- `k` (_int_): The default number of passages to retrieve. Defaults to `3`.

Methods

`forward(self, query_or_queries: Union[str, List[str]], k:Optional[int]) -> dspy.Prediction`

Enables making queries to the RAGatouille-made index for retrieval. Internally, the method handles

Parameters:
- `query_or_queries` (Union[str, List[str]]): Query string used for retrieval.
- `k` (_int_, _optional_): Number of passages to retrieve. Defaults to 3.

Returns:
- `dspy.Prediction`: List of k passages

---
sidebar_position: 1
---

dspy.ColBERTv2

Constructor

The constructor initializes the `ColBERTv2` class instance and sets up the request parameters for i

```python
class ColBERTv2:
    def __init__(
        self,
        url: str = "http://0.0.0.0",
        port: Optional[Union[str, int]] = None,
        post_requests: bool = False,
    ):
```

Parameters:
- `url` (_str_): URL for ColBERTv2 server.
- `port` (_Union[str, int]_, _Optional_): Port endpoint for ColBERTv2 server. Defaults to `None`.
- `post_requests` (_bool_, _Optional_): Flag for using HTTP POST requests. Defaults to `False`.

Methods

`__call__(self, query: str, k: int = 10, simplify: bool = False) -> Union[list[str], list[dotdict]]`

Enables making queries to the ColBERTv2 server for retrieval. Internally, the method handles the s

Parameters:
- `query` (_str_): Query string used for retrieval.
- `k` (_int_, _optional_): Number of passages to retrieve. Defaults to 10.
- `simplify` (_bool_, _optional_): Flag for simplifying output to a list of strings. Defaults to False.

Returns:
- `Union[list[str], list[dotdict]]`: Depending on `simplify` flag, either a list of strings representing the p

Quickstart

```python
import dspy

colbertv2_wiki17_abstracts = dspy.ColBERTv2(url='http://20.102.90.50:2017/wiki17_abstracts')

retrieval_response = colbertv2_wiki17_abstracts('When was the first FIFA World Cup held?', k=5)
```

---
sidebar_position: 11
---

retrieve.YouRM

Constructor

Initialize an instance of the `YouRM` class that calls the [You.com APIs](https://documentation.you.

```python
YouRM(
    ydc_api_key: Optional[str] = None,
    k: int = 3,
    endpoint: Literal["search", "news"] = "search",
    num_web_results: Optional[int] = None,
    safesearch: Optional[Literal["off", "moderate", "strict"]] = None,
    country: Optional[str] = None,
    search_lang: Optional[str] = None,
    ui_lang: Optional[str] = None,
    spellcheck: Optional[bool] = None,
)
```

Parameters:

- `ydc_api_key` (Optional[str]): you.com API key, if `YDC_API_KEY` is not set in the environment
- `k` (int): If ``endpoint="search"``, the max snippets to return per search hit.
      If ``endpoint="news"``, the max articles to return.
- `endpoint` (Literal["search", "news"]): you.com endpoints
- `num_web_results` (Optional[int]): The max number of web results to return, must be under 20
- `safesearch` (Optional[Literal["off", "moderate", "strict"]]): Safesearch settings, one of "off", "mode
- `country` (Optional[str]): Country code, ex: 'US' for United States, see API reference for more info
- `search_lang` (Optional[str]): (News API) Language codes, ex: 'en' for English, see API reference
- `ui_lang` (Optional[str]): (News API) User interface language for the response, ex: 'en' for English
                    See API reference for more info
- `spellcheck` (Optional[bool]): (News API) Whether to spell check query or not, defaults to True

Methods

`forward(self, query_or_queries: Union[str, List[str]], k: Optional[int] = None) -> dspy.Predictio

If `endpoint="search"`, search the web for the top `k` snippets matching the given query or queries.

If `endpoint="news"`, search the web for the top `k` articles matching the given query or queries.

---
sidebar_position: 5
---

retrieve.MilvusRM

Constructor

Initialize an instance of the `MilvusRM` class, with the option to use OpenAI's `text-embedding-3-sm

```python
MilvusRM(
    collection_name: str,
    uri: Optional[str] = "http://localhost:19530",
    token: Optional[str] = None,
    db_name: Optional[str] = "default",
    embedding_function: Optional[Callable] = None,
    k: int = 3,
)
```

Parameters:
- `collection_name (str)`: The name of the Milvus collection to query against.
- `uri (str, optional)`: The Milvus connection uri. Defaults to "http://localhost:19530".
- `token (str, optional)`: The Milvus connection token. Defaults to None.
- `db_name (str, optional)`: The Milvus database name. Defaults to "default".
- `embedding_function (callable, optional)`: The function to convert a list of text to embeddings.
    The embedding function should take a list of text strings as input and output a list of embeddings
    Defaults to None. By default, it will get OpenAI client by the environment variable OPENAI_API_
- `k (int, optional)`: The number of top passages to retrieve. Defaults to 3.

Methods

`forward(self, query_or_queries: Union[str, List[str]], k: Optional[int] = None) -> dspy.Predictio

Search the Milvus collection for the top `k` passages matching the given query or queries, using em

Parameters:
- `query_or_queries` (_Union[str, List[str]]_): The query or list of queries to search for.
- `k` (_Optional[int]_, _optional_): The number of results to retrieve. If not specified, defaults to the v

Returns:
- `dspy.Prediction`: Contains the retrieved passages, each represented as a `dotdict` with schema `

Quickstart

To support passage retrieval, it assumes that a Milvus collection has been created and populated w

---
sidebar_position: 1
---

dspy.TypedPredictor

The `TypedPredictor` class is a sophisticated module designed for making predictions with strict typ

Constructor

```python
TypedPredictor(
    CodeSignature
    max_retries=3
)
```

Parameters:
* `signature` (dspy.Signature): The signature that defines the input and output fields along with thei
* `max_retries` (int, optional): The maximum number of retries for generating a valid prediction outp

Methods

`copy() -> "TypedPredictor"`

Creates and returns a deep copy of the current TypedPredictor instance.

Returns: A new instance of TypedPredictor that is a deep copy of the original instance.

`_make_example(type_: Type) -> str`

A static method that generates a JSON object example based pn the schema of the specified Pyda

Parameters:
* `type_`: A Pydantic model class for which an example JSON object is to be generated.

Returns: A string that represents a JSON object example, which validates against the provided

`_prepare_signature() -> dspy.Signature`

Prepares and returns a modified version of the signature associated with the TypedPredictor instan

Returns: A dspy.Signature object that has been enhanced with formatting and parsing specifica

`forward(**kwargs) -> dspy.Prediction`

Executes the prediction logic, making use of the `dspy.Predict` component to generate predictions

---
sidebar_position: 3
---

dspy.predictor

Overview

`def predictor(func) -> dspy.Module`

The `@predictor` decorator is used to create a predictor module based on the provided function. It

* **Input**: Function with input parameters and return type annotation.
* **Output**: A dspy.Module instance capable of making predictions.

Example

```python
import dspy

context = ["Roses are red.", "Violets are blue"]
question = "What color are roses?"

@dspy.predictor
def generate_answer(self, context: list[str], question) -> str:
    """Answer questions with short factoid answers."""
    pass

generate_answer(context=context, question=question)
```

```
{
  "label": "Local Language Model Clients",
  "position": 6,
  "link": {
    "type": "generated-index",
    "description": "DSPy supports various methods including `built-in wrappers`, `server integration
  }
}
```

---
sidebar_position: 2
---

dspy.TypedChainOfThought

Overview

`def TypedChainOfThought(signature, max_retries=3) -> dspy.Module`

Adds a Chain of Thoughts `dspy.OutputField` to the `dspy.TypedPredictor` module by prepending i

* **Inputs**:
    * `signature`: The `dspy.Signature` specifying the input/output fields
    * `max_retries`: Maximum number of retries if outputs fail validation
* **Output**: A dspy.Module instance capable of making predictions.

Example

```python
from dspy import InputField, OutputField, Signature
from dspy.functional import TypedChainOfThought
from pydantic import BaseModel

# We define a pydantic type that automatically checks if it's argument is valid python code.
class CodeOutput(BaseModel):
    code: str
    api_reference: str

class CodeSignature(Signature):
    function_description: str = InputField()
    solution: CodeOutput = OutputField()

cot_predictor = TypedChainOfThought(CodeSignature)
prediction = cot_predictor(
    function_description="Write a function that adds two numbers."
)

print(prediction["code"])
print(prediction["api_reference"])
```

---
sidebar_position: 4
---

dspy.cot

Overview

`def cot(func) -> dspy.Module`

The `@cot` decorator is used to create a Chain of Thoughts module based on the provided function

* **Input**: Function with input parameters and return type annotation.
* **Output**: A dspy.Module instance capable of making predictions.

Example

```python
import dspy

context = ["Roses are red.", "Violets are blue"]
question = "What color are roses?"

@dspy.cot
def generate_answer(self, context: list[str], question) -> str:
    """Answer questions with short factoid answers."""
    pass

generate_answer(context=context, question=question)
```

dspy.HFClientVLLM

Setting up the vLLM Server

Follow these steps to set up the vLLM Server:

1. Build the server from source by following the instructions provided in the [Build from Source guid

2. Start the server by running the following command, and specify your desired model, host, and po

Example command:

```bash
  python -m vllm.entrypoints.openai.api_server --model mosaicml/mpt-7b --port 8000
```

This will launch the vLLM server.

Sending requests to the server

After setting up the vLLM server and ensuring that it displays "Connected" when it's running, you ca

Initialize the `HFClientVLLM` within your program with the desired parameters. Here is an example

```python
  lm = dspy.HFClientVLLM(model="mosaicml/mpt-7b", port=8000, url="http://localhost")
```
Customize the `model`, `port`, `url`, and `max_tokens` according to your requirements. The `model

Please refer to the [official vLLM repository](https://github.com/vllm-project/vllm) for more detailed i

Sending requests to vLLM server using [dspy.OpenAI](https://dspy-docs.vercel.app/api/langua

Query the vLLM server using OpenAI SDK through [`dspy.OpenAI`](https://dspy-docs.vercel.app/ap

```python
  lm = dspy.OpenAI(model="mosaicml/mpt-7b", api_base="http://localhost:8000/v1/", api_key="EM
```

Similarly, customize the `model`, `port`, `url` (vLLM arguments), along with the remaining OpenAI c

dspy.ChatModuleClient

Prerequisites

1. Install the required packages using the following commands:

   ```shell
   pip install --no-deps --pre --force-reinstall mlc-ai-nightly-cu118 mlc-chat-nightly-cu118 -f https://ml
   pip install transformers
   git lfs install
   ```

   Adjust the pip wheels according to your OS/platform by referring to the provided commands in [M

Running MLC Llama-2 models

1. Create a directory for prebuilt models:

   ```shell
   mkdir -p dist/prebuilt
   ```

2. Clone the necessary libraries from the repository:

   ```shell
   git clone https://github.com/mlc-ai/binary-mlc-llm-libs.git dist/prebuilt/lib
   cd dist/prebuilt
   ```

3. Choose a Llama-2 model from [MLC LLMs](https://huggingface.co/mlc-ai) and clone the model r

   ```shell
   git clone https://huggingface.co/mlc-ai/mlc-chat-Llama-2-7b-chat-hf-q4f16_1
   ```

4. Initialize the `ChatModuleClient` within your program with the desired parameters. Here's an exa

   ```python
   llama = dspy.ChatModuleClient(model='dist/prebuilt/mlc-chat-Llama-2-7b-chat-hf-q4f16_1', mode
   ```

Please refer to the [official MLC repository](https://github.com/mlc-ai/mlc-llm) for more detailed info

dspy.TensorRTModel

TensorRT LLM by Nvidia happens to be one of the most optimized inference engines to run open-s

Prerequisites

Install TensorRT LLM by the following instructions [here](https://nvidia.github.io/TensorRT-LLM/inst

In order to use this module, you should have the model weights file in engine format. To understan

Running TensorRT model inside dspy

```python
from dspy import TensorRTModel

engine_dir = "<your-path-to-engine-dir>"
model_name_or_path = "<hf-model-id-or-path-to-tokenizer>"

model = TensorRTModel(engine_dir=engine_dir, model_name_or_path=model_name_or_path)
```

You can perform more customization on model loading based on the following example. Below is a

- **use_py_session** (`bool`, optional): Whether to use a Python session or not. Defaults to `False`
- **lora_dir** (`str`): The directory of LoRA adapter weights.
- **lora_task_uids** (`List[str]`): List of LoRA task UIDs; use `-1` to disable the LoRA module.
- **lora_ckpt_source** (`str`): The source of the LoRA checkpoint.

If `use_py_session` is set to `False`, the following kwargs are supported (This runs in C++ runtime)

- **max_batch_size** (`int`, optional): The maximum batch size. Defaults to `1`.
- **max_input_len** (`int`, optional): The maximum input context length. Defaults to `1024`.
- **max_output_len** (`int`, optional): The maximum output context length. Defaults to `1024`.
- **max_beam_width** (`int`, optional): The maximum beam width, similar to `n` in OpenAI API. De
- **max_attention_window_size** (`int`, optional): The attention window size that controls the slidin
- **sink_token_length** (`int`, optional): The sink token length. Defaults to `1`.

> Please note that you need to complete the build processes properly before applying these custom

Now to run the model, we need to add the following code:

```python
response = model("hello")
```

This gives this result:

```
{
  "label": "Local Language Model Clients",
  "position": 6,
  "link": {
    "type": "generated-index",
    "description": "DSPy supports various methods including `built-in wrappers`, `server integration
  }
}
```

dspy.OllamaLocal

:::note
Adapted from documentation provided by https://github.com/insop
:::

Ollama is a good software tool that allows you to run LLMs locally, such as Mistral, Llama2, and Ph
The following are the instructions to install and run Ollama.

Prerequisites

Install Ollama by following the instructions from this page:

- https://ollama.ai

Download model: `ollama pull`

Download a model by running the `ollama pull` command. You can download Mistral, Llama2, and

```bash
# download mistral
ollama pull mistral
```

Here is the list of other models you can download:
- https://ollama.ai/library

Running Ollama model

Run model: `ollama run`

You need to start the model server with the `ollama run` command.

```bash
# run mistral
ollama run mistral
```

Sending requests to the server

Here is the code to load a model through Ollama:

```python
lm = dspy.OllamaLocal(model='mistral')
```

dspy.HFClientTGI

Prerequisites

- Docker must be installed on your system. If you don't have Docker installed, you can get it from [h

Setting up the Text-Generation-Inference Server

1. Clone the Text-Generation-Inference repository from GitHub by executing the following comman

   ```
   git clone https://github.com/huggingface/text-generation-inference.git
   ```

2. Change into the cloned repository directory:

   ```
   cd text-generation-inference
   ```

3. Execute the Docker command under the "Get Started" section to run the server:

   ```
   model=meta-llama/Llama-2-7b-hf # set to the specific Hugging Face model ID you wish to use.
   num_shard=2 # set to the number of shards you wish to use.
   volume=$PWD/data # share a volume with the Docker container to avoid downloading weights ev

   docker run --gpus all --shm-size 1g -p 8080:80 -v $volume:/data ghcr.io/huggingface/text-generat
   ```

   This command will start the server and make it accessible at `http://localhost:8080`.

If you want to connect to [Meta Llama 2 models](https://huggingface.co/meta-llama), make sure to

```
   docker run --gpus all --shm-size 1g -p 8080:80 -v $volume:/data -e HUGGING_FACE_HUB_TOK
```

Sending requests to the server

After setting up the text-generation-inference server and ensuring that it displays "Connected" when

Initialize the `HFClientTGI` within your program with the desired parameters. Here is an example ca

   ```python
   lm = dspy.HFClientTGI(model="meta-llama/Llama-2-7b-hf", port=8080, url="http://localhost")
   ```

dspy.HFModel

Initialize `HFModel` within your program with the desired model to load in. Here's an example call:

```python
llama = dspy.HFModel(model = 'meta-llama/Llama-2-7b-hf')
```

---
title: Markdown page example

Markdown page example

You don't need React to write simple standalone pages.

```
/**
 * CSS files with the .module.css suffix will be treated as CSS modules
 * and scoped locally.
 */

.heroBanner {
  padding: 4rem 0;
  text-align: center;
  position: relative;
  overflow: hidden;
}

@media screen and (max-width: 996px) {
  .heroBanner {
    padding: 2rem;
  }
}

.buttons {
  display: flex;
  align-items: center;
  justify-content: center;
}

.buttonHoverEffect {
  background-color: var(--bg-color);
  color: var(--hero-text-color) !important;
  border-color: var(--hero-text-color);
  transition: background-color 0.3s ease, color 0.3s ease;
}

.buttonHoverEffect[data-theme='dark'] {
  background-color: var(--hero-text-color);
  color: var(--bg-color) !important;
  border-color: #0079f1;
}

.buttonHoverEffect:hover {
  background-color: var(--hero-text-color);
  color: var(--bg-color) !important;
  border-color: var(--bg-color);
}
```

```
/**
 * Any CSS included here will be global. The classic template
 * bundles Infima by default. Infima is a CSS framework designed to
 * work well for content-centric websites.
 */

/* You can override the default Infima variables here. */
:root {
  --ifm-color-primary: #2e8555;
  --ifm-color-primary-dark: #29784c;
  --ifm-color-primary-darker: #277148;
  --ifm-color-primary-darkest: #205d3b;
  --ifm-color-primary-light: #33925d;
  --ifm-color-primary-lighter: #359962;
  --ifm-color-primary-lightest: #3cad6e;
  --ifm-code-font-size: 95%;
  --docusaurus-highlighted-code-line-bg: rgba(0, 0, 0, 0.1);
  --bg-color: #f5f6f7;
  --hero-text-color: #222831;
  --author-box-bg: #f9f9f9;
  --author-box-border: #ddd;
  --author-links-svg-color: #1c1e21;
}

/* For readability concerns, you should choose a lighter palette in dark mode. */
[data-theme='dark'] {
  --ifm-color-primary: #25c2a0;
  --ifm-color-primary-dark: #21af90;
  --ifm-color-primary-darker: #1fa588;
  --ifm-color-primary-darkest: #1a8870;
  --ifm-color-primary-light: #29d5b0;
  --ifm-color-primary-lighter: #32d8b4;
  --ifm-color-primary-lightest: #4fddbf;
  --docusaurus-highlighted-code-line-bg: rgba(0, 0, 0, 0.3);
  --bg-color: #1c1e21;
  --hero-text-color: #f5f6f7;
  --author-box-bg: #2e2e2e;
  --author-box-border: #333;
  --author-links-svg-color: #f5f6f7;
}
```

```
.features {
  display: flex;
  align-items: center;
  padding: 2rem 0;
  width: 100%;
}

.featureSvg {
  height: 200px;
  width: 200px;
}
```

```
.features {
  display: flex;
  align-items: center;
  padding: 2rem 0;
  width: 100%;
}

.featureSvg {
  height: 200px;
  width: 200px;
}
```

```
#!/bin/bash

# The $1 argument is the version number passed from the workflow
VERSION=$1

echo "version: $VERSION"

for i in {1..5}; do
  if python3 -m pip install --index-url https://test.pypi.org/simple/ --extra-index-url https://pypi.org/sim
    break
  else
    echo "Attempt $i failed. Waiting before retrying..."
    sleep 10
  fi
done
```

· Changes Description

This MR/PR contains the following changes:
...

· Contributor Checklist

- [] Pre-Commit checks are passing (locally and remotely)
- [] Title of your PR / MR corresponds to the required format
- [] Commit message follows required format {label}(dspy): {message}

·· Warnings

Anything we should be aware of ?

```
import csv
import datetime
import os
from timeit import default_timer as timer

import openai
from dotenv import load_dotenv

import dspy
from dspy.evaluate import Evaluate

from .tasks.biodex import BioDexTask
from .tasks.gsm8k import GSM8KTask
from .tasks.hotpotqa import HotPotQATask
from .tasks.scone import ScoNeTask
from .tasks.tweet import TweetTask
from .tasks.tweet_metric import TweetMetricTask

datasets = ["ScoNe", "HotPotQA", "GSM8K", "BioDex", "Tweet"]

class OptimizerTester:
    def __init__(self, datasets=datasets, default_train_num = 200, default_dev_num = 100, default_t
                 num_threads = 10, default_breadth = 10, default_depth = 3, default_temperature = 1.4,
                 prompt_model_name = "gpt-3.5-turbo-1106", task_model_name = "meta-llama/Llama-2-
                 prompt_model = None, task_model = None,
                 colbert_v2_endpoint = "http://20.102.90.50:2017/wiki17_abstracts"):
        self.datasets = datasets
        self.TRAIN_NUM = default_train_num
        self.DEV_NUM = default_dev_num
        self.EVAL_NUM = default_test_num
        self.NUM_THREADS = num_threads
        self.BREADTH = default_breadth
        self.DEPTH = default_depth
        self.TEMPERATURE = default_temperature
        self.PROMPT_MODEL_NAME = prompt_model_name
        self.TASK_MODEL_NAME = task_model_name
        self.COLBERT_V2_ENDPOINT = colbert_v2_endpoint

        load_dotenv()  # This will load the .env file's variables

        openai.api_key = os.environ.get('OPENAI_API_KEY')
        openai.api_base = os.environ.get('OPENAI_API_BASE')

        # Prompt gen model
        if not prompt_model:
            self.prompt_model = dspy.OpenAI(model=self.PROMPT_MODEL_NAME, max_tokens=150
        else:
```

Getting Started with Create React App

This project was bootstrapped with [Create React App](https://github.com/facebook/create-react-ap

Available Scripts

In the project directory, you can run:

`npm start`

Runs the app in the development mode.\
Open http://localhost:3000 to view it in your browser.

The page will reload when you make changes.\
You may also see any lint errors in the console.

`npm test`

Launches the test runner in the interactive watch mode.\
See the section about [running tests](https://facebook.github.io/create-react-app/docs/running-tests

`npm run build`

Builds the app for production to the `build` folder.\
It correctly bundles React in production mode and optimizes the build for the best performance.

The build is minified and the filenames include the hashes.\
Your app is ready to be deployed!

See the section about [deployment](https://facebook.github.io/create-react-app/docs/deployment) fo

`npm run eject`

Note: this is a one-way operation. Once you `eject`, you can't go back!

If you aren't satisfied with the build tool and configuration choices, you can `eject` at any time. This

Instead, it will copy all the configuration files and the transitive dependencies (webpack, Babel, ESL

You don't have to ever use `eject`. The curated feature set is suitable for small and middle deploym

Learn More

You can learn more in the [Create React App documentation](https://facebook.github.io/create-reac

To learn React, check out the [React documentation](https://reactjs.org/).

```
from abc import ABC, abstractmethod


class BaseTask(ABC):
    def __init__(self):
        pass

    @abstractmethod
    def get_program(self):
        pass

    @abstractmethod
    def get_metric(self):
        pass

    def get_trainset(self, TRAIN_NUM=None):
        if TRAIN_NUM:
            self.TRAIN_NUM = TRAIN_NUM
            return self.trainset[:TRAIN_NUM]
        else:
            return self.trainset[:self.TRAIN_NUM]

    def get_devset(self, TRAIN_NUM=None, DEV_NUM=None):
        if TRAIN_NUM:
            self.TRAIN_NUM = TRAIN_NUM
        if DEV_NUM:
            self.DEV_NUM = DEV_NUM

        index = -1
        if DEV_NUM and TRAIN_NUM:
            index = max(len(self.trainset) - DEV_NUM, TRAIN_NUM)
        else:
            index = max(len(self.trainset) - self.DEV_NUM, self.TRAIN_NUM)
        return self.trainset[index:]

    def get_evalset(self, EVAL_NUM=None):
        if EVAL_NUM:
            self.EVAL_NUM = EVAL_NUM
            return self.devset[:EVAL_NUM]
        else:
            return self.devset[:self.EVAL_NUM]

    def set_splits(self, TRAIN_NUM=None, DEV_NUM=None, EVAL_NUM=None):
        if TRAIN_NUM:
            self.TRAIN_NUM = TRAIN_NUM
        if DEV_NUM:
            self.DEV_NUM = DEV_NUM
```

```
import os
from functools import lru_cache

import openai
from dotenv import load_dotenv

import dspy
from dspy.datasets import HotPotQA

from .base_task import BaseTask


class TweetSignature(dspy.Signature):
    ("""Given context and a question, answer with a tweet""")

    context = dspy.InputField()
    question = dspy.InputField()
    answer = dspy.OutputField(desc="Yes or No")

class TweetCoT(dspy.Module):
    def __init__(self):
        super().__init__()
        self.generate_answer = dspy.ChainOfThought(TweetSignature)

    def forward(self, context, question):
        return self.generate_answer(context=context, question=question)

class MultiHopTweet(dspy.Module):
    def __init__(self,passages_per_hop):
        super().__init__()
        self.retrieve = dspy.Retrieve(k = passages_per_hop)
        self.generate_query = dspy.ChainOfThought("context ,question->search_query")
        self.generate_answer = TweetCoT()

    def forward (self,question) :
        context = []
        for hop in range(2):
            query = self.generate_query(context = context, question = question).search_query
            context += self.retrieve(query).passages
        return dspy.Prediction(context=context, answer=self.generate_answer(context = context , que

# Define the signature for automatic assessments.
class Assess(dspy.Signature):
    """Assess the quality of a tweet along the specified dimension."""

    context = dspy.InputField(desc='ignore if N/A')
    assessed_text = dspy.InputField()
```

```
from .aggregation import majority
from .chain_of_thought import ChainOfThought
from .chain_of_thought_with_hint import ChainOfThoughtWithHint
from .knn import KNN
from .multi_chain_comparison import MultiChainComparison
from .predict import Predict
from .program_of_thought import ProgramOfThought
from .react import ReAct
from .retry import Retry
```

```
from __future__ import annotations

import math
import os
import pickle
import re
from collections import defaultdict
from enum import Enum
from functools import lru_cache
from typing import DefaultDict, Dict, List, Optional, Union

import datasets
import tqdm
from pydantic import BaseModel
from rapidfuzz import process

import dspy
from dsp.utils import deduplicate
from dspy.evaluate import Evaluate

from .base_task import BaseTask

# Must point to a MedDra download with mdhier.asc
data_dir = '/future/u/okhattab/data/2023/MedDraV2/meddra_23_0_english/MedAscii' # NOTE: EDIT

global meddra_root
meddra_root = None

def _remove_non_llt(candidates):
    return [c for c in candidates if c.level == Level.LLT]


def get_path(node):
    path = f"under {node.parent.term} < {node.parent.parent.term} < {node.parent.parent.parent.term
    return path


def flatten(ll):
    return [x for l in ll for x in l]


def _matches_to_terms_and_paths(matches):
    s = []
    for n in matches:
        new_s = f"{n.node.term} ({n.score} score for query '{n.query}'; {n.node.count} counts; {get_pat
        s.append(new_s)
    return s
```

```
import random

from datasets import load_dataset

from dspy.datasets.dataset import Dataset


class HotPotQA(Dataset):
    def __init__(self, *args, only_hard_examples=True, keep_details='dev_titles', unofficial_dev=Tru
        super().__init__(*args, **kwargs)
        assert only_hard_examples, "Care must be taken when adding support for easy examples." \
                                   "Dev must be all hard to match official dev, but training can be flexible."

        hf_official_train = load_dataset("hotpot_qa", 'fullwiki', split='train', trust_remote_code=True)
        hf_official_dev = load_dataset("hotpot_qa", 'fullwiki', split='validation', trust_remote_code=True

        official_train = []
        for raw_example in hf_official_train:
            if raw_example['level'] == 'hard':
                if keep_details is True:
                    keys = ['id', 'question', 'answer', 'type', 'supporting_facts', 'context']
                elif keep_details == 'dev_titles':
                    keys = ['question', 'answer', 'supporting_facts']
                else:
                    keys = ['question', 'answer']

                example = {k: raw_example[k] for k in keys}

                if 'supporting_facts' in example:
                    example['gold_titles'] = set(example['supporting_facts']['title'])
                    del example['supporting_facts']

                official_train.append(example)

        rng = random.Random(0)
        rng.shuffle(official_train)

        self._train = official_train[:len(official_train)*75//100]

        if unofficial_dev:
            self._dev = official_train[len(official_train)*75//100:]
        else:
            self._dev = None

        for example in self._train:
            if keep_details == 'dev_titles':
                del example['gold_titles']
```

```
import dspy
from dspy.datasets import HotPotQA

# Load the dataset.
dataset = HotPotQA(train_seed=1, train_size=200, eval_seed=2023, dev_size=200, test_size=0, k

# Tell DSPy that the 'question' field is the input. Any other fields are labels and/or metadata.
trainset = [x.without('id', 'type').with_inputs('question') for x in dataset.train]
devset = [x.without('id', 'type').with_inputs('question') for x in dataset.dev]
valset, devset = devset[:50], devset[50:]


# Define the signature for automatic assessments.
class Assess(dspy.Signature):
    """Assess the quality of a tweet along the specified dimension."""

    context = dspy.InputField(desc='ignore if N/A')
    assessed_text = dspy.InputField()
    assessment_question = dspy.InputField()
    assessment_answer = dspy.OutputField(desc="Yes or No")


gpt4T = dspy.OpenAI(model='gpt-4-1106-preview', max_tokens=1000, model_type='chat')
retrieve = dspy.Retrieve(k=5)
METRIC = None

def metric(gold, pred, trace=None):
    question, answer, tweet = gold.question, gold.answer, pred.output
    context = retrieve(question).passages

    engaging = "Does the assessed text make for a self-contained, engaging tweet?"
    faithful = "Is the assessed text grounded in the context? Say no if it includes significant facts not
    correct = f"The text above is should answer `{question}`. The gold answer is `{answer}`."
    correct = f"{correct} Does the assessed text above contain the gold answer?"

    with dspy.context(lm=gpt4T):
        faithful = dspy.Predict(Assess)(context=context, assessed_text=tweet, assessment_question=
        correct =  dspy.Predict(Assess)(context='N/A', assessed_text=tweet, assessment_question=c
        engaging = dspy.Predict(Assess)(context='N/A', assessed_text=tweet, assessment_question=

    correct, engaging, faithful = (m.assessment_answer.split()[0].lower() == 'yes' for m in [correct, en
    score = (correct + engaging + faithful) if correct and (len(tweet) <= 280) else 0

    if METRIC is not None:
        if METRIC == 'correct': return correct
        if METRIC == 'engaging': return engaging
        if METRIC == 'faithful': return faithful
```

```
import random

import tqdm
from datasets import load_dataset


class GSM8K:
    def __init__(self) -> None:
        super().__init__()
        self.do_shuffle = False

        dataset = load_dataset("gsm8k", 'main')

        hf_official_train = dataset['train']
        hf_official_test = dataset['test']
        official_train = []
        official_test = []

        for example in tqdm.tqdm(hf_official_train):
            question = example['question']

            answer = example['answer'].strip().split()
            assert answer[-2] == '####'

            gold_reasoning = ' '.join(answer[:-2])
            answer = str(int(answer[-1].replace(',', '')))

            official_train.append(dict(question=question, gold_reasoning=gold_reasoning, answer=answ

        for example in tqdm.tqdm(hf_official_test):
            question = example['question']

            answer = example['answer'].strip().split()
            assert answer[-2] == '####'

            gold_reasoning = ' '.join(answer[:-2])
            answer = str(int(answer[-1].replace(',', '')))

            official_test.append(dict(question=question, gold_reasoning=gold_reasoning, answer=answ

        rng = random.Random(0)
        rng.shuffle(official_train)

        rng = random.Random(0)
        rng.shuffle(official_test)

        trainset = official_train[:200]
```

```
import glob
import os
import random

import pandas as pd

import dspy

from .base_task import BaseTask


def load_scone(dirname):
    dfs = []
    for filename in glob.glob(dirname + "/*.csv"):
        df = pd.read_csv(filename, index_col=0)
        df['category'] = os.path.basename(filename).replace(".csv", "")
        dfs.append(df)
    data_df = pd.concat(dfs)

    def as_example(row):
        # The 'one_scoped' file is from an earlier dataset, MoNLI, and
        # so is formatted a bit differently:
        suffix = '' if row['category'] == 'one_scoped' else '_edited'
        # Reformat the hypothesis to be an embedded clause in a question:
        hkey = 'sentence2' + suffix
        question = row[hkey][0].lower() + row[hkey][1: ].strip(".")
        question = f"Can we logically conclude for sure that {question}?"
        # Binary task formulation:
        label = "Yes" if row['gold_label' + suffix] == 'entailment' else "No"
        return dspy.Example({
            "context": row['sentence1' + suffix],
            "question": question,
            "answer": label,
            "category": row['category'],
        }).with_inputs("context", "question")

    return list(data_df.apply(as_example, axis=1).values)

class ScoNeSignature(dspy.Signature):
    """context, question -> answer"""

    context = dspy.InputField()
    question = dspy.InputField()
    answer = dspy.OutputField(desc="Yes or No")

class ScoNeCoT(dspy.Module):
    def __init__(self):
```

```
import time

import boto3
from flask import Flask, jsonify, request
from flask_cors import CORS

app = Flask(__name__)
CORS(app)

dynamo_resource = boto3.resource("dynamodb", region_name="us-west-1")
table = dynamo_resource.Table('dsp-inspect-app')


@app.route("/")
def index():
  return "This is the main page."


@app.route("/data/<id>", methods=["GET"])
def get_item(id):
  response = table.get_item(Key={"id": id})

  if 'Item' in response:
    return jsonify(response['Item'])
  else:
    return 'Item not found', 404


@app.route("/inspect-db", methods=["GET"])
def inspect_db():
  return table.scan()


@app.route('/log-item', methods=['POST'])
def log_item():
    data = request.get_json()

    if 'id' not in data or 'content' not in data:
        return 'Missing required fields', 400

    data['expiry_time'] = int(time.time() + 86400)
    table.put_item(Item=data)

    return 'Data created successfully', 201


if __name__ == "__main__":
```

Getting Started with Create React App

This project was bootstrapped with [Create React App](https://github.com/facebook/create-react-ap

Available Scripts

In the project directory, you can run:

`npm start`

Runs the app in the development mode.\
Open http://localhost:3000 to view it in your browser.

The page will reload when you make changes.\
You may also see any lint errors in the console.

`npm test`

Launches the test runner in the interactive watch mode.\
See the section about [running tests](https://facebook.github.io/create-react-app/docs/running-tests

`npm run build`

Builds the app for production to the `build` folder.\
It correctly bundles React in production mode and optimizes the build for the best performance.

The build is minified and the filenames include the hashes.\
Your app is ready to be deployed!

See the section about [deployment](https://facebook.github.io/create-react-app/docs/deployment) fo

`npm run eject`

Note: this is a one-way operation. Once you `eject`, you can't go back!

If you aren't satisfied with the build tool and configuration choices, you can `eject` at any time. This

Instead, it will copy all the configuration files and the transitive dependencies (webpack, Babel, ESL

You don't have to ever use `eject`. The curated feature set is suitable for small and middle deploym

Learn More

You can learn more in the [Create React App documentation](https://facebook.github.io/create-reac

To learn React, check out the [React documentation](https://reactjs.org/).

```
{
  "name": "dsp-app",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@emotion/react": "^11.10.5",
    "@emotion/styled": "^11.10.5",
    "@mui/icons-material": "^5.11.9",
    "@mui/material": "^5.11.9",
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "axios": "^1.3.3",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

```
{
  "name": "dsp-app",
  "version": "0.1.0",
  "lockfileVersion": 2,
  "requires": true,
  "packages": {
    "": {
      "name": "dsp-app",
      "version": "0.1.0",
      "dependencies": {
        "@emotion/react": "^11.10.5",
        "@emotion/styled": "^11.10.5",
        "@mui/icons-material": "^5.11.9",
        "@mui/material": "^5.11.9",
        "@testing-library/jest-dom": "^5.16.5",
        "@testing-library/react": "^13.4.0",
        "@testing-library/user-event": "^13.5.0",
        "axios": "^1.3.3",
        "react": "^18.2.0",
        "react-dom": "^18.2.0",
        "react-scripts": "5.0.1",
        "web-vitals": "^2.1.4"
      }
    },
    "node_modules/@adobe/css-tools": {
      "version": "4.1.0",
      "resolved": "https://registry.npmjs.org/@adobe/css-tools/-/css-tools-4.1.0.tgz",
      "integrity": "sha512-mMVJ/j/GbZ/De4ZHWbQAQO1J6iVnjtZLc9WEdkUQb8S/Bu2cAF2bETXUg
    },
    "node_modules/@ampproject/remapping": {
      "version": "2.2.0",
      "resolved": "https://registry.npmjs.org/@ampproject/remapping/-/remapping-2.2.0.tgz",
      "integrity": "sha512-qRmjj8nj9qmLTQXXmaR1cck3UXSRMPrbsLJAasZpF+t3riI71BXed5ebIOY
      "dependencies": {
        "@jridgewell/gen-mapping": "^0.1.0",
        "@jridgewell/trace-mapping": "^0.3.9"
      },
      "engines": {
        "node": ">=6.0.0"
      }
    },
    "node_modules/@babel/code-frame": {
      "version": "7.18.6",
      "resolved": "https://registry.npmjs.org/@babel/code-frame/-/code-frame-7.18.6.tgz",
      "integrity": "sha512-TDCmlK5eOvH+eH7cdAFlNXeVJqWIQ7gW9tY1GJIpUtFb6CmjVyq2VM3u
      "dependencies": {
        "@babel/highlight": "^7.18.6"
```

Getting Started with Create React App

This project was bootstrapped with [Create React App](https://github.com/facebook/create-react-ap

Available Scripts

In the project directory, you can run:

`npm start`

Runs the app in the development mode.\
Open http://localhost:3000 to view it in your browser.

The page will reload when you make changes.\
You may also see any lint errors in the console.

`npm test`

Launches the test runner in the interactive watch mode.\
See the section about [running tests](https://facebook.github.io/create-react-app/docs/running-tests

`npm run build`

Builds the app for production to the `build` folder.\
It correctly bundles React in production mode and optimizes the build for the best performance.

The build is minified and the filenames include the hashes.\
Your app is ready to be deployed!

See the section about [deployment](https://facebook.github.io/create-react-app/docs/deployment) fo

`npm run eject`

Note: this is a one-way operation. Once you `eject`, you can't go back!

If you aren't satisfied with the build tool and configuration choices, you can `eject` at any time. This

Instead, it will copy all the configuration files and the transitive dependencies (webpack, Babel, ESL

You don't have to ever use `eject`. The curated feature set is suitable for small and middle deploym

Learn More

You can learn more in the [Create React App documentation](https://facebook.github.io/create-reac

To learn React, check out the [React documentation](https://reactjs.org/).

```
# https://www.robotstxt.org/robotstxt.html
User-agent: *
Disallow:
```

```
body {
  margin: 0;
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',
    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  background-color: rgb(224, 232, 240);
}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
    monospace;
}
```

```
// jest-dom adds custom jest matchers for asserting on DOM nodes.
// allows you to do things like:
// expect(element).toHaveTextContent(/react/i)
// learn more: https://github.com/testing-library/jest-dom
import '@testing-library/jest-dom';
```

```
.App {
  text-align: center;
  margin: 10px;
}
```

```
body {
  margin: 0;
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',
    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  background-color: rgb(224, 232, 240);
}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
    monospace;
}
```

```
import { render, screen } from '@testing-library/react';
import App from './App';

test('renders learn react link', () => {
  render(<App />);
  const linkElement = screen.getByText(/learn react/i);
  expect(linkElement).toBeInTheDocument();
});
```

```
const reportWebVitals = onPerfEntry => {
  if (onPerfEntry && onPerfEntry instanceof Function) {
    import('web-vitals').then(({ getCLS, getFID, getFCP, getLCP, getTTFB }) => {
      getCLS(onPerfEntry);
      getFID(onPerfEntry);
      getFCP(onPerfEntry);
      getLCP(onPerfEntry);
      getTTFB(onPerfEntry);
    });
  }
};

export default reportWebVitals;
```

```
.App {
  text-align: center;
  margin: 10px;
}
```

```
body {
  margin: 0;
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',
    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  background-color: rgb(224, 232, 240);
}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
    monospace;
}
```

```
.Display {
  background-color: rgb(224, 232, 240);
}
```

```
import React, { useState, useEffect } from 'react';
import axios from "axios";
import styles from './display.module.css';
import { Typography, Accordion, AccordionSummary, AccordionDetails } from '@mui/material';
import ExpandMoreIcon from '@mui/icons-material/ExpandMore';


const Display = () => {
  const [data, setData] = useState([]);
  const queryParams = new URLSearchParams(window.location.search);
  const id = queryParams.get('id');

  useEffect(() => {
    axios
    .get(`http://127.0.0.1:5000/data/${id}`)
    .then((response) => {
      setData(response.data.content);
    })
    .catch((error) => {
      console.error(error);
    });
  }, []);

  function renderItems(items) {
    return (
      <div>
        {Array.isArray(items) ?
          (items.length > 1 ?
            <div>
              {items.map((item, i) => {
                return (
                  <div key={item.id}>
                    <Typography sx={{ textAlign: 'left' }}>({i+1})</Typography>
                    {renderItems(item)}
                  </div>
                )
              })}
            </div>
            :
            renderItems(items[0])
          )
          :
          (typeof items === 'object' ? (
            <ul>
              {Object.keys(items).map((keyName, i) => (
                <Accordion
                  sx={{
```

```
from pydantic import BaseModel, Field

import dspy
from dspy.functional import TypedPredictor
from dspy.teleprompt import LabeledFewShot

turbo = dspy.OpenAI(model='gpt-3.5-turbo')
colbertv2_wiki17_abstracts = dspy.ColBERTv2(url='http://20.102.90.50:2017/wiki17_abstracts')
dspy.settings.configure(lm=turbo, rm=colbertv2_wiki17_abstracts)

class SyntheticFact(BaseModel):
    fact: str = Field(..., description="a statement")
    varacity: bool = Field(..., description="is the statement true or false")

class ExampleSignature(dspy.Signature):
    """Generate an example of a synthetic fact."""
    fact: SyntheticFact = dspy.OutputField()

generator = TypedPredictor(ExampleSignature)
examples = generator(config=dict(n=10))

# If you have examples and want more
existing_examples = [
    dspy.Example(fact="The sky is blue", varacity=True),
    dspy.Example(fact="The sky is green", varacity=False),
]
trained = LabeledFewShot().compile(student=generator, trainset=existing_examples)

augmented_examples = trained(config=dict(n=10))
```

```
import dspy
from dspy.datasets import HotPotQA

# Load the dataset.
dataset = HotPotQA(train_seed=1, train_size=200, eval_seed=2023, dev_size=200, test_size=0, k

# Tell DSPy that the 'question' field is the input. Any other fields are labels and/or metadata.
trainset = [x.without('id', 'type').with_inputs('question') for x in dataset.train]
devset = [x.without('id', 'type').with_inputs('question') for x in dataset.dev]
valset, devset = devset[:50], devset[50:]


# Define the signature for automatic assessments.
class Assess(dspy.Signature):
    """Assess the quality of a tweet along the specified dimension."""

    context = dspy.InputField(desc='ignore if N/A')
    assessed_text = dspy.InputField()
    assessment_question = dspy.InputField()
    assessment_answer = dspy.OutputField(desc="Yes or No")


gpt4T = dspy.OpenAI(model='gpt-4-1106-preview', max_tokens=1000, model_type='chat')
retrieve = dspy.Retrieve(k=5)
METRIC = None

def metric(gold, pred, trace=None):
    question, answer, tweet = gold.question, gold.answer, pred.output
    context = retrieve(question).passages

    engaging = "Does the assessed text make for a self-contained, engaging tweet?"
    faithful = "Is the assessed text grounded in the context? Say no if it includes significant facts not
    correct = f"The text above is should answer `{question}`. The gold answer is `{answer}`."
    correct = f"{correct} Does the assessed text above contain the gold answer?"

    with dspy.context(lm=gpt4T):
        faithful = dspy.Predict(Assess)(context=context, assessed_text=tweet, assessment_question=
        correct =  dspy.Predict(Assess)(context='N/A', assessed_text=tweet, assessment_question=c
        engaging = dspy.Predict(Assess)(context='N/A', assessed_text=tweet, assessment_question=

    correct, engaging, faithful = (m.assessment_answer.split()[0].lower() == 'yes' for m in [correct, en
    score = (correct + engaging + faithful) if correct and (len(tweet) <= 280) else 0

    if METRIC is not None:
        if METRIC == 'correct': return correct
        if METRIC == 'engaging': return engaging
        if METRIC == 'faithful': return faithful
```

```
import inspect
import logging
import math
import os
import random
import shutil
import sys

import numpy as np

try:
    from IPython.core.magics.code import extract_symbols
except ImportError:
    # Won't be able to read code from juptyer notebooks
    extract_symbols = None

import dspy
from dspy.predict.parameter import Parameter
from dspy.teleprompt.bootstrap import BootstrapFewShot, LabeledFewShot

"""
This file consists of helper functions for our variety of optimizers.
"""

### OPTIMIZER TRAINING UTILS ###


def create_minibatch(trainset, batch_size=50):
    """Create a minibatch from the trainset."""

    # Ensure batch_size isn't larger than the size of the dataset
    batch_size = min(batch_size, len(trainset))

    # Randomly sample indices for the mini-batch
    sampled_indices = random.sample(range(len(trainset)), batch_size)

    # Create the mini-batch using the sampled indices
    minibatch = [trainset[i] for i in sampled_indices]

    return minibatch


def eval_candidate_program(batch_size, trainset, candidate_program, evaluate):
    """Evaluate a candidate program on the trainset, using the specified batch size."""
    # Evaluate on the full trainset
    if batch_size >= len(trainset):
        score = evaluate(candidate_program, devset=trainset, display_table=0)
```

```
{
  "generate_answer": {
    "lm": null,
    "traces": [],
    "train": [],
    "demos": [
      {
        "augmented": true,
        "context": "It is not true that there is not a single person walking in the city.",
        "question": "Can we logically conclude for sure that it is not true that there is not a single celeb
        "rationale": "produce the answer. We know that the double negative in the context implies that
        "answer": "No"
      },
      {
        "augmented": true,
        "context": "the boy, not girl, will play an trombone, but not for another week",
        "question": "Can we logically conclude for sure that the boy, not girl, will play an instrument, bu
        "rationale": "produce the answer. We know that the boy will play a trombone, which is a type o
        "answer": "Yes"
      },
      {
        "augmented": true,
        "context": "A man is not holding anything in his hands.",
        "question": "Can we logically conclude for sure that a man is not holding beverages in his hand
        "rationale": "produce the answer. We know that the man is not holding anything in his hands. B
        "answer": "Yes"
      },
      {
        "augmented": true,
        "context": "There is not a boat nearby.",
        "question": "Can we logically conclude for sure that there is not a speedboat nearby?",
        "rationale": "produce the answer. We know that there is not a boat nearby. A speedboat is a ty
        "answer": "Yes"
      },
      {
        "augmented": true,
        "context": "The man is not listening to music.",
        "question": "Can we logically conclude for sure that the man is not listening to rockabilly?",
        "rationale": "produce the answer. We know that the man is not listening to music. Rockabilly is
        "answer": "Yes"
      },
      {
        "augmented": true,
        "context": "There is not a single person walking in the city.",
        "question": "Can we logically conclude for sure that there is not a single mover walking in the c
        "rationale": "produce the answer. We know that there is not a single person walking in the city.
        "answer": "Yes"
```

```
import multiprocessing
from typing import Optional


class PythonREPL:
    @classmethod
    def worker(cls, code: str, globals, locals, queue):
        try:
            exec(code, globals, locals)
            queue.put(None)
        except Exception as e:
            queue.put(repr(e))

    def run(self, command: str, timeout=5, globals={}, locals=None) -> Optional[str]:
        queue: multiprocessing.Queue = multiprocessing.Queue()
        p = multiprocessing.Process(
            target=self.worker, args=(command, globals, locals, queue),
        )
        p.start()
        p.join(timeout)
        if p.is_alive():
            p.terminate()
            return "TimeoutError"
        return queue.get_nowait()


# We had to move the code above to a class for pickle reasons
def execute_code(command: str, timeout=5, globals={}, locals=None) -> Optional[str]:
    return PythonREPL().run(command, timeout, globals, locals)
```

instruction,context,response,category
"Who were the children of the legendary Garth Greenhand, the High King of the First Men in the se
Give me a list of basic ingredients for baking cookies,,"To bake cookies you will need flour, sugar, e
Write a funny and whimsical horoscope reading,,"The stars say you should be patient, and that it ca
Who was the first person to do spacewalk?,,"Alexei Leonov, a soviet cosmonaut was the first perso
What causes an airplane wing to fly?,,"As Bernoulli's principal explains, because of the shape, air m
Could a nuclear explosion change the Earth's rotation?,,"Nuclear explosions release more energy t
"In music theory, what is the circle of fifths?",,"The circle of fifths is a way of organizing the 12 chro
What should I do on a free afternoon in San Francisco?,,"You can visit and take photos at San Fran
Extract the dates from the following.,"YouTube has been led by a CEO since its founding in 2005, a
As per the passage whose works did Narendranath study?,"In 1871, at the age of eight, Narendran
What is the third limb of yoga?,"sanas originated in India. In his Yoga Sutras, Patanjali (c. 2nd to 4t
"Summarize the meaning of ""Lovers"" in the slogan ""Virginia is for lovers""","""Virginia is for Lover

History
A team led by David N. Martin and George Woltz of Martin and Woltz Inc. of Richmond, Virginia cre

In 1969, the Virginia State Travel Service (now the Virginia Tourism Corporation) adopted the ""Virg

In 2009, ""Virginia is for Lovers"" was inducted into the Madison Avenue Advertising Walk of Fame,

In popular culture
The slogan has been mentioned by a variety of artists over the years. In 2005, post-hardcore band
- The ""lovers"" in the slogan was originally attached to qualifiers such as history lovers, beach love
- Contrary to claims, the word ""Lovers"" in the slogan bears no reference to the United States Supr
why do human beings have a tendency to think they are always right?,,"As a first attempt to answe
Sell me this pencil,,"I·m not sure that this pencil is right for you. First, I would need to know more a
What are the merits of skipping breakfast? Can you target your answer to a typical American and d
How do you make wine?,"Winemaking (also wine making) or vinification is the production of wine, s

Winemaking can be divided into two general categories: still wine production (without carbonation)
Which record label created Vinyl,"The LP (from ""long playing"" or ""long play"") is an analog sound
What record label did 101 Damnations go through?,"101 Damnations is the debut studio album by
"Given this paragraph, how many public high schools are in Arlington, Virginia?","Arlington Public S
Who was the first Time Magazine Man of the Year?,,Charles Lindbergh was named Time Magazine
What are some examples of different types of beer?,,"Blonde, Pilsner, Ale, Lager, Pale Ale, India P
Why are cookies delicious?,,"Cookies are delicious typically because they contain fat (oils usually),
Please share some important points to remember for Oscar awards from the given text,"The Acade
2. The Academy of Motion Picture Arts and Sciences (AMPAS) bestows them annually in appreciat
3. Many people consider the Academy Awards to be the most prestigious and important honours in
4. The Oscar trophy has an Art Deco-styled rendering of a knight.",summarization
"List some of the characters on The X-Files who were infected by, but did not die from, exposure to
"Extract the names of persons and countries in two different lists separated by commas: ""Sunday·
Countries: Russia, Ukraine",general_qa
Write a tribute to my high school swim coach.,,"I·ve known Coach for most of my life, growing up in

Coach left a lasting, positive impact on so many of us. Pushing us not only to be better swimmers,

```
{
  "prog": {
    "lm": null,
    "traces": [],
    "train": [],
    "demos": [
      {
        "augmented": true,
        "question": "Mark is baking bread. He has to let it rise for 120 minutes twice. He also needs to
        "rationale": "find the total time it takes Mark to finish making the bread. We know that he has t
        "answer": "280"
      },
      {
        "augmented": true,
        "question": "Ben has $2000 for his business operations costs. He orders goods from his suppl
        "rationale": "find the remaining amount of money. We first start with the total amount of money
        "answer": "1000"
      },
      {
        "augmented": true,
        "question": "Debbie works at a post office packing boxes to mail. Each large box takes 4 feet o
        "rationale": "find the total amount of tape Debbie used. We know that she used 4 feet of tape f
        "answer": "44 feet"
      },
      {
        "augmented": true,
        "question": "Heloise has dogs and cats in the ratio of 10:17, with the total number of pets being
        "rationale": "find the number of dogs Heloise remains with. We know that the total number of p
        "answer": "60"
      },
      {
        "question": "Jorge is 24 years younger than Simon. In 2005, Jorge is 16 years old. In 2010, ho
        "gold_reasoning": "In 2005, Simon was 16+24 = <<16+24=40>>40 years old. There are 2010-
        "answer": "45"
      },
      {
        "question": "Sally is selling boxes of crackers for her scout troop's fund-raiser. If she sells 50%
        "gold_reasoning": "Let S be the number of boxes sold on Saturday. S+1.5S=150 2.5S=150 S=
        "answer": "60"
      },
      {
        "question": "Audrey is 7 years older than Heracles. In 3 years, Audrey will be twice as old as H
        "gold_reasoning": "Since Audrey is older than Heracles by 7 years, in 3 years he will be 10 ye
        "answer": "10"
      },
      {
        "question": "Billy is breeding mice for an experiment. He starts with 8 mice, who each have 6 p
```

Integration Examples
This folder contains a cookbook of using DSPy with 3rd party integrations.

DSPy currently supports integrations with: (sorted alphabetically):
- ChromadbRM
- ClarifaiRM
- MarqoRM
- MongoDBAtlasRM
- PineconeRM
- QdrantRM
- VectaraRM
- WeaviateRM
- YouRM
- MilvusRM

```
# first line: 52
@functools.lru_cache(maxsize=None)
@NotebookCacheMemory.cache
def colbertv2_get_request_v2_wrapped(*args, **kwargs):
    return colbertv2_get_request_v2(*args, **kwargs)
```

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

```
# first line: 52
@functools.lru_cache(maxsize=None)
@NotebookCacheMemory.cache
def colbertv2_get_request_v2_wrapped(*args, **kwargs):
    return colbertv2_get_request_v2(*args, **kwargs)
```

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

```
# first line: 52
@functools.lru_cache(maxsize=None)
@NotebookCacheMemory.cache
def colbertv2_get_request_v2_wrapped(*args, **kwargs):
    return colbertv2_get_request_v2(*args, **kwargs)
```

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

{"duration": 0.008608818054199219, "input_args": {"**": "{}", "*": "['http://20.102.90.50:2017/wiki17_

```
{
  "customizations": {
    "vscode": {
      "extensions": [
        "ms-vscode-remote.vscode-remote-extensionpack",
        "charliermarsh.ruff",
        "ms-azuretools.vscode-docker",
        "ms-toolsai.jupyter",
        "ms-python.mypy-type-checker",
        "ms-vsliveshare.vsliveshare",
        "ms-python.python",
        "eamodio.gitlens",
        "github.vscode-pull-request-github"
      ],
      "settings": {
        "git.autofetch": true,
        "python.defaultInterpreterPath": "${workspaceFolder}/.venv/bin/python",
        "python.testing.pytestEnabled": true,
        "terminal.integrated.defaultProfile.linux": "zsh"
      }
    }
  },
  "features": {
    "ghcr.io/devcontainers-contrib/features/tmux-apt-get:1": {},
    "ghcr.io/devcontainers-contrib/features/zsh-plugins:0": {
      "omzPlugins": "https://github.com/zsh-users/zsh-syntax-highlighting.git https://github.com/zsh-u
      "plugins": "zsh-syntax-highlighting zsh-autosuggestions"
    },
    "ghcr.io/devcontainers/features/git:1": {},
    "ghcr.io/devcontainers/features/github-cli:1": {},
    "ghcr.io/schlich/devcontainer-features/powerlevel10k:1": {}
  },
  "image": "mcr.microsoft.com/devcontainers/python:3.9",
  "name": "Python 3",
  "postCreateCommand": "chmod +x ./.devcontainer/post_create.sh && ./.devcontainer/post_create
}
```

```
from rich.console import Console
from rich.text import Text

text = Text()
console = Console()

text.append(
    "Be sure to check that you have the Poetry Python interpreter selected in VSCode otherwise"
    " your terminal will not include the necessary packages to develop. If you have the correct interp
    " configured, you'll see an interpreter that looks like this: ",
    style="bold yellow",
)
text.append("3.9.18 ('.venv': Poetry)", style="code")
text.append(
    ". If you have previously selected another Python interpreter for the VSCode workspace, you ma
    " the Poetry Python interpreter.\n\nTo do so, follow the instructions at the following link and choo
    " virtual environment interpreter.",
    style="bold yellow",
)

console.print(text)
console.print(
    "VSCode Python Interpreter Documentation\n",
    style="link https://code.visualstudio.com/docs/python/environments#_working-with-python-interp
)
```

```
#!/bin/sh

set -e  # Exit immediately if a command exits with a non-zero status.

git config --global --add safe.directory /workspaces/dspy

pip install poetry==1.7.1
poetry config installer.max-workers 4
poetry config virtualenvs.in-project true

poetry install --with dev

sudo apt update
sudo apt-get -y install python3-distutils

poetry run pre-commit install --install-hooks

personalization_script="./.devcontainer/.personalization.sh"

# Developers can place a personalization script in the location specified above
# to further customize their dev container
if [ -f "$personalization_script" ]; then
    echo "File $personalization_script exists. Running the script..."
    chmod +x "$personalization_script"
    $personalization_script
fi

chmod +x ./.devcontainer/interpreter_warning.py && poetry run python ./.devcontainer/interpreter_w
```

```
from typing import Any

import dspy


class TestIntroIntegration:
    def test_dspy_workflow(self) -> None:
        self.setup_dspy()

        dev_example, dev_set, training_set = self.assert_dataset_loading()

        self.assert_basic_qa(dev_example)

        self.assert_retrieval(dev_example)

        self.assert_compilation(dev_set, training_set)

    def assert_compilation(self, devset, trainset) -> None:
        class GenerateAnswer(dspy.Signature):
            """Answer questions with short factoid answers."""

            context = dspy.InputField(desc="may contain relevant facts")
            question = dspy.InputField()
            answer = dspy.OutputField(desc="often between 1 and 5 words")

        class RAG(dspy.Module):
            def __init__(self, num_passages=3):
                super().__init__()

                self.retrieve = dspy.Retrieve(k=num_passages)
                self.generate_answer = dspy.ChainOfThought(GenerateAnswer)

            def forward(self, question):
                context = self.retrieve(question).passages
                prediction = self.generate_answer(context=context, question=question)
                return dspy.Prediction(context=context, answer=prediction.answer)

        from dspy.teleprompt import BootstrapFewShot

        # Validation logic: check that the predicted answer is correct.
        # Also check that the retrieved context actually contains that answer.
        def validate_context_and_answer(example, pred, trace=None):  # noqa
            answer_em = dspy.evaluate.answer_exact_match(example, pred)
            answer_pm = dspy.evaluate.answer_passage_match(example, pred)
            return answer_em and answer_pm

        # Set up a basic teleprompter, which will compile our RAG program.
```

```
from .aggregation import majority
from .chain_of_thought import ChainOfThought
from .chain_of_thought_with_hint import ChainOfThoughtWithHint
from .knn import KNN
from .multi_chain_comparison import MultiChainComparison
from .predict import Predict
from .program_of_thought import ProgramOfThought
from .react import ReAct
from .retry import Retry
```

```
from typing import Callable

from dsp.templates import Field, TemplateV2, format_answers, passages2text


class Type:
    """A primitive datatype that defines and represents a prompt label."""

    def __init__(self, prefix: str, desc: str, format=None) -> None:
        self.prefix = prefix
        self.desc = desc
        self.format = format

    def __call__(self, **kwargs):
        kwargs = {**self.__dict__, **kwargs}
        return Type(**kwargs)

    def __eq__(self, __value: object) -> bool:
        return isinstance(__value, Type) and self.__dict__ == __value.__dict__


class Template(TemplateV2):
    """A template datatype that represents the structure of communicate with the LM."""

    def __init__(self, instructions: str, **kwargs):
        self.instructions = instructions
        self.kwargs = kwargs

        self.fields: list[Field] = []
        self.format_handlers: dict[str, Callable] = {
            "context": passages2text,
            "passages": passages2text,
            "answers": format_answers,
        }

        for key, value in kwargs.items():
            prefix: str = value.prefix
            separator: str = (
                " " if prefix.rstrip() == prefix and len(prefix) > 0 else prefix[len(prefix.rstrip()) :]
            )
            field = Field(
                name=prefix.strip(),
                description=value.desc,
                input_variable=key,
                output_variable=key,
                separator=separator,
            )
```

```
import inspect
import logging
import math
import os
import random
import shutil
import sys

import numpy as np

try:
    from IPython.core.magics.code import extract_symbols
except ImportError:
    # Won't be able to read code from juptyer notebooks
    extract_symbols = None

import dspy
from dspy.predict.parameter import Parameter
from dspy.teleprompt.bootstrap import BootstrapFewShot, LabeledFewShot

"""
This file consists of helper functions for our variety of optimizers.
"""

### OPTIMIZER TRAINING UTILS ###


def create_minibatch(trainset, batch_size=50):
    """Create a minibatch from the trainset."""

    # Ensure batch_size isn't larger than the size of the dataset
    batch_size = min(batch_size, len(trainset))

    # Randomly sample indices for the mini-batch
    sampled_indices = random.sample(range(len(trainset)), batch_size)

    # Create the mini-batch using the sampled indices
    minibatch = [trainset[i] for i in sampled_indices]

    return minibatch


def eval_candidate_program(batch_size, trainset, candidate_program, evaluate):
    """Evaluate a candidate program on the trainset, using the specified batch size."""
    # Evaluate on the full trainset
    if batch_size >= len(trainset):
        score = evaluate(candidate_program, devset=trainset, display_table=0)
```

```
from .aggregation import majority
from .chain_of_thought import ChainOfThought
from .chain_of_thought_with_hint import ChainOfThoughtWithHint
from .knn import KNN
from .multi_chain_comparison import MultiChainComparison
from .predict import Predict
from .program_of_thought import ProgramOfThought
from .react import ReAct
from .retry import Retry
```

```
import re
from collections import namedtuple
from typing import Any, Union

import dsp
from dsp.primitives.demonstrate import Example

from .utils import format_answers, passages2text

Field = namedtuple("Field", "name separator input_variable output_variable description")

# TODO: de-duplicate with dsp/templates/template.py


class TemplateV2:
    def __init__(
        self,
        template,
        format_handlers={
            "passages": passages2text,
            "context": passages2text,
            "answer": format_answers,
            "answers": format_answers,
        },
    ):
        self.format_handlers = format_handlers

        template = template.strip()

        self.instructions = re.search("(.*)\n", template).group(1)
        template = template[len(self.instructions) :].strip()

        self.fields = []
        while len(template) > 0:
            match = re.search("(.*)(\s){(.*)}\s(.*\${.*})", template)
            if match is not None:
                name = match.group(1)
                separator = match.group(2)
                variable = match.group(3)
                description = match.group(4)
            else:
                match = re.search("(.*)(\s){(.*)}", template)
                if match is not None:
                    name = match.group(1)
                    separator = match.group(2)
                    variable = match.group(3)
                    description = None
```

```
from pathlib import Path
from typing import Any, Optional, Union

from dsp.modules.lm import LM

## Utility functions to load models


def load_tensorrt_model(
    engine_dir: Union[str, Path],
    use_py_session: Optional[bool] = False,
    **kwargs,
) -> tuple[Any, dict]:
    import tensorrt_llm
    from tensorrt_llm.runtime import ModelRunner, ModelRunnerCpp

    runtime_rank = tensorrt_llm.mpi_rank()
    runner_cls = ModelRunner if use_py_session else ModelRunnerCpp
    runner_kwargs = {
        "engine_dir": engine_dir,
        "lora_dir": kwargs.get("lora_dir", None),
        "rank": runtime_rank,
        "lora_ckpt_source": kwargs.get("lora_ckpt_source", "hf"),
    }

    if not use_py_session:
        engine_cpp_kwargs = {}
        defaults = {
            "max_batch_size": 1,
            "max_input_len": 1024,
            "max_output_len": 1024,
            "max_beam_width": 1,
            "max_attention_window_size": None,
            "sink_token_length": None,
        }

        for key, value in defaults.items():
            engine_cpp_kwargs[key] = kwargs.get(key, value)
        runner_kwargs.update(**engine_cpp_kwargs)

    runner = runner_cls.from_dir(**runner_kwargs)
    return runner, runner_kwargs


def tokenize(prompt: Union[list[dict], str], tokenizer: Any, **kwargs) -> list[int]:
    defaults = {
        "add_special_tokens": False,
```

```
import datetime
import hashlib
from typing import Any, Literal

import requests

from dsp.modules.lm import LM


def post_request_metadata(model_name, prompt):
    """Creates a serialized request object for the Ollama API."""
    timestamp = datetime.datetime.now().timestamp()
    id_string = str(timestamp) + model_name + prompt
    hashlib.sha1().update(id_string.encode("utf-8"))
    id_hash = hashlib.sha1().hexdigest()
    return {"id": f"chatcmpl-{id_hash}", "object": "chat.completion", "created": int(timestamp), "model"


class OllamaLocal(LM):
    """Wrapper around a locally hosted Ollama model (API: https://github.com/jmorganca/ollama/blo
    Returns dictionary info in the OpenAI API style (https://platform.openai.com/docs/api-reference/c

    Args:
        model (str, optional): Name of Ollama model. Defaults to "llama2".
        model_type (Literal["chat", "text"], optional): The type of model that was specified. Mainly to de
        base_url (str):  Protocol, host name, and port to the served ollama model. Defaults to "http://lo
        timeout_s (float): Timeout period (in seconds) for the post request to llm.
        **kwargs: Additional arguments to pass to the API.
    """

    def __init__(
        self,
        model: str = "llama2",
        model_type: Literal["chat", "text"] = "text",
        base_url: str = "http://localhost:11434",
        timeout_s: float = 120,
        temperature: float = 0.0,
        max_tokens: int = 150,
        top_p: int = 1,
        top_k: int = 20,
        frequency_penalty: float = 0,
        presence_penalty: float = 0,
        n: int = 1,
        num_ctx: int = 1024,
        **kwargs,
    ):
        super().__init__(model)
```

```
import functools
from typing import Any, List, Optional, Union

import requests

from dsp.modules.cache_utils import CacheMemory, NotebookCacheMemory
from dsp.utils import dotdict

# TODO: Ideally, this takes the name of the index and looks up its port.


class ColBERTv2:
    """Wrapper for the ColBERTv2 Retrieval."""

    def __init__(
        self,
        url: str = "http://0.0.0.0",
        port: Optional[Union[str, int]] = None,
        post_requests: bool = False,
    ):
        self.post_requests = post_requests
        self.url = f"{url}:{port}" if port else url

    def __call__(
        self, query: str, k: int = 10, simplify: bool = False,
    ) -> Union[list[str], list[dotdict]]:
        if self.post_requests:
            topk: list[dict[str, Any]] = colbertv2_post_request(self.url, query, k)
        else:
            topk: list[dict[str, Any]] = colbertv2_get_request(self.url, query, k)

        if simplify:
            return [psg["long_text"] for psg in topk]

        return [dotdict(psg) for psg in topk]


@CacheMemory.cache
def colbertv2_get_request_v2(url: str, query: str, k: int):
    assert (
        k <= 100
    ), "Only k <= 100 is supported for the hosted ColBERTv2 server at the moment."

    payload = {"query": query, "k": k}
    res = requests.get(url, params=payload, timeout=10)

    topk = res.json()["topk"][:k]
```

```
"""Module for interacting with Snowflake Cortex."""
import json
from typing import Any

import backoff
from pydantic_core import PydanticCustomError

from dsp.modules.lm import LM

try:
    from snowflake.snowpark import Session
    from snowflake.snowpark import functions as snow_func

except ImportError:
    pass


def backoff_hdlr(details) -> None:
    """Handler from https://pypi.org/project/backoff ."""
    print(
        f"Backing off {details['wait']:0.1f} seconds after {details['tries']} tries ",
        f"calling function {details['target']} with kwargs",
        f"{details['kwargs']}",
    )


def giveup_hdlr(details) -> bool:
    """Wrapper function that decides when to give up on retry."""
    if "rate limits" in str(details):
        return False
    return True


class Snowflake(LM):
    """Wrapper around Snowflake's CortexAPI.

    Currently supported models include 'snowflake-arctic','mistral-large','reka-flash','mixtral-8x7b',
    'llama2-70b-chat','mistral-7b','gemma-7b','llama3-8b','llama3-70b','reka-core'.
    """

    def __init__(self, model: str = "mixtral-8x7b", credentials=None, **kwargs):
        """Parameters

        ----------
        model : str
            Which pre-trained model from Snowflake to use?
            Choices are 'snowflake-arctic','mistral-large','reka-flash','mixtral-8x7b','llama2-70b-chat','mis
```

```
import os
from functools import wraps
from pathlib import Path

from joblib import Memory

from dsp.utils import dotdict

cache_turn_on = os.environ.get('DSP_CACHEBOOL', 'True').lower() != 'false'


def noop_decorator(arg=None, *noop_args, **noop_kwargs):
    def decorator(func):
        @wraps(func)
        def wrapper(*args, **kwargs):
            return func(*args, **kwargs)

        return wrapper

    if callable(arg):
        return decorator(arg)
    else:
        return decorator


cachedir = os.environ.get('DSP_CACHEDIR') or os.path.join(Path.home(), 'cachedir_joblib')
CacheMemory = Memory(location=cachedir, verbose=0)

cachedir2 = os.environ.get('DSP_NOTEBOOK_CACHEDIR')
NotebookCacheMemory = dotdict()
NotebookCacheMemory.cache = noop_decorator

if cachedir2:
    NotebookCacheMemory = Memory(location=cachedir2, verbose=0)


if not cache_turn_on:
    CacheMemory = dotdict()
    CacheMemory.cache = noop_decorator

    NotebookCacheMemory = dotdict()
    NotebookCacheMemory.cache = noop_decorator
```

```
import logging
import os
from typing import Any, Optional

import backoff

from dsp.modules.lm import LM

try:
    import anthropic

    anthropic_rate_limit = anthropic.RateLimitError
except ImportError:
    anthropic_rate_limit = Exception

logger = logging.getLogger(__name__)
BASE_URL = "https://api.anthropic.com/v1/messages"


def backoff_hdlr(details):
    """Handler from https://pypi.org/project/backoff/."""
    print(
        "Backing off {wait:0.1f} seconds after {tries} tries "
        "calling function {target} with kwargs "
        "{kwargs}".format(**details),
    )


def giveup_hdlr(details):
    """Wrapper function that decides when to give up on retry."""
    if "rate limits" in details.message:
        return False
    return True


class Claude(LM):
    """Wrapper around anthropic's API. Supports both the Anthropic and Azure APIs."""

    def __init__(
        self,
        model: str = "claude-3-opus-20240229",
        api_key: Optional[str] = None,
        api_base: Optional[str] = None,
        **kwargs,
    ):
        super().__init__(model)
        try:
```

```
"""AWS providers for LMs."""

from abc import ABC, abstractmethod
from typing import Any, Optional

import backoff

try:
    import boto3
    from botocore.exceptions import ClientError
    ERRORS = (ClientError,)

except ImportError:
    ERRORS = (Exception,)


def backoff_hdlr(details):
    """Handler from https://pypi.org/project/backoff/."""
    print(
        "Backing off {wait:0.1f} seconds after {tries} tries "
        "calling function {target} with kwargs "
        "{kwargs}".format(**details),
    )


def giveup_hdlr(details):
    """Wrapper function that decides when to give up on retry."""
    if "max retries" in details.args[0]:
        return False
    return True

class AWSProvider(ABC):
    """This abstract class adds support for AWS model providers such as Bedrock and SageMaker.
    The subclasses such as Bedrock and Sagemaker implement the abstract method _call_model a
    Usage Example:
        bedrock = dspy.Bedrock(region_name="us-west-2")
        bedrock_mixtral = dspy.AWSMistral(bedrock, "mistral.mixtral-8x7b-instruct-v0:1", **kwargs)
        bedrock_haiku = dspy.AWSAnthropic(bedrock, "anthropic.claude-3-haiku-20240307-v1:0", **k
        bedrock_llama2 = dspy.AWSMeta(bedrock, "meta.llama2-13b-chat-v1", **kwargs)

        sagemaker = dspy.Sagemaker(region_name="us-west-2")
        sagemaker_model = dspy.AWSMistral(sagemaker, "<YOUR_ENDPOINT_NAME>", **kwargs)
    """

    def __init__(
        self,
        region_name: str,
```

```
"""AWS models for LMs."""

from __future__ import annotations

import json
import logging
from abc import abstractmethod
from typing import Any

from dsp.modules.aws_providers import AWSProvider, Bedrock, Sagemaker
from dsp.modules.lm import LM

# Heuristic translating number of chars to tokens
# ~4 chars = 1 token
CHARS2TOKENS: int = 4


class AWSModel(LM):
    """This class adds support for an AWS model.

    It is an abstract class and should not be instantiated directly.
    Instead, use one of the subclasses - AWSMistral, AWSAnthropic, or AWSMeta.
    The subclasses implement the abstract methods _create_body and _call_model
    and work in conjunction with the AWSProvider classes Bedrock and Sagemaker.
    Usage Example:
        bedrock = dspy.Bedrock(region_name="us-west-2")
        bedrock_mixtral = dspy.AWSMistral(bedrock, "mistral.mixtral-8x7b-instruct-v0:1", **kwargs)
        bedrock_haiku = dspy.AWSAnthropic(bedrock, "anthropic.claude-3-haiku-20240307-v1:0", **k
        bedrock_llama2 = dspy.AWSMeta(bedrock, "meta.llama2-13b-chat-v1", **kwargs)

        sagemaker = dspy.Sagemaker(region_name="us-west-2")
        sagemaker_model = dspy.AWSMistral(sagemaker, "<YOUR_ENDPOINT_NAME>", **kwargs)
    """

    def __init__(
        self,
        model: str,
        max_context_size: int,
        max_new_tokens: int,
        **kwargs,
    ) -> None:
        """_summary_.

        Args:
            model (str, optional): An LM name, e.g., a bedrock name or an AWS endpoint.
            max_context_size (int): The maximum context size in tokens.
            max_new_tokens (int): The maximum number of tokens to be sampled from the LM.
```

```
from typing import Any, Optional

import backoff

from dsp.modules.lm import LM

try:
    import mistralai
    from mistralai.client import MistralClient
    from mistralai.exceptions import MistralAPIException
    from mistralai.models.chat_completion import ChatCompletionResponse, ChatMessage
    mistralai_api_error = MistralAPIException
except ImportError:
    mistralai_api_error = Exception


def backoff_hdlr(details):
    """Handler from https://pypi.org/project/backoff/"""
    print(
        "Backing off {wait:0.1f} seconds after {tries} tries "
        "calling function {target} with kwargs "
        "{kwargs}".format(**details),
    )


def giveup_hdlr(details):
    """wrapper function that decides when to give up on retry"""
    if "rate limits" in details.message:
        return False
    return True


class Mistral(LM):
    """Wrapper around Mistral AI's API.

    Currently supported models include `mistral-small-latest`, `mistral-medium-latest`, `mistral-large-
    """

    def __init__(
        self,
        model: str = "mistral-medium-latest",
        api_key: Optional[str] = None,
        **kwargs,
    ):
        """
        Parameters
        ----------
```

```
import functools
import json
from typing import Literal, Optional

import openai

from dsp.modules.cache_utils import CacheMemory, NotebookCacheMemory, cache_turn_on
from dsp.modules.gpt3 import GPT3

try:
    import openai.error
    from openai.openai_object import OpenAIObject
    ERRORS = (openai.error.RateLimitError, openai.error.ServiceUnavailableError, openai.error.AP
except Exception:
    ERRORS = (openai.RateLimitError, openai.APIError)
    OpenAIObject = dict


def backoff_hdlr(details):
    """Handler from https://pypi.org/project/backoff/"""
    print(
        "Backing off {wait:0.1f} seconds after {tries} tries "
        "calling function {target} with kwargs "
        "{kwargs}".format(**details),
    )

class Databricks(GPT3):
    """Wrapper around DSPy's OpenAI Wrapper. Supports Databricks Model Serving Endpoints for C

    Args:
        model (str, required): Databricks-hosted LLM model to use.
        api_key (Optional[str], optional): Databricks authentication token. Defaults to None.
        api_base (Optional[str], optional): Databricks model serving endpoint. Defaults to None.
        model_type (Literal["chat", "text"], optional): The type of model that was specified. Mainly to de
        **kwargs: Additional arguments to pass to the OpenAI API provider.
    """

    def __init__(
        self,
        model: str,
        api_key: Optional[str] = None,
        api_base: Optional[str] = None,
        model_type: Literal["chat", "text", "embeddings"] = None,
        **kwargs,
    ):
        super().__init__(
            model=model,
```

```
"""Clarifai LM integration"""
from typing import Any, Optional

from dsp.modules.lm import LM


class ClarifaiLLM(LM):
    """Integration to call models hosted in clarifai platform.

    Args:
        model (str, optional): Clarifai URL of the model. Defaults to "Mistral-7B-Instruct".
        api_key (Optional[str], optional): CLARIFAI_PAT token. Defaults to None.
        **kwargs: Additional arguments to pass to the API provider.
    Example:
        import dspy
        dspy.configure(lm=dspy.Clarifai(model=MODEL_URL,
                                api_key=CLARIFAI_PAT,
                                inference_params={"max_tokens":100,'temperature':0.6}))
    """

    def __init__(
        self,
        model: str = "https://clarifai.com/mistralai/completion/models/mistral-7B-Instruct",
        api_key: Optional[str] = None,
        **kwargs,
    ):
        super().__init__(model)

        try:
            from clarifai.client.model import Model
        except ImportError as err:
            raise ImportError("ClarifaiLLM requires `pip install clarifai`.") from err


        self.provider = "clarifai"
        self.pat = api_key
        self._model = Model(url=model, pat=api_key)
        self.kwargs = {"n": 1, **kwargs}
        self.history: list[dict[str, Any]] = []
        self.kwargs["temperature"] = (
            self.kwargs["inference_params"]["temperature"]
            if "inference_params" in self.kwargs
            and "temperature" in self.kwargs["inference_params"]
            else 0.0
        )
        self.kwargs["max_tokens"] = (
            self.kwargs["inference_params"]["max_tokens"]
```

```
from .aggregation import majority
from .chain_of_thought import ChainOfThought
from .chain_of_thought_with_hint import ChainOfThoughtWithHint
from .knn import KNN
from .multi_chain_comparison import MultiChainComparison
from .predict import Predict
from .program_of_thought import ProgramOfThought
from .react import ReAct
from .retry import Retry
```

```
"""Module for interacting with Google Vertex AI."""

from typing import Any, Dict

import backoff
from pydantic_core import PydanticCustomError

from dsp.modules.lm import LM

try:
    import vertexai  # type: ignore[import-untyped]
    from vertexai.language_models import CodeGenerationModel, TextGenerationModel
    from vertexai.preview.generative_models import GenerativeModel
except ImportError:
    pass


def backoff_hdlr(details):
    """Handler from https://pypi.org/project/backoff/"""
    print(
        f"Backing off {details['wait']:0.1f} seconds after {details['tries']} tries "
        f"calling function {details['target']} with kwargs "
        f"{details['kwargs']}",
    )


def giveup_hdlr(details):
    """wrapper function that decides when to give up on retry"""
    if "rate limits" in details.message:
        return False
    return True


class GoogleVertexAI(LM):
    """Wrapper around GoogleVertexAI's API.

    Currently supported models include `gemini-pro-1.0`.
    """

    def __init__(
        self,
        model: str = "text-bison@002",
        **kwargs,
    ):
        """
        Parameters
        ----------
```

```
class SentenceTransformersCrossEncoder:
    """Wrapper for sentence-transformers cross-encoder model.
    """
    def __init__(
        self, model_name_or_path: str = "cross-encoder/ms-marco-MiniLM-L-12-v2",
    ):
        try:
            from sentence_transformers.cross_encoder import CrossEncoder
        except ImportError:
            raise ModuleNotFoundError(
                "You need to install sentence-transformers library to use SentenceTransformersCrossEn
            )
        self.model = CrossEncoder(model_name_or_path)

    def __call__(self, query: str, passage: list[str]) -> list[float]:
        return self.model.predict([[query, p] for p in passage]).tolist()
```

```
import logging
from typing import Any

import backoff

try:
    import groq
    from groq import Groq

    groq_api_error = (groq.APIError, groq.RateLimitError)
except ImportError:
    groq_api_error = Exception


from dsp.modules.lm import LM


def backoff_hdlr(details):
    """Handler from https://pypi.org/project/backoff/"""
    print(
        "Backing off {wait:0.1f} seconds after {tries} tries "
        "calling function {target} with kwargs "
        "{kwargs}".format(**details),
    )


class GroqLM(LM):
    """Wrapper around groq's API.

    Args:
        model (str, optional): groq supported LLM model to use. Defaults to "mixtral-8x7b-32768".
        api_key (Optional[str], optional): API provider Authentication token. use Defaults to None.
        **kwargs: Additional arguments to pass to the API provider.
    """

    def __init__(
        self,
        api_key: str,
        model: str = "mixtral-8x7b-32768",
        **kwargs,
    ):
        super().__init__(model)
        self.provider = "groq"
        if api_key:
            self.api_key = api_key
            self.client = Groq(api_key=api_key)
        else:
```

```
import os
import random
import re
import shutil
import subprocess
from typing import Literal

# from dsp.modules.adapter import TurboAdapter, DavinciAdapter, LlamaAdapter
import backoff
import requests

from dsp.modules.cache_utils import CacheMemory, NotebookCacheMemory
from dsp.modules.hf import HFModel, openai_to_hf

ERRORS = (Exception)

def backoff_hdlr(details):
    """Handler from https://pypi.org/project/backoff/"""
    print(
        "Backing off {wait:0.1f} seconds after {tries} tries "
        "calling function {target} with kwargs "
        "{kwargs}".format(**details),
    )

class HFClientTGI(HFModel):
    def __init__(self, model, port, url="http://future-hgx-1", http_request_kwargs=None, **kwargs):
        super().__init__(model=model, is_client=True)

        self.url = url
        self.ports = port if isinstance(port, list) else [port]
        self.http_request_kwargs = http_request_kwargs or {}

        self.headers = {"Content-Type": "application/json"}

        self.kwargs = {
            "model": model,
            "port": port,
            "url": url,
            "temperature": 0.01,
            "max_tokens": 75,
            "top_p": 0.97,
            "n": 1,
            "stop": ["\n", "\n\n"],
            **kwargs,
        }

        # print(self.kwargs)
```

```
import os
from typing import Any, Literal, Optional

import requests

from dsp.modules.lm import LM

SMART_ENDPOINT = "https://chat-api.you.com/smart"
RESEARCH_ENDPOINT = "https://chat-api.you.com/research"


class You(LM):
    """Wrapper around You.com's conversational Smart and Research APIs.

    Each API endpoint is designed to generate conversational
    responses to a variety of query types, including inline citations
    and web results when relevant.

    Smart Mode:
    - Quick, reliable answers for a variety of questions
    - Cites the entire web page URL

    Research Mode:
    - In-depth answers with extensive citations for a variety of questions
    - Cites the specific web page snippet relevant to the claim

    To connect to the You.com api requires an API key which
    you can get at https://api.you.com.

    For more information, check out the documentations at
    https://documentation.you.com/api-reference/.

    Args:
        endpoint: You.com conversational endpoints. Choose from "smart" or "research"
        api_key: You.com API key, if `YDC_API_KEY` is not set in the environment
    """

    def __init__(
        self,
        endpoint: Literal["smart", "research"] = "smart",
        ydc_api_key: Optional[str] = None,
    ):
        super().__init__(model="you.com")
        self.ydc_api_key = ydc_api_key or os.environ["YDC_API_KEY"]
        self.endpoint = endpoint

        # Mandatory DSPy attributes to inspect LLM call history
```

```
from abc import ABC, abstractmethod


class LM(ABC):
    """Abstract class for language models."""

    def __init__(self, model):
        self.kwargs = {
            "model": model,
            "temperature": 0.0,
            "max_tokens": 150,
            "top_p": 1,
            "frequency_penalty": 0,
            "presence_penalty": 0,
            "n": 1,
        }
        self.provider = "default"

        self.history = []

    @abstractmethod
    def basic_request(self, prompt, **kwargs):
        pass

    def request(self, prompt, **kwargs):
        return self.basic_request(prompt, **kwargs)

    def print_green(self, text: str, end: str = "\n"):
        return "\x1b[32m" + str(text) + "\x1b[0m" + end

    def print_red(self, text: str, end: str = "\n"):
        return "\x1b[31m" + str(text) + "\x1b[0m" + end

    def inspect_history(self, n: int = 1, skip: int = 0):
        """Prints the last n prompts and their completions.

        TODO: print the valid choice that contains filled output field instead of the first.
        """
        provider: str = self.provider

        last_prompt = None
        printed = []
        n = n + skip

        for x in reversed(self.history[-100:]):
            prompt = x["prompt"]
```

```
import os
from collections.abc import Iterable
from typing import Any, Optional

import backoff

from dsp.modules.lm import LM

try:
    import google.generativeai as genai
    from google.api_core.exceptions import GoogleAPICallError
    google_api_error = GoogleAPICallError
except ImportError:
    google_api_error = Exception
    # print("Not loading Google because it is not installed.")


def backoff_hdlr(details):
    """Handler from https://pypi.org/project/backoff/"""
    print(
        "Backing off {wait:0.1f} seconds after {tries} tries "
        "calling function {target} with kwargs "
        "{kwargs}".format(**details),
    )


def giveup_hdlr(details):
    """wrapper function that decides when to give up on retry"""
    if "rate limits" in details.message:
        return False
    return True


BLOCK_ONLY_HIGH = [
  {
    "category": "HARM_CATEGORY_HARASSMENT",
    "threshold": "BLOCK_ONLY_HIGH",
  },
  {
    "category": "HARM_CATEGORY_HATE_SPEECH",
    "threshold": "BLOCK_ONLY_HIGH",
  },
  {
    "category": "HARM_CATEGORY_SEXUALLY_EXPLICIT",
    "threshold": "BLOCK_ONLY_HIGH",
  },
  {
```

```
from typing import Any, Optional

import backoff

from dsp.modules.lm import LM

try:
    import cohere
    cohere_api_error = cohere.errors.UnauthorizedError
except ImportError:
    cohere_api_error = Exception
    # print("Not loading Cohere because it is not installed.")
except AttributeError:
    cohere_api_error = Exception


def backoff_hdlr(details):
    """Handler from https://pypi.org/project/backoff/"""
    print(
        "Backing off {wait:0.1f} seconds after {tries} tries "
        "calling function {target} with kwargs "
        "{kwargs}".format(**details),
    )


def giveup_hdlr(details):
    """wrapper function that decides when to give up on retry"""
    if "rate limits" in details.message:
        return False
    return True


class Cohere(LM):
    """Wrapper around Cohere's API.

    Currently supported models include `command-r-plus`, `command-r`, `command`, `command-nig
    """

    def __init__(
        self,
        model: str = "command-r",
        api_key: Optional[str] = None,
        stop_sequences: list[str] = [],
        **kwargs,
    ):
        """
        Parameters
```

```
import re
from typing import Union

from dsp.modules import LM


# This testing module was moved in PR #735 to patch Arize Phoenix logging
class DummyLM(LM):
    """Dummy language model for unit testing purposes."""

    def __init__(self, answers: Union[list[str], dict[str, str]], follow_examples: bool = False):
        """Initializes the dummy language model.

        Parameters:
        - answers: A list of strings or a dictionary with string keys and values.
        - follow_examples: If True, and the prompt contains an example exactly equal to the prompt,
                           the dummy model will return the next string in the list for each request.
        If a list is provided, the dummy model will return the next string in the list for each request.
        If a dictionary is provided, the dummy model will return the value corresponding to the key tha
        """
        super().__init__("dummy-model")
        self.provider = "dummy"
        self.answers = answers
        self.follow_examples = follow_examples

    def basic_request(self, prompt, n=1, **kwargs) -> dict[str, list[dict[str, str]]]:
        """Generates a dummy response based on the prompt."""
        dummy_response = {"choices": []}
        for _ in range(n):
            answer = None

            if self.follow_examples:
                prefix = prompt.split("\n")[-1]
                _instructions, _format, *examples, _output = prompt.split("\n---\n")
                examples_str = "\n".join(examples)
                possible_answers = re.findall(prefix + r"\s*(.*)", examples_str)
                if possible_answers:
                    # We take the last answer, as the first one is just from
                    # the "Follow the following format" section.
                    answer = possible_answers[-1]
                    print(f"DummyLM got found previous example for {prefix} with value {answer=}")
                else:
                    print(f"DummyLM couldn't find previous example for {prefix=}")

            if answer is None:
                if isinstance(self.answers, dict):
                    answer = next((v for k, v in self.answers.items() if k in prompt), None)
```

```
# from peft import PeftConfig, PeftModel
# from transformers import AutoModelForSeq2SeqLM, AutoModelForCausalLM, AutoTokenizer, Au
import os
from typing import Literal, Optional

from dsp.modules.lm import LM

# from dsp.modules.finetuning.finetune_hf import preprocess_prompt


def openai_to_hf(**kwargs):
    hf_kwargs = {}
    for k, v in kwargs.items():
        if k == "n":
            hf_kwargs["num_return_sequences"] = v
        elif k == "frequency_penalty":
            hf_kwargs["repetition_penalty"] = 1.0 - v
        elif k == "presence_penalty":
            hf_kwargs["diversity_penalty"] = v
        elif k == "max_tokens":
            hf_kwargs["max_new_tokens"] = v
        elif k == "model":
            pass
        else:
            hf_kwargs[k] = v

    return hf_kwargs


class HFModel(LM):
    def __init__(
        self,
        model: str,
        checkpoint: Optional[str] = None,
        is_client: bool = False,
        hf_device_map: Literal[
            "auto",
            "balanced",
            "balanced_low_0",
            "sequential",
        ] = "auto",
        token: Optional[str] = None,
        model_kwargs: Optional[dict] = {},
    ):
        """wrapper for Hugging Face models

        Args:
```

```
import abc
from typing import List, Optional

import numpy as np
import openai


class BaseSentenceVectorizer(abc.ABC):
    '''
    Base Class for Vectorizers. The main purpose is to vectorize text (doc/query)
    for ANN/KNN indexes. `__call__` method takes `List[Example]` as a single input, then extracts
    `field_to_vectorize` from every Example and convert them into embeddings.
    You can customize extraction logic in the `_extract_text_from_examples` method.
    '''
    # embeddings will be computed based on the string in this attribute of Example object
    field_to_vectorize = 'text_to_vectorize'

    def __init__(self) -> None:
        pass

    @abc.abstractmethod
    def __call__(self, inp_examples: List["Example"]) -> np.ndarray:
        pass

    def _extract_text_from_examples(self, inp_examples: List) -> List[str]:
        if isinstance(inp_examples[0], str):
            return inp_examples
        return [" ".join([example[key] for key in example._input_keys]) for example in inp_examples]


class SentenceTransformersVectorizer(BaseSentenceVectorizer):
    '''
    Vectorizer based on `SentenceTransformers` models. You can pick any model from this link:
    https://huggingface.co/models?library=sentence-transformers
    More details about models:
    https://www.sbert.net/docs/pretrained_models.html
    '''
    def __init__(
        self,
        model_name_or_path: str = 'all-MiniLM-L6-v2',
        vectorize_bs: int = 256,
        max_gpu_devices: int = 1,
        normalize_embeddings: bool = False,
    ):
        # this isn't a good practice, but with top-level import the whole DSP
        # module import will be slow (>5 sec), because SentenceTransformer is doing
        # it's directory/file-related magic under the hood :(
```

```
import os
import warnings
from typing import Any, Optional

import backoff

from dsp.modules.lm import LM

try:
    import premai

    premai_api_error = premai.errors.UnexpectedStatus
except ImportError:
    premai_api_error = Exception
except AttributeError:
    premai_api_error = Exception


def backoff_hdlr(details) -> None:
    """Handler for the backoff package.

    See more at: https://pypi.org/project/backoff/
    """
    print(  # noqa: T201
        "Backing off {wait:0.1f} seconds after {tries} tries calling function {target} with kwargs {kwargs}"
            **details,
        ),
    )


def giveup_hdlr(details) -> bool:
    """Wrapper function that decides when to give up on retry."""
    if "rate limits" in details.message:
        return False
    return True


def get_premai_api_key(api_key: Optional[str] = None) -> str:
    """Retrieve the PreMAI API key from a passed argument or environment variable."""
    api_key = api_key or os.environ.get("PREMAI_API_KEY")
    if api_key is None:
        raise RuntimeError(
            "No API key found. See the quick start guide at https://docs.premai.io/introduction to get yo
        )
    return api_key
```

```
from typing import Any, Union

from dsp.utils import dotdict

try:
    from azure.core.credentials import AzureKeyCredential
    from azure.search.documents import SearchClient
    from azure.search.documents._paging import SearchItemPaged
except ImportError:
    raise ImportError(
        "You need to install azure-search-documents library"
        "Please use the command: pip install azure-search-documents",
    )

# Deprecated: This module is scheduled for removal in future releases.
# Please use the AzureAISearchRM class from dspy.retrieve.azureaisearch_rm instead.
# For more information, refer to the updated documentation.

class AzureCognitiveSearch:
    """Wrapper for the Azure Cognitive Search Retrieval."""

    def __init__(
        self,
        search_service_name: str,
        search_api_key: str,
        search_index_name: str,
        field_text: str, # required field to map with "content" field in dsp framework
        field_score: str, # required field to map with "score" field in dsp framework

    ):
        self.search_service_name = search_service_name
        self.search_api_key = search_api_key
        self.search_index_name = search_index_name
        self.endpoint=f"https://{self.search_service_name}.search.windows.net"
        self.field_text = field_text # field name of the text content
        self.field_score = field_score # field name of the search score
        # Create a client
        self.credential = AzureKeyCredential(self.search_api_key)
        self.client = SearchClient(endpoint=self.endpoint,
                        index_name=self.search_index_name,
                        credential=self.credential)

    def __call__(self, query: str, k: int = 10) -> Union[list[str], list[dotdict]]:
        print("""# Deprecated: This module is scheduled for removal in future releases.
                Please use the AzureAISearchRM class from dspy.retrieve.azureaisearch_rm instead.
                For more information, refer to the updated documentation.""")
```

```
import functools
import json
import logging
from typing import Any, Callable, Literal, Optional, cast

import backoff
import openai

from dsp.modules.cache_utils import CacheMemory, NotebookCacheMemory, cache_turn_on
from dsp.modules.lm import LM

try:
    OPENAI_LEGACY = int(openai.version.__version__[0]) == 0
except Exception:
    OPENAI_LEGACY = True

try:
    import openai.error
    from openai.openai_object import OpenAIObject

    ERRORS = (
        openai.error.RateLimitError,
        openai.error.ServiceUnavailableError,
        openai.error.APIError,
    )
except Exception:
    ERRORS = (openai.RateLimitError, openai.APIError)
    OpenAIObject = dict


def backoff_hdlr(details):
    """Handler from https://pypi.org/project/backoff/"""
    print(
        "Backing off {wait:0.1f} seconds after {tries} tries "
        "calling function {target} with kwargs "
        "{kwargs}".format(**details),
    )


AzureADTokenProvider = Callable[[], str]


class AzureOpenAI(LM):
    """Wrapper around Azure's API for OpenAI.

    Args:
        api_base (str): Azure URL endpoint for model calling, often called 'azure_endpoint'.
```

```
from typing import Any

from dsp.modules.lm import LM

ibm_watsonx_ai_api_error = False

try:
    import ibm_watsonx_ai  # noqa: F401
    from ibm_watsonx_ai.foundation_models import Model  # type: ignore

except ImportError:
    ibm_watsonx_ai_api_error = Exception


class Watsonx(LM):
    """Wrapper around Watsonx AI's API.

    The constructor initializes the base class LM to support prompting requests to Watsonx models.
    This requires the following parameters:
    Args:
        model (str): the type of model to use from IBM Watsonx AI.
        credentials ([dict]): credentials to Watson Machine Learning instance.
        project_id (str): ID of the Watson Studio project.
        **kwargs: Additional arguments to pass to the API provider. This is initialized with default valu
            text generation parameters needed for communicating with Watsonx API, such as:
                - decoding_method
                - max_new_tokens
                - min_new_tokens
                - stop_sequences
                - repetition_penalty
    """

    def __init__(self, model, credentials, project_id, **kwargs):
        """Parameters

        model : str
            Which pre-trained model from Watsonx.ai to use?
            Choices are [
                `mistralai/mixtral-8x7b-instruct-v01`,
                `ibm/granite-13b-instruct-v2`,
                `meta-llama/llama-3-70b-instruct`]
        credentials : [dict]
            Credentials to Watson Machine Learning instance.
        project_id : str
            ID of the Watson Studio project.
        **kwargs: dict
            Additional arguments to pass to the API provider.
```

```
import logging
import os
from typing import Any, Optional

import backoff
import requests
from pydantic import BaseModel, ValidationError

from dsp.modules.lm import LM


def backoff_hdlr(details) -> None:
    """Log backoff details when retries occur."""
    logging.warning(
        f"Backing off {details['wait']:0.1f} seconds afters {details['tries']} tries "
        f"calling function {details['target']} with args {details['args']} and kwargs {details['kwargs']}",
    )


def giveup_hdlr(details) -> bool:
    """Decide whether to give up on retries based on the exception."""
    logging.error(
        "Giving up: After {tries} tries, calling {target} failed due to {value}".format(
            tries=details["tries"],
            target=details["target"],
            value=details.get("value", "Unknown Error"),
        ),
    )
    return False  # Always returns False to not give up


class LLMResponse(BaseModel):
    response: str


class CloudflareAIResponse(BaseModel):
    result: LLMResponse
    success: bool
    errors: list
    messages: list


class CloudflareAI(LM):
    """Wrapper around Cloudflare Workers AI API."""

    def __init__(
        self,
```

```
import functools
import json
import logging
from typing import Any, Literal, Optional, cast

import backoff
import openai

from dsp.modules.cache_utils import CacheMemory, NotebookCacheMemory, cache_turn_on
from dsp.modules.lm import LM

try:
    OPENAI_LEGACY = int(openai.version.__version__[0]) == 0
except Exception:
    OPENAI_LEGACY = True

try:
    import openai.error
    from openai.openai_object import OpenAIObject

    ERRORS = (openai.error.RateLimitError,)
except Exception:
    ERRORS = (openai.RateLimitError,)
    OpenAIObject = dict


def backoff_hdlr(details):
    """Handler from https://pypi.org/project/backoff/"""
    print(
        "Backing off {wait:0.1f} seconds after {tries} tries "
        "calling function {target} with kwargs "
        "{kwargs}".format(**details),
    )


class GPT3(LM):
    """Wrapper around OpenAI's GPT API.

    Args:
        model (str, optional): OpenAI supported LLM model to use. Defaults to "gpt-3.5-turbo-instruct"
        api_key (Optional[str], optional): API provider Authentication token. use Defaults to None.
        api_provider (Literal["openai"], optional): The API provider to use. Defaults to "openai".
        model_type (Literal["chat", "text"], optional): The type of model that was specified. Mainly to de
        **kwargs: Additional arguments to pass to the API provider.
    """

    def __init__(
```

```
import json
from typing import Union

from datasets import Dataset

from dsp.utils import dotdict


class PyseriniRetriever:
    """Wrapper for retrieval with Pyserini. Supports using either pyserini prebuilt faiss indexes or you

    def __init__(self,
                 query_encoder: str = 'castorini/dkrr-dpr-nq-retriever',
                 index: str = 'wikipedia-dpr-dkrr-nq',
                 dataset: Dataset = None,
                 id_field: str = '_id',
                 text_fields: list[str] = ['text']) -> None:
        """
        Args:

            query_encoder (`str`):
                Huggingface model to encode queries
            index (`str`):
                Either a prebuilt index from pyserini or a local path to a faiss index
            dataset (`Dataset`):
                Only required when using a local faiss index. The dataset should be the one that has bee
            id_field (`str`):
                The name of the id field of the dataset used for retrieval.
            text_fields (`list[str]`):
                A list of the names of the text fields for the dataset used for retrieval.
        """

        # Keep pyserini as an optional dependency
        from pyserini.prebuilt_index_info import FAISS_INDEX_INFO, IMPACT_INDEX_INFO, TF_IN
        from pyserini.search import FaissSearcher

        self.encoder = FaissSearcher._init_encoder_from_str(query_encoder)
        self.dataset = dataset
        self.id_field = id_field
        self.text_fields = text_fields

        if index in TF_INDEX_INFO or index in FAISS_INDEX_INFO or index in IMPACT_INDEX_INF
            self.searcher = FaissSearcher.from_prebuilt_index(index, self.encoder)
        else:
            self.searcher = FaissSearcher(index_dir=index, query_encoder=self.encoder)
            assert self.dataset is not None
            self.dataset_id_to_index = {}
```

```
# # To Run:
# # python -m dsp.modules.hf_server --port 4242 --model "google/flan-t5-base"

# # To Query:
# # curl -d '{"prompt":".."}' -X POST "http://0.0.0.0:4242" -H 'Content-Type: application/json'
# # Or use the HF client. TODO: Add support for kwargs to the server.


# from functools import lru_cache
# import argparse
# import time
# import random
# import os
# import sys
# import uvicorn
# import warnings

# from fastapi import FastAPI
# from pydantic import BaseModel
# from argparse import ArgumentParser
# from starlette.middleware.cors import CORSMiddleware

# from dsp.modules.hf import HFModel


# class Query(BaseModel):
#     prompt: str
#     kwargs: dict = {}


# warnings.filterwarnings("ignore")

# app = FastAPI()
# app.add_middleware(
#     CORSMiddleware, allow_origins=["*"], allow_methods=["*"], allow_headers=["*"]
# )

# parser = argparse.ArgumentParser("Server for Hugging Face models")
# parser.add_argument("--port", type=int, required=True, help="Server port")
# parser.add_argument("--model", type=str, required=True, help="Hugging Face model")
# args = parser.parse_args()
# # TODO: Convert this to a log message
# print(f"#> Loading the language model {args.model}")
# lm = HFModel(args.model)


# @lru_cache(maxsize=None)
```

```
from .aggregation import majority
from .chain_of_thought import ChainOfThought
from .chain_of_thought_with_hint import ChainOfThoughtWithHint
from .knn import KNN
from .multi_chain_comparison import MultiChainComparison
from .predict import Predict
from .program_of_thought import ProgramOfThought
from .react import ReAct
from .retry import Retry
```

```
# Adapted from: https://www.philschmid.de/fine-tune-flan-t5#3-fine-tune-and-evaluate-flan-t5

import copy
import glob
import json
import os
import warnings
from dataclasses import dataclass

import evaluate
import numpy as np
import torch
from datasets import Dataset
from transformers import (
    AutoConfig,
    AutoModelForCausalLM,
    AutoModelForSeq2SeqLM,
    AutoTokenizer,
    DataCollatorForSeq2Seq,
    PreTrainedTokenizer,
    Seq2SeqTrainer,
    Seq2SeqTrainingArguments,
    Trainer,
    TrainingArguments,
    set_seed,
)

# from peft import get_peft_model, LoraConfig, TaskType
from transformers.trainer_callback import TrainerCallback

# from dsp.modules.finetuning.fid import *


warnings.filterwarnings("ignore")

IGNORE_INDEX = -100
DEFAULT_SEP_TOKEN = "[SEP]"
DEFAULT_PAD_TOKEN = "[PAD]"
DEFAULT_EOS_TOKEN = "</s>"
DEFAULT_BOS_TOKEN = "<s>"
DEFAULT_UNK_TOKEN = "</s>"
SPECIAL_TOKENS_DICT = {
    "sep_token": DEFAULT_SEP_TOKEN,
    "pad_token": DEFAULT_PAD_TOKEN,
    # "eos_token": DEFAULT_EOS_TOKEN,
    # "bos_token": DEFAULT_BOS_TOKEN,
    "unk_token": DEFAULT_UNK_TOKEN,
```

```
import inspect
import logging
import math
import os
import random
import shutil
import sys

import numpy as np

try:
    from IPython.core.magics.code import extract_symbols
except ImportError:
    # Won't be able to read code from juptyer notebooks
    extract_symbols = None

import dspy
from dspy.predict.parameter import Parameter
from dspy.teleprompt.bootstrap import BootstrapFewShot, LabeledFewShot

"""
This file consists of helper functions for our variety of optimizers.
"""

### OPTIMIZER TRAINING UTILS ###


def create_minibatch(trainset, batch_size=50):
    """Create a minibatch from the trainset."""

    # Ensure batch_size isn't larger than the size of the dataset
    batch_size = min(batch_size, len(trainset))

    # Randomly sample indices for the mini-batch
    sampled_indices = random.sample(range(len(trainset)), batch_size)

    # Create the mini-batch using the sampled indices
    minibatch = [trainset[i] for i in sampled_indices]

    return minibatch


def eval_candidate_program(batch_size, trainset, candidate_program, evaluate):
    """Evaluate a candidate program on the trainset, using the specified batch size."""
    # Evaluate on the full trainset
    if batch_size >= len(trainset):
        score = evaluate(candidate_program, devset=trainset, display_table=0)
```

```
from .aggregation import majority
from .chain_of_thought import ChainOfThought
from .chain_of_thought_with_hint import ChainOfThoughtWithHint
from .knn import KNN
from .multi_chain_comparison import MultiChainComparison
from .predict import Predict
from .program_of_thought import ProgramOfThought
from .react import ReAct
from .retry import Retry
```

```
import random

from pydantic import BaseModel

import dsp
from dspy.predict.parameter import Parameter
from dspy.primitives.prediction import Prediction
from dspy.signatures.signature import ensure_signature, signature_to_template


class Predict(Parameter):
    def __init__(self, signature, **config):
        self.stage = random.randbytes(8).hex()
        self.signature = ensure_signature(signature)
        self.config = config
        self.reset()

    def reset(self):
        self.lm = None
        self.traces = []
        self.train = []
        self.demos = []

    def dump_state(self):
        state_keys = ["lm", "traces", "train"]
        state = {k: getattr(self, k) for k in state_keys}

        state["demos"] = []
        for demo in self.demos:
            demo = demo.copy()

            for field in demo:
                if isinstance(demo[field], BaseModel):
                    demo[field] = demo[field].model_dump_json()

            state["demos"].append(demo)

        # Cache the signature instructions and the last field's name.
        state["signature_instructions"] = self.signature.instructions

        *_, last_key = self.signature.fields.keys()
        state["signature_prefix"] = self.signature.fields[last_key].json_schema_extra["prefix"]

        return state

    def load_state(self, state):
        for name, value in state.items():
```

```
import random
from typing import Any, Callable

import numpy as np

import dsp
from dsp.utils import EM, F1, DPR_normalize, dotdict, has_answer, normalize_text


class Example(dotdict):
    """A primitive datatype for representing an example"""

    demos: list[Any]

    def __init__(self, *args, **kwargs):
        assert len(args) <= 1
        super().__init__()

        if args:
            assert len(args) == 1
            self.update(args[0])

        self.update(**kwargs)

    def copy(self, **kwargs):
        the_copy = Example(**{**dict(self), **kwargs})

        return the_copy

    def without(self, *keys):
        """Removes the provided keys from the example and returns a copy"""
        keys = set(keys)
        return Example({k: v for k, v in self.items() if k not in keys})

    def demos_at(self, fn):
        """Returns a copy of the example with the demos stage transformed by the provided function"

        def at(example):
            try:
                return fn(example).without("augmented")
            except Exception:
                return {}

        demos = [example.copy(**at(example)) for example in self.demos]
        return self.copy(demos=demos)
```

```
import os
import random
import subprocess
import time

import tqdm
import ujson
from datasets.fingerprint import Hasher

import dsp

if os.environ.get('DSP_NOTEBOOK_CACHEDIR'):
    training_data_directory = os.path.join(os.environ.get('DSP_NOTEBOOK_CACHEDIR'), 'compiler
else:
    training_data_directory = 'cache/compiler'


compilations_assumed_to_exist={'ft-zvEdzQVQ5xwlxvNPrxl6kpnw': 'ada:ft-stanfordpraglab-2023-0


def openai_check_finetune(jobname):
    if dsp.settings.force_reuse_cached_compilation and jobname in compilations_assumed_to_exist
        return compilations_assumed_to_exist[jobname]

    command = f"""openai api fine_tunes.get -i {jobname}"""
    print(command)

    result = subprocess.run(command.split(), stdout=subprocess.PIPE, check=False)
    output = result.stdout.decode("utf-8").strip()

    try:
        output = ujson.loads(output)
        if output['status'] == 'succeeded':
            return output['fine_tuned_model']

        if output['status'] in ['pending', 'running']:
            print(f'Compiling, run ```openai api fine_tunes.follow -i {jobname}``` for details...')
            time.sleep(60)
            return openai_check_finetune(jobname)
    except:
        pass

    return False


def convert_to_training_point2(y, inputs, outputs, template):
    assert len(inputs) + len(outputs) == len(template.fields)
```

```
from functools import wraps

import dsp


# applied right to left (innermost first, like function calls)
def compose_decorators(*decorators):
    def decorator(func):
        for decorator in decorators[::-1]:
            func = decorator(func)
        return func
    return decorator


def shallow_copy_example_args(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        args = [dsp.Example(arg) if isinstance(arg, dsp.Example) else arg for arg in args]
        kwargs = {key: dsp.Example(value) if isinstance(value, dsp.Example) else value for key, value
        return func(*args, **kwargs)
    return wrapper


transformation = shallow_copy_example_args
# transformation = compose_decorators(handle_compilation, shallow_copy_example_args)



def compiled(func):
    def wrapper(*args, **kwargs):
        is_to_be_compiled = True #decorator_kwargs.get('compile', False)
        compiled_lm = dsp.settings.compiled_lm

        if is_to_be_compiled and compiled_lm:
            assert len(args) == 1, len(args)
            example = args[0]

            with dsp.settings.context(lm=compiled_lm, show_guidelines=False):
                old_demos = list(example.demos)
                example = func(example.copy(demos=[]), **kwargs)
                return example.copy(demos=old_demos)

        with dsp.settings.context(compiling=True):
            return func(*args, **kwargs)

    return wrapper
```

```
from .aggregation import majority
from .chain_of_thought import ChainOfThought
from .chain_of_thought_with_hint import ChainOfThoughtWithHint
from .knn import KNN
from .multi_chain_comparison import MultiChainComparison
from .predict import Predict
from .program_of_thought import ProgramOfThought
from .react import ReAct
from .retry import Retry
```

```
import inspect
import json
import random
import string

import requests


class FuncInspector:
  def __init__(self):
    self.calls = []


  def inspect_inner(self, func, function_calls):
    def wrapper(*args, **kwargs):
      result = func(*args, **kwargs)
      self.merge_result(result, function_calls)
      return result
    return wrapper


  def inspect_func(self, func):
    def wrapper(*args, **kwargs):
      result = func(*args, **kwargs)
      stack = inspect.stack()
      function_calls = []
      for i in range(len(stack)):
        if stack[i][3] == "<module>":
          break
        if stack[i][3] != "wrapper":
          function_calls.append(stack[i][3])
      function_calls.reverse()
      result = self.inspect_inner(result, function_calls)
      return result
    return wrapper


  def parse(self, obj, delete_empty=False):
    if isinstance(obj, list):
      for elem in obj:
        self.parse(elem, delete_empty)
    if isinstance(obj, dict):
      to_delete = []
      for key in obj:
        if delete_empty and not obj[key] or key == "completions":
          to_delete.append(key)
        else:
```

```
import logging
from collections.abc import Iterable

import numpy as np

import dsp

logger = logging.getLogger(__name__)

def retrieve(query: str, k: int, **kwargs) -> list[str]:
    """Retrieves passages from the RM for the query and returns the top k passages."""
    if not dsp.settings.rm:
        raise AssertionError("No RM is loaded.")
    passages = dsp.settings.rm(query, k=k, **kwargs)
    if not isinstance(passages, Iterable):
        # it's not an iterable yet; make it one.
        # TODO: we should unify the type signatures of dspy.Retriever
        passages = [passages]
    passages = [psg.long_text for psg in passages]

    if dsp.settings.reranker:
        passages_cs_scores = dsp.settings.reranker(query, passages)
        passages_cs_scores_sorted = np.argsort(passages_cs_scores)[::-1]
        passages = [passages[idx] for idx in passages_cs_scores_sorted]



    return passages
def retrievewithMetadata(query: str, k: int, **kwargs) -> list[str]:
    """Retrieves passages from the RM for the query and returns the top k passages."""

    if not dsp.settings.rm:
        raise AssertionError("No RM is loaded.")
    passages = dsp.settings.rm(query, k=k, **kwargs)
    if not isinstance(passages, Iterable):
        # it's not an iterable yet; make it one.
        # TODO: we should unify the type signatures of dspy.Retriever
        passages = [passages]

    return passages



def retrieveRerankEnsemble(queries: list[str], k: int,**kwargs) -> list[str]:
    if not (dsp.settings.rm and dsp.settings.reranker):
        raise AssertionError("Both RM and Reranker are needed to retrieve & re-rank.")
    queries = [q for q in queries if q]
    passages = {}
    for query in queries:
```

```
import inspect
import logging
import math
import os
import random
import shutil
import sys

import numpy as np

try:
    from IPython.core.magics.code import extract_symbols
except ImportError:
    # Won't be able to read code from juptyer notebooks
    extract_symbols = None

import dspy
from dspy.predict.parameter import Parameter
from dspy.teleprompt.bootstrap import BootstrapFewShot, LabeledFewShot

"""
This file consists of helper functions for our variety of optimizers.
"""

### OPTIMIZER TRAINING UTILS ###


def create_minibatch(trainset, batch_size=50):
    """Create a minibatch from the trainset."""

    # Ensure batch_size isn't larger than the size of the dataset
    batch_size = min(batch_size, len(trainset))

    # Randomly sample indices for the mini-batch
    sampled_indices = random.sample(range(len(trainset)), batch_size)

    # Create the mini-batch using the sampled indices
    minibatch = [trainset[i] for i in sampled_indices]

    return minibatch


def eval_candidate_program(batch_size, trainset, candidate_program, evaluate):
    """Evaluate a candidate program on the trainset, using the specified batch size."""
    # Evaluate on the full trainset
    if batch_size >= len(trainset):
        score = evaluate(candidate_program, devset=trainset, display_table=0)
```

```
from .aggregation import majority
from .chain_of_thought import ChainOfThought
from .chain_of_thought_with_hint import ChainOfThoughtWithHint
from .knn import KNN
from .multi_chain_comparison import MultiChainComparison
from .predict import Predict
from .program_of_thought import ProgramOfThought
from .react import ReAct
from .retry import Retry
```

```
import copy
import threading
from concurrent.futures import ThreadPoolExecutor, as_completed
from contextlib import contextmanager


class Settings:
    def __init__(self):
        # A lock for ensuring thread-safety when accessing _parent_configs
        self._lock = threading.Lock()

        # Dictionary to hold parent thread configurations
        self._parent_configs = {}

        # Using thread-local storage to ensure that each thread has its own configuration stack
        self._local = threading.local()

    def _get_current_config(self):
        return self._local.config_stack[-1] if hasattr(self._local, 'config_stack') and self._local.config_st

    def initialize_for_thread(self, parent_tid):
        """Initialize thread-local data for a new thread using its parent's config."""
        with self._lock:
            parent_config = self._parent_configs.get(parent_tid)
            if parent_config:
                self._local.config_stack = [copy.deepcopy(parent_config)]
            else:
                self._local.config_stack = [{}]

    @contextmanager
    def context(self, **kwargs):
        current_config = copy.deepcopy(self._get_current_config())  # Deep copy the current configura
        current_config.update(kwargs)

        if not hasattr(self._local, 'config_stack'):
            self._local.config_stack = []

        self._local.config_stack.append(current_config)

        # Register the modified config as the potential parent config
        with self._lock:
            self._parent_configs[threading.get_ident()] = copy.deepcopy(current_config)  # Deep copy t

        try:
            yield
        finally:
            self._local.config_stack.pop()
```

```
import threading
from contextlib import contextmanager

from dsp.utils.utils import dotdict


class Settings:
    """DSP configuration settings."""

    _instance = None

    def __new__(cls):
        """
        Singleton Pattern. See https://python-patterns.guide/gang-of-four/singleton/
        """

        if cls._instance is None:
            cls._instance = super().__new__(cls)
            cls._instance.lock = threading.Lock()
            cls._instance.main_tid = threading.get_ident()
            cls._instance.main_stack = []
            cls._instance.stack_by_thread = {}
            cls._instance.stack_by_thread[threading.get_ident()] = cls._instance.main_stack

            #  TODO: remove first-class support for re-ranker and potentially combine with RM to form a
            #  eg: RetrieveThenRerankPipeline(RetrievalModel, Reranker)
            #  downstream operations like dsp.retrieve would use configs from the defined pipeline.
            config = dotdict(
                lm=None,
                rm=None,
                branch_idx=0,
                reranker=None,
                compiled_lm=None,
                force_reuse_cached_compilation=False,
                compiling=False,
                skip_logprobs=False,
                trace=[],
                release=0,
                bypass_assert=False,
                bypass_suggest=False,
                assert_failures=0,
                suggest_failures=0,
                langchain_history=[],
            )
            cls._instance.__append(config)

        return cls._instance
```

```
from typing import Tuple

try:
    import faiss
    from faiss import Index
except ImportError:
    raise ImportError(
        "You need to install FAISS library to perform ANN/KNN. Please check the official doc: "
        "https://github.com/facebookresearch/faiss/blob/main/INSTALL.md",
    )


def determine_devices(max_gpu_devices: int = 0) -> Tuple[int, bool]:
    """
    Determine which device we should use
    Args:
        max_gpu_devices: an integer value, define how many GPUs we'll use.
            -1 means all devices. 0 means there are no GPUs. Default is 0.

    Returns: number of devices and is it allowed to use CUDA device (True if yes)
    """
    n_devices_total = faiss.get_num_gpus()
    is_gpu = n_devices_total > 0

    if max_gpu_devices > 0 and is_gpu:
        num_devices = min(n_devices_total, max_gpu_devices)
    elif max_gpu_devices == -1 and is_gpu:
        num_devices = n_devices_total
    else:
        num_devices = 1
        is_gpu = False
    return num_devices, is_gpu


def _get_brute_index(emb_dim: int, dist_type: str) -> Index:
    if dist_type.lower() == 'ip':
        index = faiss.IndexFlatIP(emb_dim)
    elif dist_type.lower() == 'l2':
        index = faiss.IndexFlatL2(emb_dim)
    else:
        raise ValueError(f'Wrong distance type for FAISS Flat Index: {dist_type}')

    return index


def _get_ivf_index(
    emb_dim: int,
```

```
"""
    Source: DPR Implementation from Facebook Research
    https://github.com/facebookresearch/DPR/tree/master/dpr
    Original license: https://github.com/facebookresearch/DPR/blob/main/LICENSE
"""

import unicodedata

import regex


class Tokens:
    """A class to represent a list of tokenized text."""
    TEXT = 0
    TEXT_WS = 1
    SPAN = 2
    POS = 3
    LEMMA = 4
    NER = 5

    def __init__(self, data, annotators, opts=None):
        self.data = data
        self.annotators = annotators
        self.opts = opts or {}

    def __len__(self):
        """The number of tokens."""
        return len(self.data)

    def slice(self, i=None, j=None):
        """Return a view of the list of tokens from [i, j)."""
        new_tokens = copy.copy(self)
        new_tokens.data = self.data[i: j]
        return new_tokens

    def untokenize(self):
        """Returns the original text (with whitespace reinserted)."""
        return ''.join([t[self.TEXT_WS] for t in self.data]).strip()

    def words(self, uncased=False):
        """Returns a list of the text of each token

        Args:
            uncased: lower cases text
        """
        if uncased:
            return [t[self.TEXT].lower() for t in self.data]
```

```
# TODO: This should move internally. Same for passage_match. dspy.metrics.answer_exact_matc

import dsp


def answer_exact_match(example, pred, trace=None, frac=1.0):
    assert(type(example.answer) is str or type(example.answer) is list)

    if type(example.answer) is str:
        return dsp.answer_match(pred.answer, [example.answer], frac=frac)
    else: # type(example.answer) is list
        return dsp.answer_match(pred.answer, example.answer, frac=frac)

answer_exact_match_str = dsp.answer_match

def answer_passage_match(example, pred, trace=None):
    assert(type(example.answer) is str or type(example.answer) is list)

    if type(example.answer) is str:
        return dsp.passage_match(pred.context, [example.answer])
    else: # type(example.answer) is list
        return dsp.passage_match(pred.context, example.answer)
```

```
{
  "retrieve": {
    "k": 3
  },
  "generate_query[0]": {
    "lm": null,
    "traces": [],
    "train": [],
    "demos": [
      {
        "augmented": true,
        "dspy_uuid": "c126c397-254f-455f-bd22-1df7228cf1bc",
        "context": [],
        "question": "Which of these publications was most recently published, Who Put the Bomp or S
        "rationale": "determine the most recent publication. We know that Who Put the Bomp was pub
        "search_query": "Self"
      },
      {
        "augmented": true,
        "dspy_uuid": "7c56ed8c-85b9-4b4f-be70-7eb3a20539ec",
        "context": [],
        "question": "The Victorians - Their Story In Pictures is a documentary series written by an auth
        "rationale": "find the correct search query. We know that the author was born in the 19th centu
        "search_query": "\"The Victorians - Their Story In Pictures\" author:${author_name}"
      },
      {
        "question": "What Scottish nobleman was the subject of a 1934 British short documentary and
        "answer": "Douglas Douglas-Hamilton, 14th Duke of Hamilton",
        "dspy_uuid": "66c1198b-2f31-4f1c-a1f8-d10dcfaa0a66",
        "dspy_split": "train"
      },
      {
        "question": "Whats was the population in 2003 of Centralia, the least populated municipality in
        "answer": "7",
        "dspy_uuid": "92183f94-9c08-4915-a666-dbcc22aeb633",
        "dspy_split": "train"
      },
      {
        "question": "What Brazilian professional racing driver who races for Rebellion Racing has a m
        "answer": "Bruno Senna Lalli",
        "dspy_uuid": "e4e1c569-f4d6-4b5a-9b31-d4ce129395ac",
        "dspy_split": "train"
      },
      {
        "question": "What head of state position was held by Harry S Truman when he gave Harold E
        "answer": "President of the United States",
        "dspy_uuid": "9465e8d9-277b-43c6-81f5-b3349923045c",
```

```
{
  "retrieve": {
    "k": 3
  },
  "generate_query[0]": {
    "lm": null,
    "traces": [],
    "train": [],
    "demos": [
      {
        "augmented": true,
        "dspy_uuid": "a8e15619-7c04-467a-8580-4dcf10f48bed",
        "context": [],
        "question": "Who was the Tennis Masters Cup champion in 2000, Gustavo Kuerten or Stan W
        "rationale": "find the answer. We know that Gustavo Kuerten won the tournament in 2000, so v
        "search_query": "Gustavo Kuerten"
      },
      {
        "augmented": true,
        "dspy_uuid": "ef6efe73-7815-4e31-a72d-d4a1537c5380",
        "context": [],
        "question": "In which year was the first of the vampire-themed fantasy romance novels for whi
        "rationale": "find the answer to this question. We know that the first book in The Twilight Saga
        "search_query": "\"vampire-themed fantasy romance novels 2005\""
      },
      {
        "question": "Do Stu Block and Johnny Bonnel's bands play the same type of music?",
        "answer": "no",
        "dspy_uuid": "7466462a-c7b5-4fee-a867-9324d39125e9",
        "dspy_split": "train"
      },
      {
        "question": "Some performance of Take a Bow took place near the bank of which river?",
        "answer": "River Thames",
        "dspy_uuid": "937a5676-7522-4e67-8e1e-d0b5651fad73",
        "dspy_split": "train"
      },
      {
        "question": "What Brazilian professional racing driver who races for Rebellion Racing has a m
        "answer": "Bruno Senna Lalli",
        "dspy_uuid": "e4e1c569-f4d6-4b5a-9b31-d4ce129395ac",
        "dspy_split": "train"
      },
      {
        "question": "The Victorians - Their Story In Pictures is a documentary series written by an auth
        "answer": "1950",
        "dspy_uuid": "7c56ed8c-85b9-4b4f-be70-7eb3a20539ec",
```

```
{
  "retrieve": {
    "k": 3
  },
  "generate_query[0]": {
    "lm": null,
    "traces": [],
    "train": [],
    "demos": [
      {
        "augmented": true,
        "dspy_uuid": "ef6efe73-7815-4e31-a72d-d4a1537c5380",
        "context": [],
        "question": "In which year was the first of the vampire-themed fantasy romance novels for whi
        "rationale": "find the answer to this question. We know that the first book in The Twilight Saga
        "search_query": "\"vampire-themed fantasy romance novels 2005\""
      },
      {
        "augmented": true,
        "dspy_uuid": "7c56ed8c-85b9-4b4f-be70-7eb3a20539ec",
        "context": [],
        "question": "The Victorians - Their Story In Pictures is a documentary series written by an auth
        "rationale": "find the correct search query. We know that the author was born in the 19th centu
        "search_query": "\"The Victorians - Their Story In Pictures\" author:${author_name}"
      },
      {
        "question": "What is the code name for the German offensive that started this Second World W
        "answer": "Operation Citadel",
        "dspy_uuid": "d7fb5c24-6ca2-4bf8-b9f1-0a6e0060ebea",
        "dspy_split": "train"
      },
      {
        "question": "What type of film was it that featured the actress who is regarded as the greatest
        "answer": "anti-McCarthyism",
        "dspy_uuid": "96c92062-7e1c-4712-b3c8-30f35f8b6c65",
        "dspy_split": "train"
      },
      {
        "question": "Tombstone stared an actor born May 17, 1955 known as who?",
        "answer": "Bill Paxton",
        "dspy_uuid": "86fca9c5-3c75-4aae-bc16-cb6c62335f22",
        "dspy_split": "train"
      },
      {
        "question": "Who was coach of the No. 9-ranked team that was upset in the NCAA Tournamen
        "answer": "Fred Hoiberg",
        "dspy_uuid": "089c6c36-6317-4dd1-8de5-5cd72185f0ab",
```

```
{
  "retrieve": {
    "k": 3
  },
  "generate_query[0]": {
    "lm": null,
    "traces": [],
    "train": [],
    "demos": [
      {
        "augmented": true,
        "dspy_uuid": "7466462a-c7b5-4fee-a867-9324d39125e9",
        "context": [],
        "question": "Do Stu Block and Johnny Bonnel's bands play the same type of music?",
        "rationale": "determine if Stu Block and Johnny Bonnel's bands play the same type of music. V
        "search_query": "\"Iced Earth\" OR \"Bonelord\""
      },
      {
        "augmented": true,
        "dspy_uuid": "0610d91f-d32e-4bcd-b0a7-d03fbe9538e3",
        "context": [],
        "question": "Milan Sachs conducted the premiere of the play Jen\u016ffa by the composer wh
        "rationale": "find the search query. We know that Milan Sachs conducted the premiere of the p
        "search_query": "Jen\u016ffa folk music"
      },
      {
        "question": "Who composed \"Sunflower Slow Drag\" with the King of Ragtime?",
        "answer": "Scott Hayden",
        "dspy_uuid": "4ab462bd-4d9b-4b7f-9371-14ba20ccd0db",
        "dspy_split": "train"
      },
      {
        "question": "The Organisation that allows a community to influence their operation or use and
        "answer": "2010",
        "dspy_uuid": "821a4e4d-a146-44a7-bdaa-48b7efb4e7cc",
        "dspy_split": "train"
      },
      {
        "question": "Who acted in the shot film The Shore and is also the youngest actress ever to pla
        "answer": "Kerry Condon",
        "dspy_uuid": "1e87ebea-f541-4629-b995-498965fa1127",
        "dspy_split": "train"
      },
      {
        "question": "Samantha Cristoforetti and Mark Shuttleworth are both best known for being first i
        "answer": "space",
        "dspy_uuid": "47b1cbeb-dba8-4cab-a50c-97b81fd86eb3",
```

```
import pytest
from dsp.utils import deduplicate
import dspy.evaluate
import dspy
from dspy.datasets import HotPotQA
from dspy.evaluate.evaluate import Evaluate
from dspy.teleprompt.bootstrap import BootstrapFewShot


class GenerateAnswer(dspy.Signature):
    """Answer questions with short factoid answers."""

    context = dspy.InputField(desc="may contain relevant facts")
    question = dspy.InputField()
    answer = dspy.OutputField(desc="often between 1 and 5 words")


class GenerateSearchQuery(dspy.Signature):
    """Write a simple search query that will help answer a complex question."""

    context = dspy.InputField(desc="may contain relevant facts")
    question = dspy.InputField()
    query = dspy.OutputField()


class SimplifiedBaleen(dspy.Module):
    def __init__(self, passages_per_hop=3, max_hops=2):
        super().__init__()

        self.generate_query = [
            dspy.ChainOfThought(GenerateSearchQuery) for _ in range(max_hops)
        ]
        self.retrieve = dspy.Retrieve(k=passages_per_hop)
        self.generate_answer = dspy.ChainOfThought(GenerateAnswer)
        self.max_hops = max_hops

    def forward(self, question):
        context = []

        for hop in range(self.max_hops):
            query = self.generate_query[hop](context=context, question=question).query
            passages = self.retrieve(query).passages
            context = deduplicate(context + passages)

        pred = self.generate_answer(context=context, question=question)
        return dspy.Prediction(context=context, answer=pred.answer)
```

```
import textwrap
from typing import List

import pydantic
import pytest

import dspy
from dspy import InputField, OutputField, Signature, infer_prefix
from dspy.utils.dummies import DummyLM


def test_field_types_and_custom_attributes():
    class TestSignature(Signature):
        """Instructions"""

        input1: str = InputField()
        input2: int = InputField()
        output1: List[str] = OutputField()
        output2 = OutputField()

    assert TestSignature.instructions == "Instructions"
    assert TestSignature.input_fields["input1"].annotation == str
    assert TestSignature.input_fields["input2"].annotation == int
    assert TestSignature.output_fields["output1"].annotation == List[str]
    assert TestSignature.output_fields["output2"].annotation == str


def test_no_input_output():
    with pytest.raises(TypeError):

        class TestSignature(Signature):
            input1: str


def test_no_input_output2():
    with pytest.raises(TypeError):

        class TestSignature(Signature):
            input1: str = pydantic.Field()


def test_all_fields_have_prefix():
    class TestSignature(Signature):
        input = InputField(prefix="Modified:")
        output = OutputField()

    assert TestSignature.input_fields["input"].json_schema_extra["prefix"] == "Modified:"
```

```
"""Tests for AWS models.
Note: Requires configuration of your AWS credentials with the AWS CLI and creating sagemaker e
TODO: Create mock fixtures for pytest to remove the need for AWS credentials and endpoints.
"""

import dsp
import dspy


def get_lm(lm_provider: str, model_path: str, **kwargs) -> dsp.modules.lm.LM:
    """get the language model"""
    # extract model vendor and name from model name
    # Model path format is <MODEL_VENDOR>/<MODEL_NAME_OR_ENDPOINT>
    model_vendor = model_path.split("/")[0]
    model_name = model_path.split("/")[1]

    if lm_provider == "Bedrock":
        bedrock = dspy.Bedrock(region_name="us-west-2")
        if model_vendor == "mistral":
            return dspy.AWSMistral(bedrock, model_name, **kwargs)
        elif model_vendor == "anthropic":
            return dspy.AWSAnthropic(bedrock, model_name, **kwargs)
        elif model_vendor == "meta":
            return dspy.AWSMeta(bedrock, model_name, **kwargs)
        else:
            raise ValueError(
                "Model vendor missing or unsupported: Model path format is <MODEL_VENDOR>/<MO
            )
    elif lm_provider == "Sagemaker":
        sagemaker = dspy.Sagemaker(region_name="us-west-2")
        if model_vendor == "mistral":
            return dspy.AWSMistral(sagemaker, model_name, **kwargs)
        elif model_vendor == "meta":
            return dspy.AWSMeta(sagemaker, model_name, **kwargs)
        else:
            raise ValueError(
                "Model vendor missing or unsupported: Model path format is <MODEL_VENDOR>/<MO
            )
    else:
        raise ValueError(f"Unsupported model: {model_name}")


def run_tests():
    """Test the providers and models"""
    # Configure your AWS credentials with the AWS CLI before running this script
    provider_model_tuples = [
        ("Bedrock", "mistral/mistral.mixtral-8x7b-instruct-v0:1"),
```

```
"""Tests for Cloudflare models.
Note: Requires configuration of your Cloudflare account_id and api_key.
"""

import dspy

models = {
    "@cf/qwen/qwen1.5-0.5b-chat": "https://huggingface.co/qwen/qwen1.5-0.5b-chat",
    "@hf/meta-llama/meta-llama-3-8b-instruct": "https://llama.meta.com",
    "@hf/nexusflow/starling-lm-7b-beta": "https://huggingface.co/Nexusflow/Starling-LM-7B-beta",
    "@cf/meta/llama-3-8b-instruct": "https://llama.meta.com",
    "@hf/thebloke/neural-chat-7b-v3-1-awq": "",
    "@cf/meta/llama-2-7b-chat-fp16": "https://ai.meta.com/llama/",
    "@cf/mistral/mistral-7b-instruct-v0.1": "https://mistral.ai/news/announcing-mistral-7b/",
    "@cf/tinyllama/tinyllama-1.1b-chat-v1.0": "https://huggingface.co/TinyLlama/TinyLlama-1.1B-Cha
    "@hf/mistral/mistral-7b-instruct-v0.2": "https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2",
    "@cf/fblgit/una-cybertron-7b-v2-bf16": "",
    "@hf/thebloke/codellama-7b-instruct-awq": "https://huggingface.co/TheBloke/CodeLlama-7B-Inst
    "@cf/thebloke/discolm-german-7b-v1-awq": "https://huggingface.co/TheBloke/DiscoLM_German
    "@cf/meta/llama-2-7b-chat-int8": "https://ai.meta.com/llama/",
    "@hf/thebloke/mistral-7b-instruct-v0.1-awq": "https://huggingface.co/TheBloke/Mistral-7B-Instruc
    "@hf/thebloke/openchat_3.5-awq": "",
    "@cf/qwen/qwen1.5-7b-chat-awq": "https://huggingface.co/qwen/qwen1.5-7b-chat-awq",
    "@hf/thebloke/llama-2-13b-chat-awq": "https://huggingface.co/TheBloke/Llama-2-13B-chat-AWQ
    "@hf/thebloke/deepseek-coder-6.7b-base-awq": "",
    "@hf/thebloke/openhermes-2.5-mistral-7b-awq": "",
    "@hf/thebloke/deepseek-coder-6.7b-instruct-awq": "",
    "@cf/deepseek-ai/deepseek-math-7b-instruct": "https://huggingface.co/deepseek-ai/deepseek-m
    "@cf/tiiuae/falcon-7b-instruct": "https://huggingface.co/tiiuae/falcon-7b-instruct",
    "@hf/nousresearch/hermes-2-pro-mistral-7b": "https://huggingface.co/NousResearch/Hermes-2-
    "@hf/thebloke/zephyr-7b-beta-awq": "https://huggingface.co/TheBloke/zephyr-7B-beta-AWQ",
    "@cf/qwen/qwen1.5-1.8b-chat": "https://huggingface.co/qwen/qwen1.5-1.8b-chat",
    "@cf/defog/sqlcoder-7b-2": "https://huggingface.co/defog/sqlcoder-7b-2",
    "@cf/microsoft/phi-2": "https://huggingface.co/microsoft/phi-2",
    "@hf/google/gemma-7b-it": "https://ai.google.dev/gemma/docs",
}


def get_lm(name: str) -> dspy.LM:
    return dspy.CloudflareAI(model=name)


def run_tests():
    """Test the providers and models"""
    # Configure your AWS credentials with the AWS CLI before running this script
    models
```

```
from pytest_mock.plugin import MockerFixture
from transformers import AutoModelForSeq2SeqLM

import dspy


class MockConfig:
    def __init__(self, architectures: list[str]):
        self.architectures = architectures


def test_load_gated_model(mocker: MockerFixture):
    conf = MockConfig(architectures=["ConditionalGeneration"])
    mocker.patch("transformers.AutoModelForSeq2SeqLM.from_pretrained")
    mocker.patch("transformers.AutoConfig.from_pretrained", return_value=conf)
    mocker.patch("transformers.AutoTokenizer.from_pretrained")

    some_token = "asdfasdfasdf"
    model = "google/gemma-7b"
    _ = dspy.HFModel(model, token=some_token)
    AutoModelForSeq2SeqLM.from_pretrained.assert_called_with(model, device_map="auto", toker


def test_load_ungated_model(mocker: MockerFixture):
    conf = MockConfig(architectures=["ConditionalGeneration"])
    mocker.patch("transformers.AutoModelForSeq2SeqLM.from_pretrained")
    mocker.patch("transformers.AutoConfig.from_pretrained", return_value=conf)
    mocker.patch("transformers.AutoTokenizer.from_pretrained")
    _ = dspy.HFModel("openai-community/gpt2")
    # no token used in automodel
    AutoModelForSeq2SeqLM.from_pretrained.assert_called_with("openai-community/gpt2", device
```

```
from dsp.modules.sentence_vectorizer import FastEmbedVectorizer
import pytest

from dspy.primitives.example import Example

# Skip the test if the 'fastembed' package is not installed
pytest.importorskip("fastembed", reason="'fastembed' is not installed. Use `pip install fastembed` to


@pytest.mark.parametrize(
    "n_dims,model_name", [(384, "BAAI/bge-small-en-v1.5"), (512, "jinaai/jina-embeddings-v2-small
)
def test_fastembed_with_examples(n_dims, model_name):
    vectorizer = FastEmbedVectorizer(model_name)

    examples = [
        Example(query="What's the price today?", response="The price is $10.00").with_inputs("query
        Example(query="What's the weather today?", response="The weather is sunny").with_inputs("
        Example(query="Who was leading the team?", response="It was Jim. Rather enthusiastic guy
            "query", "response"
        ),
    ]

    embeddings = vectorizer(examples)

    assert embeddings.shape == (len(examples), n_dims)


@pytest.mark.parametrize(
    "n_dims,model_name", [(384, "BAAI/bge-small-en-v1.5"), (512, "jinaai/jina-embeddings-v2-small
)
def test_fastembed_with_strings(n_dims, model_name):
    vectorizer = FastEmbedVectorizer(model_name)

    inputs = [
        "Jonathan Kent is a fictional character appearing in American comic books published by DC C
        "Clark Kent is a fictional character appearing in American comic books published by DC Comi
        "Martha Kent is a fictional character appearing in American comic books published by DC Cor
    ]

    embeddings = vectorizer(inputs)

    assert embeddings.shape == (len(inputs), n_dims)
```

```
import dspy
from dspy.primitives.program import (
    Module,
    set_attribute_by_name,
)  # Adjust the import based on your file structure
from dspy.utils import DummyLM


class HopModule(dspy.Module):
    def __init__(self):
        super().__init__()
        self.predict1 = dspy.Predict("question -> query")
        self.predict2 = dspy.Predict("query -> answer")

    def forward(self, question):
        query = self.predict1(question=question).query
        return self.predict2(query=query)


def test_module_initialization():
    module = Module()
    assert module._compiled is False, "Module _compiled attribute should be False upon initializatio


def test_named_predictors():
    module = HopModule()
    named_preds = module.named_predictors()
    assert len(named_preds) == 2, "Should identify correct number of Predict instances"
    names, preds = zip(*named_preds)
    assert "predict1" in names and "predict2" in names, "Named predictors should include 'predict1' a


def test_predictors():
    module = HopModule()
    preds = module.predictors()
    assert len(preds) == 2, "Should return correct number of Predict instances"
    assert all(isinstance(p, dspy.Predict) for p in preds), "All returned items should be instances of Pr


def test_forward():
    program = HopModule()
    dspy.settings.configure(lm=DummyLM({"What is 1+1?": "let me check", "let me check": "2"}))
    result = program(question="What is 1+1?").answer
    assert result == "2"


def test_nested_named_predictors():
```

```
import pytest
from dspy import Example


def test_example_initialization():
    example = Example(a=1, b=2)
    assert example.a == 1
    assert example.b == 2


def test_example_initialization_from_base():
    base = Example(a=1, b=2)
    example = Example(base=base, c=3)
    assert example.a == 1
    assert example.b == 2
    assert example.c == 3


def test_example_initialization_from_dict():
    base_dict = {"a": 1, "b": 2}
    example = Example(base=base_dict, c=3)
    assert example.a == 1
    assert example.b == 2
    assert example.c == 3


def test_example_set_get_item():
    example = Example()
    example["a"] = 1
    assert example["a"] == 1


def test_example_attribute_access():
    example = Example(a=1)
    assert example.a == 1
    example.a = 2
    assert example.a == 2


def test_example_deletion():
    example = Example(a=1, b=2)
    del example["a"]
    with pytest.raises(AttributeError):
        _ = example.a


def test_example_len():
```

```
import pytest
from dspy.primitives.python_interpreter import PythonInterpreter, TextPrompt, CodePrompt

def test_execute_simple_code():
    interpreter = PythonInterpreter(action_space={'print': print})
    code = "print('Hello, World!')"
    result = interpreter.execute(code)
    assert result is None, "Simple print statement should return None"

def test_action_space_limitation():
    def func(string):
        pass
    interpreter = PythonInterpreter(action_space={})
    code = "func('This should not execute')"
    with pytest.raises(Exception):
        interpreter.execute(code)

def test_import_whitelist():
    interpreter = PythonInterpreter(action_space={}, import_white_list=['math'])
    code = "import math\nresult = math.sqrt(4)"
    result = interpreter.execute(code)
    assert result == 2, "Should be able to import and use math.sqrt"

def test_fuzzy_variable_matching():
    interpreter = PythonInterpreter(action_space={})
    code = "result = number + 1"
    result = interpreter.execute(code, fuzz_state={'number': 4})
    assert result == 5, "Fuzzy variable matching should work"

def test_text_prompt_keyword_extraction():
    prompt = TextPrompt("Hello {name}, how are you?")
    assert 'name' in prompt.key_words, "Keyword 'name' should be extracted"

def test_text_prompt_formatting():
    prompt = TextPrompt("Hello {name}, how are you?")
    formatted = prompt.format(name="Alice")
    assert formatted == "Hello Alice, how are you?", "Should format with provided value"

def test_code_prompt_execution():
    action_space = {'len': len}
    interpreter = PythonInterpreter(action_space=action_space)
    code_prompt = CodePrompt("result = len('hello')")
    result, _ = code_prompt.execute(interpreter)
    assert result == 5, "Code execution should return the length of 'hello'"
```

```
import signal
import threading

import pytest

import dsp
import dspy
from dspy.evaluate.evaluate import Evaluate
from dspy.evaluate.metrics import answer_exact_match
from dspy.predict import Predict
from dspy.utils.dummies import DummyLM


def new_example(question, answer):
    """Helper function to create a new example."""
    return dspy.Example(
        question=question,
        answer=answer,
    ).with_inputs("question")


def test_evaluate_initialization():
    devset = [new_example("What is 1+1?", "2")]
    ev = Evaluate(
        devset=devset,
        metric=answer_exact_match,
        display_progress=False,
    )
    assert ev.devset == devset
    assert ev.metric == answer_exact_match
    assert ev.num_threads == len(devset)
    assert ev.display_progress == False


def test_evaluate_call():
    dspy.settings.configure(lm=DummyLM({"What is 1+1?": "2", "What is 2+2?": "4"}))
    devset = [new_example("What is 1+1?", "2"), new_example("What is 2+2?", "4")]
    program = Predict("question -> answer")
    assert program(question="What is 1+1?").answer == "2"
    ev = Evaluate(
        devset=devset,
        metric=answer_exact_match,
        display_progress=False,
    )
    score = ev(program)
    assert score == 100.0
```

```
# FILEPATH: /Users/ahle/repos/dspy/tests/evaluate/test_metrics.py

import dsp, dspy
from dspy.evaluate.metrics import answer_exact_match
from dspy.predict import Predict

def test_answer_exact_match_string():
    example = dspy.Example(
        question="What is 1+1?",
        answer="2",
    ).with_inputs("question")
    pred = Predict("question -> answer")
    pred.answer = "2"
    assert answer_exact_match(example, pred)

def test_answer_exact_match_list():
    example = dspy.Example(
        question="What is 1+1?",
        answer=["2", "two"],
    ).with_inputs("question")
    pred = Predict("question -> answer")
    pred.answer = "2"
    assert answer_exact_match(example, pred)

def test_answer_exact_match_no_match():
    example = dspy.Example(
        question="What is 1+1?",
        answer="2",
    ).with_inputs("question")
    pred = Predict("question -> answer")
    pred.answer = "3"
    assert not answer_exact_match(example, pred)
```

```
import unittest
import uuid

import pandas as pd

from dspy import Example
from dspy.datasets.dataset import Dataset

dummy_data = """content,question,answer
"This is content 1","What is this?","This is answer 1"
"This is content 2","What is that?","This is answer 2"
"""

with open("dummy.csv", "w") as file:
    file.write(dummy_data)


class CSVDataset(Dataset):
    def __init__(self, file_path, input_keys=None, *args, **kwargs) -> None:
        super().__init__(input_keys=input_keys, *args, **kwargs)
        df = pd.read_csv(file_path)
        data = df.to_dict(orient="records")
        self._train = [
            Example(**record, dspy_uuid=str(uuid.uuid4()), dspy_split="train").with_inputs(*input_keys)
            for record in data[:1]
        ]
        self._dev = [
            Example(**record, dspy_uuid=str(uuid.uuid4()), dspy_split="dev").with_inputs(*input_keys)
            for record in data[1:2]
        ]


class TestCSVDataset(unittest.TestCase):
    def test_input_keys(self):
        dataset = CSVDataset("dummy.csv", input_keys=["content", "question"])
        self.assertIsNotNone(dataset.train)

        for example in dataset.train:
            print(example)
            inputs = example.inputs()
            print(f"Example inputs: {inputs}")
            self.assertIsNotNone(inputs)
            self.assertIn("content", inputs)
            self.assertIn("question", inputs)
            self.assertEqual(set(example._input_keys), {"content", "question"})
```

```
import datetime
import textwrap
from typing import Annotated, Any, Generic, List, Literal, Optional, TypeVar

import pydantic
import pytest
from pydantic import AfterValidator, BaseModel, Field, ValidationError, field_validator, model_valida

import dspy
from dspy.functional import FunctionalModule, TypedChainOfThought, TypedPredictor, cot, predict
from dspy.predict.predict import Predict
from dspy.primitives.example import Example
from dspy.teleprompt.bootstrap import BootstrapFewShot
from dspy.teleprompt.vanilla import LabeledFewShot
from dspy.utils.dummies import DummyLM


def test_simple():
    @predictor
    def hard_question(topic: str) -> str:
        """Think of a hard factual question about a topic."""

    expected = "What is the speed of light?"
    lm = DummyLM([expected])
    dspy.settings.configure(lm=lm)

    question = hard_question(topic="Physics")
    lm.inspect_history(n=2)

    assert question == expected


def test_list_output():
    @predictor
    def hard_questions(topics: List[str]) -> List[str]:
        pass

    expected = ["What is the speed of light?", "What is the speed of sound?"]
    lm = DummyLM(['{"value": ["What is the speed of light?", "What is the speed of sound?"]}'])
    dspy.settings.configure(lm=lm)

    question = hard_questions(topics=["Physics", "Music"])
    lm.inspect_history(n=2)

    assert question == expected
```

```
from typing import Generic, TypeVar

import pydantic
import dspy
from dspy.evaluate import Evaluate
from dspy.functional import TypedPredictor
from dspy.teleprompt.signature_opt_typed import optimize_signature, make_info
from dspy.utils import DummyLM

from dspy.evaluate import Evaluate
from dspy.evaluate.metrics import answer_exact_match
from dspy.functional import TypedPredictor


hotpotqa = [
    ex.with_inputs("question")
    for ex in [
        dspy.Example(
            question="At My Window was released by which American singer-songwriter?",
            answer="John Townes Van Zandt",
        ),
        dspy.Example(
            question="which  American actor was Candace Kita  guest starred with ",
            answer="Bill Murray",
        ),
        dspy.Example(
            question="Which of these publications was most recently published, Who Put the Bomp or S
            answer="Self",
        ),
        dspy.Example(
            question="The Victorians - Their Story In Pictures is a documentary series written by an aut
            answer="1950",
        ),
        dspy.Example(
            question="Which magazine has published articles by Scott Shaw, Tae Kwon Do Times or S
            answer="Tae Kwon Do Times",
        ),
        dspy.Example(
            question="In what year was the club founded that played Manchester City in the 1972 FA C
            answer="1874",
        ),
        dspy.Example(
            question="Which is taller, the Empire State Building or the Bank of America Tower?",
            answer="The Empire State Building",
        ),
        dspy.Example(
            question='Which American actress who made their film debut in the 1995 teen drama "Kids'
```

```
"""Instructions:
Add to dev container features:
    "ghcr.io/itsmechlark/features/postgresql:1": {},
    "ghcr.io/robbert229/devcontainer-features/postgresql-client:1": {}
Add to .personalization.sh:
    sudo apt install -y postgresql-16-pgvector

    sudo /etc/init.d/postgresql restart

    psql -v ON_ERROR_STOP=1 --user ${PGUSER} <<EOF
    create extension if not exists vector;
    EOF
poetry install -E postgres
"""
from dspy.primitives.example import Example
import pytest
import psycopg2
from dspy.retrieve.pgvector_rm import PgVectorRM

DB_URL = "postgresql://postgres:password@localhost/postgres"
PG_TABLE_NAME = "test_table"


def get_pgvectorrm():
    openai_client = None  # Mock or use a real OpenAI client
    pgvectorrm = PgVectorRM(DB_URL, PG_TABLE_NAME, openai_client=openai_client, embeddi
    return pgvectorrm


@pytest.fixture
def setup_pgvectorrm():
    pgvectorrm = get_pgvectorrm()
    conn = psycopg2.connect(DB_URL)
    cursor = conn.cursor()
    cursor.execute(f'DROP TABLE IF EXISTS {PG_TABLE_NAME}')
    conn.commit()

    cursor.execute(f"CREATE TABLE IF NOT EXISTS {PG_TABLE_NAME} (id SERIAL PRIMARY K
    cursor.execute(f"INSERT INTO {PG_TABLE_NAME} (text, embedding) VALUES ('Dummy text1'
    conn.commit()

    yield pgvectorrm

    cursor.execute(f'TRUNCATE TABLE {PG_TABLE_NAME}')
    conn.commit()

    cursor.close()
```

```
import logging

import pytest

import dspy
from dsp.modules.dummy_lm import DummyLM
from dspy.datasets import HotPotQA

try:
    from llama_index.core import Settings, VectorStoreIndex
    from llama_index.core.base.base_retriever import BaseRetriever
    from llama_index.core.embeddings.mock_embed_model import MockEmbedding
    from llama_index.core.readers.string_iterable import StringIterableReader

    from dspy.retrieve.llama_index_rm import LlamaIndexRM

except ImportError:
    logging.info("Optional dependency llama-index is not installed - skipping LlamaIndexRM tests.")


@pytest.fixture()
def rag_setup() -> dict:
    """Builds the necessary fixtures to test LI"""
    pytest.importorskip("llamaindex")
    dataset = HotPotQA(train_seed=1, train_size=8, eval_seed=2023, dev_size=4, test_size=0)
    trainset = [x.with_inputs("question") for x in dataset.train]
    devset = [x.with_inputs("question") for x in dataset.dev]
    ragset = [f"Question: {x.question} Answer: {x.answer}" for x in dataset.train]
    dummyset = {x.question: x.answer for x in dataset.train}

    Settings.embed_model = MockEmbedding(8)
    docs = StringIterableReader().load_data(texts=ragset)
    index = VectorStoreIndex.from_documents(documents=docs)
    retriever = index.as_retriever()
    rm = LlamaIndexRM(retriever)

    return {
        "index": index,
        "retriever": retriever,
        "rm": rm,
        "lm": DummyLM(answers=dummyset),
        "trainset": trainset,
        "devset": devset,
    }


def test_lirm_as_rm(rag_setup):
```

```
import pytest
import dspy
from dspy.predict import Predict
from dspy.utils.dummies import DummyLM
from dspy import Example
from dspy.teleprompt import BootstrapFewShot
import textwrap


# Define a simple metric function for testing
def simple_metric(example, prediction, trace=None):
    # Simplified metric for testing: true if prediction matches expected output
    return example.output == prediction.output


examples = [
    Example(input="What is the color of the sky?", output="blue").with_inputs("input"),
    Example(
        input="What does the fox say?", output="Ring-ding-ding-ding-dingeringeding!"
    ),
]
trainset = [examples[0]]
valset = [examples[1]]


def test_bootstrap_initialization():
    # Initialize BootstrapFewShot with a dummy metric and minimal setup
    bootstrap = BootstrapFewShot(
        metric=simple_metric, max_bootstrapped_demos=1, max_labeled_demos=1
    )
    assert bootstrap.metric == simple_metric, "Metric not correctly initialized"


class SimpleModule(dspy.Module):
    def __init__(self, signature):
        super().__init__()
        self.predictor = Predict(signature)

    def forward(self, **kwargs):
        return self.predictor(**kwargs)


def test_compile_with_predict_instances():
    # Create Predict instances for student and teacher
    # Note that dspy.Predict is not itself a module, so we can't use it directly here
    student = SimpleModule("input -> output")
    teacher = SimpleModule("input -> output")
```

```
import textwrap
import pytest
import re
import dspy
from dsp.modules import LM
from dspy.teleprompt.signature_opt_bayesian import MIPRO
from dspy.utils.dummies import DummyLM
from dspy import Example


# Define a simple metric function for testing
def simple_metric(example, prediction, trace=None):
    # Simplified metric for testing: true if prediction matches expected output
    return example.output == prediction.output

# Some example data
capitals = {
    "Germany": "Berlin",
    "France": "Paris",
    "Denmark": "Copenhagen",
    "Sweden": "Stockholm",
    "Norway": "Oslo",
}
# Not used for training data
extra_capitals = {
    "Spain": "Madrid",
    "Portugal": "Lisbon",
    "Italy": "Rome",
}

# Example training and validation sets
trainset = [
    Example(input="What is the color of the sky?", output="blue").with_inputs("input"),
    Example(
        input="What does the fox say?", output="Ring-ding-ding-ding-dingeringeding!"
    ).with_inputs("input"),
] + [Example(input=f"What is the capital of {country}?", output=capital).with_inputs("input") for coun


class ConditionalLM(LM):
    def __init__(self):
        super().__init__("conditional-lm")

    def basic_request(self, prompt, num_candidates=1, **kwargs):
        # If we are in the "optimization" stage, we don't say much.
        if prompt.endswith("Observations:"):
            answer = " (*silence*)"
```

TODO

```
import pytest
import dspy
from dspy.teleprompt import Ensemble


class MockProgram(dspy.Module):
    def __init__(self, output):
        super().__init__()
        self.output = output

    def forward(self, *args, **kwargs):
        return self.output


# Simple reduction function to test with
def mock_reduce_fn(outputs):
    return sum(outputs) / len(outputs)


def test_ensemble_without_reduction():
    """Test that Ensemble correctly combines outputs without applying a reduce_fn."""
    programs = [MockProgram(i) for i in range(5)]
    ensemble = Ensemble()
    ensembled_program = ensemble.compile(programs)

    outputs = ensembled_program()
    assert len(outputs) == 5, "Ensemble did not combine the correct number of outputs"


def test_ensemble_with_reduction():
    """Test that Ensemble correctly applies a reduce_fn to combine outputs."""
    programs = [MockProgram(i) for i in range(5)]
    ensemble = Ensemble(reduce_fn=mock_reduce_fn)
    ensembled_program = ensemble.compile(programs)

    output = ensembled_program()
    expected_output = sum(range(5)) / 5
    assert output == expected_output, "Ensemble did not correctly apply the reduce_fn"


def test_ensemble_with_size_limitation():
    """Test that specifying a size limits the number of programs used in the ensemble."""
    programs = [MockProgram(i) for i in range(10)]
    ensemble_size = 3
    ensemble = Ensemble(size=ensemble_size)
    ensembled_program = ensemble.compile(programs)
```

```
import pytest
import dsp, dspy
from dspy.teleprompt.knn_fewshot import KNNFewShot
from dspy.utils.dummies import DummyLM, DummyVectorizer


def mock_example(question: str, answer: str) -> dsp.Example:
    """Creates a mock DSP example with specified question and answer."""
    return dspy.Example(question=question, answer=answer).with_inputs("question")


@pytest.fixture
def setup_knn_few_shot():
    """Sets up a KNNFewShot instance for testing."""
    trainset = [
        mock_example("What is the capital of France?", "Paris"),
        mock_example("What is the largest ocean?", "Pacific"),
        mock_example("What is 2+2?", "4"),
    ]
    dsp.SentenceTransformersVectorizer = DummyVectorizer
    knn_few_shot = KNNFewShot(k=2, trainset=trainset)
    return knn_few_shot


def test_knn_few_shot_initialization(setup_knn_few_shot):
    """Tests the KNNFewShot initialization."""
    knn_few_shot = setup_knn_few_shot
    assert knn_few_shot.KNN.k == 2, "Incorrect k value for KNN"
    assert len(knn_few_shot.KNN.trainset) == 3, "Incorrect trainset size for KNN"


class SimpleModule(dspy.Module):
    def __init__(self, signature):
        super().__init__()
        self.predictor = dspy.Predict(signature)

    def forward(self, *args, **kwargs):
        return self.predictor(**kwargs)

    def reset_copy(self):
        # Creates a new instance of SimpleModule with the same predictor
        return SimpleModule(self.predictor.signature)


# TODO: Test not working yet
def _test_knn_few_shot_compile(setup_knn_few_shot):
    """Tests the compile method of KNNFewShot with SimpleModule as student."""
```

```
import textwrap
import dspy
from dspy.teleprompt.signature_opt import COPRO
from dspy.utils.dummies import DummyLM
from dspy import Example

# Define a simple metric function for testing
def simple_metric(example, prediction):
    # Simplified metric for testing: true if prediction matches expected output
    return example.output == prediction.output

# Example training and validation sets
trainset = [
    Example(input="Question: What is the color of the sky?", output="blue").with_inputs("input"),
    Example(input="Question: What does the fox say?", output="Ring-ding-ding-ding-dingeringeding
]

def test_signature_optimizer_initialization():
    optimizer = COPRO(metric=simple_metric, breadth=2, depth=1, init_temperature=1.4)
    assert optimizer.metric == simple_metric, "Metric not correctly initialized"
    assert optimizer.breadth == 2, "Breadth not correctly initialized"
    assert optimizer.depth == 1, "Depth not correctly initialized"
    assert optimizer.init_temperature == 1.4, "Initial temperature not correctly initialized"

class SimpleModule(dspy.Module):
    def __init__(self, signature):
        super().__init__()
        # COPRO doesn't work with dspy.Predict
        self.predictor = dspy.ChainOfThought(signature)

    def forward(self, **kwargs):
        return self.predictor(**kwargs)

def test_signature_optimizer_optimization_process():
    optimizer = COPRO(metric=simple_metric, breadth=2, depth=1, init_temperature=1.4)
    dspy.settings.configure(lm=DummyLM(["Optimized instruction 1", "Optimized instruction 2"]))

    student = SimpleModule("input -> output")

    # Assuming the compile method of COPRO requires a student module, a development set, and
    optimized_student = optimizer.compile(student, trainset=trainset, eval_kwargs={"num_threads":

    # Check that the optimized student has been modified from the original
    # This check can be more specific based on how the optimization modifies the student
    assert optimized_student is not student, "Optimization did not modify the student"

    # Further tests can be added to verify the specifics of the optimization process,
```

```
from dataclasses import dataclass

import dspy
from dspy.utils.dummies import dummy_rm


def test_example_no_tools():
    # Createa a simple dataset which the model will use with the Retrieve tool.
    lm = dspy.utils.DummyLM(
        [
            "Initial thoughts",  # Thought_1
            "Finish[blue]",  # Action_1
        ]
    )
    dspy.settings.configure(lm=lm, rm=dummy_rm())

    program = dspy.ReAct("question -> answer")

    # Check default tools
    assert isinstance(program.tools["Finish"], dspy.Example)

    # Call the ReAct module on a particular input
    question = "What is the color of the sky?"
    result = program(question=question)
    assert result.answer == "blue"

    # For debugging
    print("---")
    for row in lm.history:
        print(row["prompt"])
        print("Response:", row["response"]["choices"][0]["text"])
        print("---")

    assert lm.get_convo(-1).endswith(
        "Question: What is the color of the sky?\n"
        "Thought 1: Initial thoughts\n"
        "Action 1: Finish[blue]"
    )


def test_example_search():
    # Createa a simple dataset which the model will use with the Retrieve tool.
    lm = dspy.utils.DummyLM(
        [
            "Initial thoughts",  # Thought_1
            "Search[the color of the sky]",  # Thought_1
            "More thoughts",  # Thought_2
```

```
from dspy import Signature, ProgramOfThought
import dspy
from dspy.utils import DummyLM
import textwrap

class BasicQA(Signature):
    question = dspy.InputField()
    answer = dspy.OutputField(desc="often between 1 and 5 words")

def test_pot_code_generation():
    pot = ProgramOfThought(BasicQA)
    lm = DummyLM([
        "Reason_A",
        "```python\nresult = 1+1\n```",
        "Reason_B",
        "2",
    ])
    dspy.settings.configure(lm=lm)
    res = pot(question="What is 1+1?")
    assert res.answer == "2"
    assert lm.get_convo(index=-1) == textwrap.dedent("""\
        Given the final code `question`, `final_generated_code`, `code_output`, provide the final `answ

        ---

        Follow the following format.

        Question: ${question}

        Code: python code that answers the question

        Code Output: output of previously-generated python code

        Reasoning: Let's think step by step in order to ${produce the answer}. We ...

        Answer: often between 1 and 5 words

        ---

        Question: What is 1+1?

        Code: result = 1+1

        Code Output: 2

        Reasoning: Let's think step by step in order to Reason_B
```

```
import dspy
from dspy import Predict, Signature, TypedPredictor
from dspy.utils.dummies import DummyLM
import copy
import textwrap
import pydantic
import ujson


def test_initialization_with_string_signature():
    signature_string = "input1, input2 -> output"
    predict = Predict(signature_string)
    expected_instruction = "Given the fields `input1`, `input2`, produce the fields `output`."
    assert predict.signature.instructions == expected_instruction
    assert predict.signature.instructions == Signature(signature_string).instructions


def test_reset_method():
    predict_instance = Predict("input -> output")
    predict_instance.lm = "modified"
    predict_instance.traces = ["trace"]
    predict_instance.train = ["train"]
    predict_instance.demos = ["demo"]
    predict_instance.reset()
    assert predict_instance.lm is None
    assert predict_instance.traces == []
    assert predict_instance.train == []
    assert predict_instance.demos == []


def test_lm_after_dump_and_load_state():
    predict_instance = Predict("input -> output")
    predict_instance.lm = "lm_state"
    dumped_state = predict_instance.dump_state()
    new_instance = Predict("input -> output")
    new_instance.load_state(dumped_state)
    assert new_instance.lm == "lm_state"


def test_call_method():
    predict_instance = Predict("input -> output")
    lm = DummyLM(["test output"])
    dspy.settings.configure(lm=lm)
    result = predict_instance(input="test input")
    assert result.output == "test output"
    assert lm.get_convo(-1) == (
        "Given the fields `input`, produce the fields `output`.\n"
```

```
import dspy
from dspy.utils.dummies import DummyLM


def test_basic_example():
    class BasicQA(dspy.Signature):
        """Answer questions with short factoid answers."""

        question = dspy.InputField()
        answer = dspy.OutputField(desc="often between 1 and 5 words")

    # Example completions generated by a model for reference
    completions = [
        dspy.Prediction(
            rationale="I recall that during clear days, the sky often appears this color.",
            answer="blue",
        ),
        dspy.Prediction(
            rationale="Based on common knowledge, I believe the sky is typically seen as this color.",
            answer="green",
        ),
        dspy.Prediction(
            rationale="From images and depictions in media, the sky is frequently represented with this
            answer="blue",
        ),
    ]

    # Pass signature to MultiChainComparison module
    compare_answers = dspy.MultiChainComparison(BasicQA)

    # Call the MultiChainComparison on the completions
    question = "What is the color of the sky?"
    lm = DummyLM(["my rationale", "blue"])
    dspy.settings.configure(lm=lm)
    final_pred = compare_answers(completions, question=question)

    assert final_pred.rationale == "my rationale"
    assert final_pred.answer == "blue"
```

```
import functools
import dspy
from dspy.utils import DummyLM
from dspy.primitives.assertions import assert_transform_module, backtrack_handler
import pydantic


def test_retry_simple():
    predict = dspy.Predict("question -> answer")
    retry_module = dspy.Retry(predict)

    # Test Retry has created the correct new signature
    for field in predict.signature.output_fields:
        assert f"past_{field}" in retry_module.new_signature.input_fields
    assert "feedback" in retry_module.new_signature.input_fields

    lm = DummyLM(["blue"])
    dspy.settings.configure(lm=lm)
    result = retry_module.forward(
        question="What color is the sky?",
        past_outputs={"answer": "red"},
        feedback="Try harder",
    )
    assert result.answer == "blue"

    print(lm.get_convo(-1))
    assert lm.get_convo(-1).endswith(
        "Question: What color is the sky?\n\n"
        "Previous Answer: red\n\n"
        "Instructions: Try harder\n\n"
        "Answer: blue"
    )


def test_retry_forward_with_feedback():
    # First we make a mistake, then we fix it
    lm = DummyLM(["red", "blue"])
    dspy.settings.configure(lm=lm, trace=[])

    class SimpleModule(dspy.Module):
        def __init__(self):
            super().__init__()
            self.predictor = dspy.Predict("question -> answer")

        def forward(self, **kwargs):
            result = self.predictor(**kwargs)
            print(f"SimpleModule got {result.answer=}")
```

```
import textwrap
import dspy
from dspy import ChainOfThought
from dspy.utils import DummyLM


def test_initialization_with_string_signature():
    lm = DummyLM(["find the number after 1", "2"])
    dspy.settings.configure(lm=lm)
    predict = ChainOfThought("question -> answer")
    assert list(predict.extended_signature.output_fields.keys()) == [
        "rationale",
        "answer",
    ]
    assert predict(question="What is 1+1?").answer == "2"

    print(lm.get_convo(-1))
    assert lm.get_convo(-1) == textwrap.dedent(
        """\
        Given the fields `question`, produce the fields `answer`.

        ---

        Follow the following format.

        Question: ${question}
        Reasoning: Let's think step by step in order to ${produce the answer}. We ...
        Answer: ${answer}

        ---

        Question: What is 1+1?
        Reasoning: Let's think step by step in order to find the number after 1
        Answer: 2"""
    )
```

```
from dspy.predict.aggregation import majority
from dspy.primitives.prediction import Prediction, Completions
from dsp.utils import normalize_text


def test_majority_with_prediction():
    prediction = Prediction.from_completions(
        [{"answer": "2"}, {"answer": "2"}, {"answer": "3"}]
    )
    result = majority(prediction)
    assert result.completions[0]["answer"] == "2"


def test_majority_with_completions():
    completions = Completions([{"answer": "2"}, {"answer": "2"}, {"answer": "3"}])
    result = majority(completions)
    assert result.completions[0]["answer"] == "2"


def test_majority_with_list():
    completions = [{"answer": "2"}, {"answer": "2"}, {"answer": "3"}]
    result = majority(completions)
    assert result.completions[0]["answer"] == "2"


def test_majority_with_normalize():
    completions = [{"answer": "2"}, {"answer": " 2"}, {"answer": "3"}]
    result = majority(completions, normalize=normalize_text)
    assert result.completions[0]["answer"] == "2"


def test_majority_with_field():
    completions = [
        {"answer": "2", "other": "1"},
        {"answer": "2", "other": "1"},
        {"answer": "3", "other": "2"},
    ]
    result = majority(completions, field="other")
    assert result.completions[0]["other"] == "1"


def test_majority_with_no_majority():
    completions = [{"answer": "2"}, {"answer": "3"}, {"answer": "4"}]
    result = majority(completions)
    assert (
        result.completions[0]["answer"] == "2"
    )  # The first completion is returned in case of a tie
```

```
import pytest
import numpy as np
import dsp, dspy
from dspy.utils import DummyVectorizer
from dspy.predict import KNN


def mock_example(question: str, answer: str) -> dsp.Example:
    """Creates a mock DSP example with specified question and answer."""
    return dspy.Example(question=question, answer=answer).with_inputs("question")


@pytest.fixture
def setup_knn():
    """Sets up a KNN instance with a mocked vectorizer for testing."""
    dsp.SentenceTransformersVectorizer = DummyVectorizer
    trainset = [
        mock_example("What is the capital of France?", "Paris"),
        mock_example("What is the largest ocean?", "Pacific"),
        mock_example("What is 2+2?", "4"),
    ]
    knn = KNN(k=2, trainset=trainset)
    return knn


def test_knn_initialization(setup_knn):
    """Tests the KNN initialization and checks if the trainset vectors are correctly created."""
    knn = setup_knn
    assert knn.k == 2, "Incorrect k value"
    assert len(knn.trainset_vectors) == 3, "Incorrect size of trainset vectors"
    assert isinstance(
        knn.trainset_vectors, np.ndarray
    ), "Trainset vectors should be a NumPy array"


def test_knn_query(setup_knn):
    """Tests the KNN query functionality for retrieving the nearest neighbors."""
    knn = setup_knn
    query = {"question": "What is 3+3?"}  # A query close to "What is 2+2?"
    nearest_samples = knn(**query)
    assert len(nearest_samples) == 2, "Incorrect number of nearest samples returned"
    assert nearest_samples[0].answer == "4", "Incorrect nearest sample returned"


def test_knn_query_specificity(setup_knn):
    """Tests the KNN query functionality for specificity of returned examples."""
    knn = setup_knn
```

```
import dspy
from dspy import ChainOfThoughtWithHint
from dspy.utils import DummyLM


def test_cot_with_no_hint():
    lm = DummyLM(["find the number after 1", "2"])
    dspy.settings.configure(lm=lm)
    predict = ChainOfThoughtWithHint("question -> answer")
    # Check output fields have the right order
    assert list(predict.extended_signature2.output_fields.keys()) == [
        "rationale",
        "hint",
        "answer",
    ]
    assert predict(question="What is 1+1?").answer == "2"

    final_convo = lm.get_convo(-1)
    assert final_convo.endswith(
        "Question: What is 1+1?\n"
        "Reasoning: Let's think step by step in order to find the number after 1\n"
        "Answer: 2"
    )


def test_cot_with_hint():
    lm = DummyLM(["find the number after 1", "2"])
    dspy.settings.configure(lm=lm)
    predict = ChainOfThoughtWithHint("question -> answer")
    assert list(predict.extended_signature2.output_fields.keys()) == [
        "rationale",
        "hint",
        "answer",
    ]
    assert predict(question="What is 1+1?", hint="think small").answer == "2"

    final_convo = lm.get_convo(-1)
    assert final_convo.endswith(
        "Question: What is 1+1?\n\n"
        "Reasoning: Let's think step by step in order to find the number after 1\n\n"
        "Hint: think small\n\n"
        "Answer: 2"
    )
```

```
import sys
from datetime import datetime

import requests
import semver
from packaging.version import Version as PyPIVersion


def get_latest_version(package_name, tag_version):
    # Returns latest version, and T/F as to whether it needs to be incremented
    response = requests.get(f"https://test.pypi.org/pypi/{package_name}/json")
    if response.status_code == 200:
        data = response.json()
        # Flatten the list of files for all releases and get the latest upload
        all_uploads = [
            (release['upload_time'], release['filename'], version)
            for version, releases in data['releases'].items()
            for release in releases
        ]
        # If a release with tag_version does not exist, that is the latest version
        # Then increment is False, as no need to increment the version
        tag_release_exists = any(upload for upload in all_uploads if upload[2] == tag_version)
        if not(tag_release_exists):
            return tag_version, False
        # Else, get the latest release version, and set increment to True
        else:
            # Sort all uploads by upload time in descending order
            latest_upload = max(all_uploads, key=lambda x: datetime.fromisoformat(x[0].rstrip('Z')))
            return latest_upload[2], True

    elif response.status_code == 404:
        # If no existing releases can get a 404
        return tag_version, False
    return None, None

def increment_version(curr_version):
    pypi_v = PyPIVersion(curr_version)
    if pypi_v.pre:
        pre = "".join([str(i) for i in pypi_v.pre])
        parsed_v = semver.Version(*pypi_v.release, pre)
    else:
        parsed_v = semver.Version(*pypi_v.release)
    new_v = str(parsed_v.bump_prerelease())
    return new_v

if __name__ == "__main__":
    if len(sys.argv) != 3:
```

```
from .aggregation import majority
from .chain_of_thought import ChainOfThought
from .chain_of_thought_with_hint import ChainOfThoughtWithHint
from .knn import KNN
from .multi_chain_comparison import MultiChainComparison
from .predict import Predict
from .program_of_thought import ProgramOfThought
from .react import ReAct
from .retry import Retry
```

```
import inspect
import logging
import math
import os
import random
import shutil
import sys

import numpy as np

try:
    from IPython.core.magics.code import extract_symbols
except ImportError:
    # Won't be able to read code from juptyer notebooks
    extract_symbols = None

import dspy
from dspy.predict.parameter import Parameter
from dspy.teleprompt.bootstrap import BootstrapFewShot, LabeledFewShot

"""
This file consists of helper functions for our variety of optimizers.
"""

### OPTIMIZER TRAINING UTILS ###


def create_minibatch(trainset, batch_size=50):
    """Create a minibatch from the trainset."""

    # Ensure batch_size isn't larger than the size of the dataset
    batch_size = min(batch_size, len(trainset))

    # Randomly sample indices for the mini-batch
    sampled_indices = random.sample(range(len(trainset)), batch_size)

    # Create the mini-batch using the sampled indices
    minibatch = [trainset[i] for i in sampled_indices]

    return minibatch


def eval_candidate_program(batch_size, trainset, candidate_program, evaluate):
    """Evaluate a candidate program on the trainset, using the specified batch size."""
    # Evaluate on the full trainset
    if batch_size >= len(trainset):
        score = evaluate(candidate_program, devset=trainset, display_table=0)
```

```
import random
import re

import dspy
from dspy.propose.dataset_summary_generator import create_dataset_summary
from dspy.propose.utils import create_example_string, create_predictor_level_history_string, strip_
from dspy.teleprompt.utils import get_signature

from .propose_base import Proposer

# Hardcoded variables (TODO: update)
MAX_INSTRUCT_IN_HISTORY = 5  # 10

TIPS = {
        "none": "",
        "creative": "Don't be afraid to be creative when creating the new instruction!",
        "simple": "Keep the instruction clear and concise.",
        "description": "Make sure your instruction is very informative and descriptive.",
        "high_stakes": "The instruction should include a high stakes scenario in which the LM must so
        "persona": 'Include a persona that is relevant to the task in the instruction (ie. "You are a ...")',
    }

### SIGNATURES USED TO HELP WITH INSTRUCTION GENERATION ###

class DescribeProgram(dspy.Signature):
    (
        """Below is some pseudo-code for a pipeline that solves tasks with calls to language models. P
    )
    program_code = dspy.InputField(
        format=str,
        desc="Pseudocode for a language model program designed to solve a particular task.",
        prefix="PROGRAM CODE:",
    )
    program_example = dspy.InputField(
        format=str,
        desc="An example of the program in use.",
        prefix="EXAMPLE OF PROGRAM IN USE:",
    )
    program_description = dspy.OutputField(
        desc="Describe what task the program is designed to solve, and how it goes about solving this
        prefix="SUMMARY OF PROGRAM ABOVE:",
    )


class DescribeModule(dspy.Signature):
    (
        """Below is some pseudo-code for a pipeline that solves tasks with calls to language models. P
```

```
from .aggregation import majority
from .chain_of_thought import ChainOfThought
from .chain_of_thought_with_hint import ChainOfThoughtWithHint
from .knn import KNN
from .multi_chain_comparison import MultiChainComparison
from .predict import Predict
from .program_of_thought import ProgramOfThought
from .react import ReAct
from .retry import Retry
```

```
import re

import dspy
from dspy.propose.utils import strip_prefix


class ObservationSummarizer(dspy.Signature):
    ("""Given a series of observations I have made about my dataset, please summarize them into a
    observations = dspy.InputField(desc="Observations I have made about my dataset")
    summary = dspy.OutputField(desc="Two to Three sentence summary of only the most significan

class DatasetDescriptor(dspy.Signature):
    ("""Given several examples from a dataset please write observations about trends that hold for m
    """Some areas you may consider in your observations: topics, content, syntax, conciceness, etc.
    """It will be useful to make an educated guess as to the nature of the task this dataset will enable

    examples = dspy.InputField(desc="Sample data points from the dataset")
    observations = dspy.OutputField(desc="Somethings that holds true for most or all of the data you

class DatasetDescriptorWithPriorObservations(dspy.Signature):
    ("""Given several examples from a dataset please write observations about trends that hold for m
    """I will also provide you with a few observations I have already made.  Please add your own obs
    """Some areas you may consider in your observations: topics, content, syntax, conciceness, etc.
    """It will be useful to make an educated guess as to the nature of the task this dataset will enable

    examples = dspy.InputField(desc="Sample data points from the dataset")
    prior_observations = dspy.InputField(desc="Some prior observations I made about the data")
    observations = dspy.OutputField(desc="Somethings that holds true for most or all of the data you

def order_input_keys_in_string(unordered_repr):
    # Regex pattern to match the input keys structure
    pattern = r"input_keys=\{([^\}]+)\}"

    # Function to reorder keys
    def reorder_keys(match):
        # Extracting the keys from the match
        keys_str = match.group(1)
        # Splitting the keys, stripping extra spaces, and sorting them
        keys = sorted(key.strip() for key in keys_str.split(','))
        # Formatting the sorted keys back into the expected structure
        return f"input_keys={{{', '.join(keys)}}}"

    # Using re.sub to find all matches of the pattern and replace them using the reorder_keys functio
    ordered_repr = re.sub(pattern, reorder_keys, unordered_repr)

    return ordered_repr
```

```
import dsp
import dspy
from dspy.signatures import Signature


class BasicGenerateInstruction(dspy.Signature):
    ("""You are an instruction optimizer for large language models. I will give you a ``signature`` of

Your task is to propose a new improved instruction and prefix for the output field that will lead a goo
    example_instructions = dspy.InputField(format=dsp.passages2text, desc="Example instruction
    proposed_instruction = dspy.InputField(desc="The improved instructions for the language mod
    proposed_prefix_for_output_field = dspy.OutputField(desc="The string at the end of the promp

class BasicGenerateInstructionWithExamplesAndDataObservationsAndTip(dspy.Signature):
    ("""You are an instruction optimizer for large language models. I will give you a ``signature`` of

Your task is to propose a new improved instruction and prefix for the output field that will lead a goo
    dataset_summary = dspy.InputField(desc="Summary of the dataset.")
    examples = dspy.InputField(format=dsp.passages2text, desc="Example(s) of the task")
    example_instructions = dspy.InputField(format=dsp.passages2text, desc="Example instruction
    tip = dspy.InputField(desc="A tip for something to keep in mind when generating the new instr
    proposed_instruction = dspy.OutputField(desc="The improved instructions for the language m
    proposed_prefix_for_output_field = dspy.OutputField(desc="The string at the end of the promp

class BasicGenerateInstructionWithDataObservationsAndTip(dspy.Signature):
    ("""You are an instruction optimizer for large language models. I will give you a ``signature`` of

Your task is to propose a new improved instruction and prefix for the output field that will lead a goo
    dataset_summary = dspy.InputField(desc="Summary of the dataset.")
    example_instructions = dspy.InputField(format=dsp.passages2text, desc="Example instruction
    tip = dspy.InputField(desc="A tip for something to keep in mind when generating the new instr
    proposed_instruction = dspy.OutputField(desc="The improved instructions for the language m
    proposed_prefix_for_output_field = dspy.OutputField(desc="The string at the end of the promp

class BasicGenerateInstructionWithExamplesAndTip(dspy.Signature):
    ("""You are an instruction optimizer for large language models. I will give you a ``signature`` of

Your task is to propose a new improved instruction and prefix for the output field that will lead a goo
    examples = dspy.InputField(format=dsp.passages2text, desc="Example(s) of the task")
    example_instructions = dspy.InputField(format=dsp.passages2text, desc="Example instruction
    tip = dspy.InputField(desc="A tip for something to keep in mind when generating the new instr
    proposed_instruction = dspy.OutputField(desc="The improved instructions for the language m
    proposed_prefix_for_output_field = dspy.OutputField(desc="The string at the end of the promp

class BasicGenerateInstructionWithTip(dspy.Signature):
    ("""You are an instruction optimizer for large language models. I will give you a ``signature`` of
```

```
from abc import ABC, abstractmethod


class Proposer(ABC):
    def __init__(self):
        pass

    @abstractmethod
    def propose_instructions_for_program(self):
        pass

    def propose_instruction_for_predictor(self):
        pass
```

```
import pydantic

# The following arguments can be used in DSPy InputField and OutputField in addition
# to the standard pydantic.Field arguments. We just hope pydanitc doesn't add these,
# as it would give a name clash.
DSPY_FIELD_ARG_NAMES = ["desc", "prefix", "format", "parser", "__dspy_field_type"]


def move_kwargs(**kwargs):
    # Pydantic doesn't allow arbitrary arguments to be given to fields,
    # but asks that
    # > any extra data you want to add to the JSON schema should be passed
    # > as a dictionary to the json_schema_extra keyword argument.
    # See: https://docs.pydantic.dev/2.6/migration/#changes-to-pydanticfield
    pydantic_kwargs = {}
    json_schema_extra = {}
    for k, v in kwargs.items():
        if k in DSPY_FIELD_ARG_NAMES:
            json_schema_extra[k] = v
        else:
            pydantic_kwargs[k] = v
    # Also copy over the pydantic "description" if no dspy "desc" is given.
    if "description" in kwargs and "desc" not in json_schema_extra:
        json_schema_extra["desc"] = kwargs["description"]
    pydantic_kwargs["json_schema_extra"] = json_schema_extra
    return pydantic_kwargs


def InputField(**kwargs):
    return pydantic.Field(**move_kwargs(**kwargs, __dspy_field_type="input"))


def OutputField(**kwargs):
    return pydantic.Field(**move_kwargs(**kwargs, __dspy_field_type="output"))


def new_to_old_field(field):
    return (OldInputField if field.json_schema_extra["__dspy_field_type"] == "input" else OldOutputF
        prefix=field.json_schema_extra["prefix"],
        desc=field.json_schema_extra["desc"],
        format=field.json_schema_extra.get("format"),
    )


class OldField:
    """A more ergonomic datatype that infers prefix and desc if omitted."""
```

```
from .aggregation import majority
from .chain_of_thought import ChainOfThought
from .chain_of_thought_with_hint import ChainOfThoughtWithHint
from .knn import KNN
from .multi_chain_comparison import MultiChainComparison
from .predict import Predict
from .program_of_thought import ProgramOfThought
from .react import ReAct
from .retry import Retry
```

```
import ast
import inspect
import re
import types
import typing
from contextlib import ExitStack, contextmanager
from copy import deepcopy
from typing import Any, Dict, Tuple, Type, Union  # noqa: UP035

from pydantic import BaseModel, Field, create_model
from pydantic.fields import FieldInfo

import dsp
from dspy.signatures.field import InputField, OutputField, new_to_old_field


def signature_to_template(signature) -> dsp.Template:
    """Convert from new to legacy format."""
    return dsp.Template(
        signature.instructions,
        **{name: new_to_old_field(field) for name, field in signature.fields.items()},
    )


def _default_instructions(cls) -> str:
    inputs_ = ", ".join([f"`{field}`" for field in cls.input_fields])
    outputs_ = ", ".join([f"`{field}`" for field in cls.output_fields])
    return f"Given the fields {inputs_}, produce the fields {outputs_}."


class SignatureMeta(type(BaseModel)):
    def __call__(cls, *args, **kwargs):  # noqa: ANN002
        if cls is Signature:
            return make_signature(*args, **kwargs)
        return super().__call__(*args, **kwargs)

    def __new__(mcs, signature_name, bases, namespace, **kwargs):  # noqa: N804
        # Set `str` as the default type for all fields
        raw_annotations = namespace.get("__annotations__", {})
        for name, field in namespace.items():
            if not isinstance(field, FieldInfo):
                continue  # Don't add types to non-field attributes
            if not name.startswith("__") and name not in raw_annotations:
                raw_annotations[name] = str
        namespace["__annotations__"] = raw_annotations

        # Let Pydantic do its thing
```

```
import dspy

try:
    import graphviz  # type: ignore
    graphviz_available = True
except ImportError:
    graphviz_available = False


class ModuleGraph:
    def __init__(self, module_name, module):
        if graphviz_available is False:
            raise ImportError(
                """Please install graphviz to use this feature.
                Run 'pip install graphviz""")

        self.graph = graphviz.Digraph(format='png')
        self.nodes = set()
        self.module_name = module_name
        self.module = module
        self.inspect_settings(dspy.settings)
        self.add_module(self.module_name, self.module)

    def inspect_settings(self, settings):
        """Check for the existence and configuration of LM and RM and add them to the graph."""
        components = {'lm': settings.lm, 'rm': settings.rm}
        for component_name, component in components.items():
            if component:
                details = {attr: getattr(component, attr) for attr in dir(component)
                           if not attr.startswith('_') and not callable(getattr(component, attr))}
                component_details = f"{component_name.upper()} Details: " + ', '.join(f"{k}: {v}" for k, v in
                self.graph.node(component_name, label=component_details, shape='box')
                self.nodes.add(component_name)

    def add_module(self, module_name, module):
        """Add a module to the graph"""

        module_type = type(module)

        if 'dspy.predict' in str(module_type):
            module_name = self.generate_module_name(module_name, module_type)
            self.process_submodule(module_name, module)
        else:
            self.process_submodules(module_name, module)

    def generate_module_name(self, base_name, module_type):  # noqa: ANN201
        """ Generate a module name based on the module type"""
```

```
from .aggregation import majority
from .chain_of_thought import ChainOfThought
from .chain_of_thought_with_hint import ChainOfThoughtWithHint
from .knn import KNN
from .multi_chain_comparison import MultiChainComparison
from .predict import Predict
from .program_of_thought import ProgramOfThought
from .react import ReAct
from .retry import Retry
```

```
import logging
import random
from typing import List, Optional

from pydantic import BaseModel

import dspy


class DescriptionSignature(dspy.Signature):
    field_name = dspy.InputField(desc="name of a field")
    example = dspy.InputField(desc="an example value for the field")
    description = dspy.OutputField(desc="a short text only description of what the field contains")


class SyntheticDataGenerator:
    def __init__(self, schema_class: Optional[BaseModel] = None, examples: Optional[List[dspy.Exa
        self.schema_class = schema_class
        self.examples = examples

    def generate(self, sample_size: int) -> List[dspy.Example]:
        """Generate synthetic examples.

        Args:
            sample_size (int): number of examples to generate
        Raises:
            ValueError: either a schema_class or examples should be provided
        Returns:
            List[dspy.Example]: list of synthetic examples generated
        """
        if not self.schema_class and not self.examples:
            raise ValueError("Either a schema_class or examples must be provided.")
        if self.examples and len(self.examples) >= sample_size:
            logging.info("No additional data generation needed.")
            return self.examples[:sample_size]

        additional_samples_needed = sample_size - (len(self.examples) if self.examples else 0)
        generated_examples = self._generate_additional_examples(additional_samples_needed)

        return self.examples + generated_examples if self.examples else generated_examples

    def _define_or_infer_fields(self):
        """Define fields to generate if a schema class is provided.
        Infer fields to generate if an inital sample of examples is provided.

        Returns:
            dict: dictionary of fields to generate
```

```
import random
from collections.abc import Mapping
from typing import List, Optional, Union

from datasets import Dataset
from rich import print as rprint
from tqdm import tqdm, trange

import dspy

from .config import SynthesizerArguments
from .instruction_suffixes import (
    INPUT_GENERATION_TASK_WITH_EXAMPLES_SUFFIX,
    INPUT_GENERATION_TASK_WITH_FEEDBACK_SUFFIX,
)
from .signatures import (
    ExplainTask,
    GenerateFieldDescription,
    GenerateInputFieldsData,
    GenerateOutputFieldsData,
    GetFeedbackOnGeneration,
    UnderstandTask,
    UpdateTaskDescriptionBasedOnFeedback,
)
from .utils import format_examples

__all__ = [
    "Synthesizer",
    "SynthesizerArguments",
]

class Synthesizer:
    def __init__(self, config: SynthesizerArguments):
        self.config = config
        self.input_lm = config.input_lm_model or dspy.settings.lm
        self.output_lm = config.output_lm_model or dspy.settings.lm

        self.explain_task = dspy.Predict(ExplainTask)
        self.understand_task = dspy.Predict(UnderstandTask)
        self.get_feedback_on_generation = dspy.Predict(GetFeedbackOnGeneration)
        self.generate_field_description = dspy.Predict(GenerateFieldDescription)
        self.update_task_description = dspy.Predict(UpdateTaskDescriptionBasedOnFeedback)

        self.generate_input_data = GenerateInputFieldsData
        self.generate_output_data = GenerateOutputFieldsData

    def _gather_feedback(self, examples: dspy.Example) -> str:
```

```
import inspect
import logging
import math
import os
import random
import shutil
import sys

import numpy as np

try:
    from IPython.core.magics.code import extract_symbols
except ImportError:
    # Won't be able to read code from juptyer notebooks
    extract_symbols = None

import dspy
from dspy.predict.parameter import Parameter
from dspy.teleprompt.bootstrap import BootstrapFewShot, LabeledFewShot

"""
This file consists of helper functions for our variety of optimizers.
"""

### OPTIMIZER TRAINING UTILS ###


def create_minibatch(trainset, batch_size=50):
    """Create a minibatch from the trainset."""

    # Ensure batch_size isn't larger than the size of the dataset
    batch_size = min(batch_size, len(trainset))

    # Randomly sample indices for the mini-batch
    sampled_indices = random.sample(range(len(trainset)), batch_size)

    # Create the mini-batch using the sampled indices
    minibatch = [trainset[i] for i in sampled_indices]

    return minibatch


def eval_candidate_program(batch_size, trainset, candidate_program, evaluate):
    """Evaluate a candidate program on the trainset, using the specified batch size."""
    # Evaluate on the full trainset
    if batch_size >= len(trainset):
        score = evaluate(candidate_program, devset=trainset, display_table=0)
```

```
from .aggregation import majority
from .chain_of_thought import ChainOfThought
from .chain_of_thought_with_hint import ChainOfThoughtWithHint
from .knn import KNN
from .multi_chain_comparison import MultiChainComparison
from .predict import Predict
from .program_of_thought import ProgramOfThought
from .react import ReAct
from .retry import Retry
```

```
from typing import Any, Optional

from pydantic import BaseModel, model_validator


class SynthesizerArguments(BaseModel):
    feedback_mode: Optional[str] = None
    num_example_for_feedback: Optional[int] = None

    input_lm_model: Optional[Any] = None
    output_lm_model: Optional[Any] = None
    output_teacher_module: Optional[Any] = None

    num_example_for_optim: Optional[int] = None

    @model_validator(mode='after')
    def validate_feedback_mode(self):
        if self.feedback_mode and self.feedback_mode not in ["human", "llm"]:
            raise ValueError("Feedback mode should be either 'human' or 'llm'.")

        if self.feedback_mode and not self.num_example_for_feedback:
            raise ValueError("Number of examples for feedback is required when feedback mode is pro

        return self
```

```
INPUT_GENERATION_TASK_WITH_EXAMPLES_SUFFIX = """\n\nI'll also be providing you some

INPUT_GENERATION_TASK_WITH_FEEDBACK_SUFFIX = "\n\nAdditionally, I'll be providing you
```

```
import dspy

from .utils import format_examples


class UnderstandTask(dspy.Signature):
    """I'll be providing you a task description. Your task is to prepare a concise, comprehensible sum

    task_description = dspy.InputField(
        prefix="Task Description:",
        desc="Description of the task.",
    )
    explanation = dspy.OutputField(
        prefix="Task Description:",
        desc="Explanation of the task.",
    )

class ExplainTask(dspy.Signature):
    """Analyze the provided set of datapoints carefully, and prepare a concise, comprehensible summ

    examples = dspy.InputField(
        prefix="Examples Datapoints:",
        desc="List of datapoints to analyze and explain the task.",
        format=format_examples,
    )
    explanation = dspy.OutputField(
        prefix="Task Description:",
        desc="Explanation of the task.",
    )

class UpdateTaskDescriptionBasedOnFeedback(dspy.Signature):
    """Update the task description based on the feedback provided. Ensure that the revised task des

    task_description = dspy.InputField(
        prefix="Task Description:",
        desc="Description of the task.",
    )
    feedback = dspy.InputField(
        prefix="Feedback:",
        desc="Feedback on the task description.",
    )
    updated_task_description = dspy.OutputField(
        prefix="Task Description:",
        desc="Updated description of the task.",
    )

class GetFeedbackOnGeneration(dspy.Signature):
```

```
from dspy.primitives.example import Example


class Prediction(Example):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

        del self._demos
        del self._input_keys

        self._completions = None

    @classmethod
    def from_completions(cls, list_or_dict, signature=None):
        obj = cls()
        obj._completions = Completions(list_or_dict, signature=signature)
        obj._store = {k: v[0] for k, v in obj._completions.items()}

        return obj

    def __repr__(self):
        store_repr = ",\n    ".join(f"{k}={repr(v)}" for k, v in self._store.items())

        if self._completions is None or len(self._completions) == 1:
            return f"Prediction(\n    {store_repr}\n)"

        num_completions = len(self._completions)
        return f"Prediction(\n    {store_repr},\n    completions=Completions(...)\n) ({num_completions-1

    def __str__(self):
        return self.__repr__()

    @property
    def completions(self):
        return self._completions


class Completions:
    def __init__(self, list_or_dict, signature=None):
        self.signature = signature

        if isinstance(list_or_dict, list):
            kwargs = {}
            for arg in list_or_dict:
                for k, v in arg.items():
                    kwargs.setdefault(k, []).append(v)
        else:
```

"""
TODO: If we want to have Prediction::{**keys, completions, box} where box.{key} will behave as a v
the completions internally.

The main thing left is to determine the semantic (and then implement them) for applying operations

If we have a string (query) and completions (5 queries), and we modify the string, what happens to

- Option 1: We modify the string and the (other) completions are not affected.
- Option 2: We modify the string and the (other) completions too.
- Option 3: We modify the string and the (other) completions are deleted.

Option 2 seems most reasonable, but it depends on what metadata the box is going to store.

It seems that a box fundamentally has two functions then:
- Store a value and its "alternatives" (and hence allow transparent application over operations on va
    - But not all operations can/should be applied on all as a map.
    - I guess mainly "give me a string or list or dict or tuple or int or float" has to commit to a value.
    - There also needs to be a .item().
- Potentially track the "source" of the value (i.e., the predictor that generated it, and its inputs)
- Give the value (eventually) to something that will consume the main value (implicitly or explicitly) o

It might be wise to make this responsible for a smaller scope for now:

- Just one string (and its alternatives).
- No source tracking.
- Allow operations on the string to map over the alternatives.
- Seamless extraction at code boundaries.
    - Basically, code will either treat this as string implicitly
      (and hence only know about the one value, and on best effort basis we update the alternatives
    - Or code will explicitly work with the string or explicitly work with the full set of alternatives.

- By default, all programs (and their sub-programs) will be running inside a context in which preserv
- But outside the program, once we see that none of the parent contexts have preserve_boxes=Tru
  unpack all boxes before returning to user.

Okay, so we'll have predictors return a `pred` in which `pred.query` is a box.

You'd usually do one of:

Things that just give you one string
  1- Print `pred.query` or save it in a dict somewhere or a file somewhere.
  2- Call `pred.query.item()` to get the string explicitly.
  3- Modifications in freeform Python.
    - Modify it by calling `pred.query = 'new query'` altogether.
    - Modify it by doing `pred.query += 'new query'` or templating `f'{pred.query} new query'`.
    - Other modifications are not allowed on strings (e.g., `pred.query[0] = 'a'` or `pred.query[0] +=

```
from .aggregation import majority
from .chain_of_thought import ChainOfThought
from .chain_of_thought_with_hint import ChainOfThoughtWithHint
from .knn import KNN
from .multi_chain_comparison import MultiChainComparison
from .predict import Predict
from .program_of_thought import ProgramOfThought
from .react import ReAct
from .retry import Retry
```

```
# ========== Copyright 2023 @ CAMEL-AI.org. All Rights Reserved. ==========
# Licensed under the Apache License, Version 2.0 (the ·License·);
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an ·AS IS· BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
# ========== Copyright 2023 @ CAMEL-AI.org. All Rights Reserved. ==========
import ast
import builtins
import difflib
import importlib
import re
import typing
from collections.abc import Mapping
from typing import (
    Any,
    Dict,
    List,
    Optional,
    Set,
    Tuple,
)


class InterpreterError(ValueError):
    r"""An error raised when the interpreter cannot evaluate a Python
    expression, due to syntax error or unsupported operations.
    """

    pass


class PythonInterpreter:
    r"""A customized python interpreter to control the execution of
    LLM-generated codes. The interpreter makes sure the code can only execute
    functions given in action space and import white list. It also supports
    fuzzy variable matching to receive uncertain input variable name.

    .. highlight:: none

    This class is adapted from the Camel adaptation https://github.com/camel-ai/
```

```
class Example:
    def __init__(self, base=None, **kwargs):
        # Internal storage and other attributes
        self._store = {}
        self._demos = []
        self._input_keys = None

        # Initialize from a base Example if provided
        if base and isinstance(base, type(self)):
            self._store = base._store.copy()

        # Initialize from a dict if provided
        elif base and isinstance(base, dict):
            self._store = base.copy()

        # Update with provided kwargs
        self._store.update(kwargs)

    def __getattr__(self, key):
        if key.startswith("__") and key.endswith("__"):
            raise AttributeError
        if key in self._store:
            return self._store[key]
        raise AttributeError(f"'{type(self).__name__}' object has no attribute '{key}'")

    def __setattr__(self, key, value):
        if key.startswith("_") or key in dir(self.__class__):
            super().__setattr__(key, value)
        else:
            self._store[key] = value

    def __getitem__(self, key):
        return self._store[key]

    def __setitem__(self, key, value):
        self._store[key] = value

    def __delitem__(self, key):
        del self._store[key]

    def __contains__(self, key):
        return key in self._store

    def __len__(self):
        return len([k for k in self._store if not k.startswith("dspy_")])

    def __repr__(self):
```

```
import copy
from collections import deque
from collections.abc import Generator

import ujson

# NOTE: Note: It's important (temporary decision) to maintain named_parameters that's different in
# named_sub_modules for the time being.

class BaseModule:
    def __init__(self):
        pass

    def named_parameters(self):
        """
        Unlike PyTorch, handles (non-recursive) lists of parameters too.
        """

        import dspy
        from dspy.predict.parameter import Parameter

        visited = set()
        named_parameters = []

        def add_parameter(param_name, param_value):
            if isinstance(param_value, Parameter) and id(param_value) not in visited:
                visited.add(id(param_value))
                named_parameters.append((param_name, param_value))

        for name, value in self.__dict__.items():
            if isinstance(value, Parameter):
                add_parameter(name, value)

            elif isinstance(value, dspy.Module):
                # When a sub-module is pre-compiled, keep it frozen.
                if not getattr(value, "_compiled", False):
                    for sub_name, param in value.named_parameters():
                        add_parameter(f"{name}.{sub_name}", param)

            elif isinstance(value, (list, tuple)):
                for idx, item in enumerate(value):
                    add_parameter(f"{name}[{idx}]", item)

            elif isinstance(value, dict):
                for key, item in value.items():
                    add_parameter(f"{name}['{key}']", item)
```

```
import inspect
import uuid
from typing import Any

import dsp
import dspy

###################### Assertion Helpers ######################


def _build_error_msg(feedback_msgs):
    """Build an error message from a list of feedback messages."""
    return "\n".join([msg for msg in feedback_msgs])


###################### Assertion Exceptions ######################


class DSPyAssertionError(AssertionError):
    """Custom exception raised when a DSPy `Assert` fails."""

    def __init__(
        self,
        id: str,
        msg: str,
        target_module: Any = None,
        state: Any = None,
        is_metric: bool = False,
    ) -> None:
        super().__init__(msg)
        self.id = id
        self.msg = msg
        self.target_module = target_module
        self.state = state
        self.is_metric = is_metric


class DSPySuggestionError(AssertionError):
    """Custom exception raised when a DSPy `Suggest` fails."""

    def __init__(
        self,
        id: str,
        msg: str,
        target_module: Any = None,
        state: Any = None,
        is_metric: bool = False,
```

```
import re

from dspy.primitives.assertions import *
from dspy.primitives.module import BaseModule


class ProgramMeta(type):
    pass
    # def __call__(cls, *args, **kwargs):
    #     obj = super(ProgramMeta, cls).__call__(*args, **kwargs)

    #     if issubclass(cls, Program) and not getattr(obj, "_program_init_called", False):
    #         obj._base_init()
    #         obj._program_init_called = True
    #     return obj


class Module(BaseModule, metaclass=ProgramMeta):
    def _base_init(self):
        self._compiled = False

    def __init__(self):
        self._compiled = False

    def __call__(self, *args, **kwargs):
        return self.forward(*args, **kwargs)

    def named_predictors(self):
        from dspy.predict.predict import Predict

        return [(name, param) for name, param in self.named_parameters() if isinstance(param, Predi

    def predictors(self):
        return [param for _, param in self.named_predictors()]

    def __repr__(self):
        s = []

        for name, param in self.named_predictors():
            s.append(f"{name} = {param}")

        return "\n".join(s)

    def map_named_predictors(self, func):
        """Applies a function to all named predictors."""
        for name, predictor in self.named_predictors():
            set_attribute_by_name(self, name, func(predictor))
```

```
from .aggregation import majority
from .chain_of_thought import ChainOfThought
from .chain_of_thought_with_hint import ChainOfThoughtWithHint
from .knn import KNN
from .multi_chain_comparison import MultiChainComparison
from .predict import Predict
from .program_of_thought import ProgramOfThought
from .react import ReAct
from .retry import Retry
```

```
import dspy


class AnswerCorrectnessSignature(dspy.Signature):
    """Verify that the predicted answer matches the gold answer."""

    question = dspy.InputField()
    gold_answer = dspy.InputField(desc="correct answer for question")
    predicted_answer = dspy.InputField(desc="predicted answer for question")
    is_correct = dspy.OutputField(desc='True or False')

class AnswerCorrectness(dspy.Module):
    def __init__(self):
        super().__init__()
        self.evaluate_correctness = dspy.ChainOfThought(AnswerCorrectnessSignature)

    def forward(self, question, gold_answer, predicted_answer):
        return self.evaluate_correctness(question=question, gold_answer=gold_answer, predicted_an


class AnswerFaithfulnessSignature(dspy.Signature):
    """Verify that the predicted answer is based on the provided context."""

    context = dspy.InputField(desc="relevant facts for producing answer")
    question = dspy.InputField()
    answer = dspy.InputField(desc="often between 1 and 5 words")
    is_faithful = dspy.OutputField(desc='True or False')

class AnswerFaithfulness(dspy.Module):
    def __init__(self):
        super().__init__()
        self.evaluate_faithfulness = dspy.ChainOfThought(AnswerFaithfulnessSignature)

    def forward(self, context, question, answer):
        return self.evaluate_faithfulness(context=context, question=question, answer=answer)
```

```
import contextlib
import signal
import sys
import threading
import types

import pandas as pd
import tqdm
from tqdm.contrib.logging import logging_redirect_tqdm

import dspy

try:
    from IPython.display import HTML
    from IPython.display import display as ipython_display
except ImportError:
    ipython_display = print

    def HTML(x) -> str:  # noqa: N802
        return x


from concurrent.futures import ThreadPoolExecutor, as_completed

# TODO: Counting failures and having a max_failure count. When that is exceeded (also just at the
# we print the number of failures, the first N examples that failed, and the first N exceptions raised.


class Evaluate:
    def __init__(
        self,
        *,
        devset,
        metric=None,
        num_threads=1,
        display_progress=False,
        display_table=False,
        max_errors=5,
        return_outputs=False,
        **_kwargs,
    ):
        self.devset = devset
        self.metric = metric
        self.num_threads = num_threads
        self.display_progress = display_progress
        self.display_table = display_table
        self.max_errors = max_errors
```

```
# TODO: This should move internally. Same for passage_match. dspy.metrics.answer_exact_matc

import dsp


def answer_exact_match(example, pred, trace=None, frac=1.0):
    assert(type(example.answer) is str or type(example.answer) is list)

    if type(example.answer) is str:
        return dsp.answer_match(pred.answer, [example.answer], frac=frac)
    else: # type(example.answer) is list
        return dsp.answer_match(pred.answer, example.answer, frac=frac)

answer_exact_match_str = dsp.answer_match

def answer_passage_match(example, pred, trace=None):
    assert(type(example.answer) is str or type(example.answer) is list)

    if type(example.answer) is str:
        return dsp.passage_match(pred.context, [example.answer])
    else: # type(example.answer) is list
        return dsp.passage_match(pred.context, example.answer)
```

```
from .aggregation import majority
from .chain_of_thought import ChainOfThought
from .chain_of_thought_with_hint import ChainOfThoughtWithHint
from .knn import KNN
from .multi_chain_comparison import MultiChainComparison
from .predict import Predict
from .program_of_thought import ProgramOfThought
from .react import ReAct
from .retry import Retry
```

```
from .aggregation import majority
from .chain_of_thought import ChainOfThought
from .chain_of_thought_with_hint import ChainOfThoughtWithHint
from .knn import KNN
from .multi_chain_comparison import MultiChainComparison
from .predict import Predict
from .program_of_thought import ProgramOfThought
from .react import ReAct
from .retry import Retry
```

```
import logging
import os
import sys
import typing as t

import structlog

logger = structlog.get_logger()


class LogSettings:
    def __init__(self, output_type: str, method: str, file_name: t.Optional[str]) -> None:
        self.output_type = output_type
        self.method = method
        self.file_name = file_name
        self._configure_structlog()

    def _configure_structlog(self):
        if self.output_type == "str":
            renderer = structlog.dev.ConsoleRenderer()
        else:
            renderer = structlog.processors.JSONRenderer()

        structlog.configure(
            processors=[
                structlog.stdlib.add_logger_name,
                structlog.stdlib.add_log_level,
                structlog.processors.CallsiteParameterAdder(
                    {
                        structlog.processors.CallsiteParameter.FILENAME,
                        structlog.processors.CallsiteParameter.LINENO,
                    },
                ),
                structlog.processors.TimeStamper(fmt="iso"),
                structlog.processors.StackInfoRenderer(),
                structlog.processors.format_exc_info,
                structlog.processors.UnicodeDecoder(),
                renderer,
            ],
            logger_factory=structlog.stdlib.LoggerFactory(),
            wrapper_class=structlog.stdlib.BoundLogger,
        )

    def set_log_output(
        self,
        method: t.Optional[str] = None,
        file_name: t.Optional[str] = None,
```

```
import random
import re
from typing import Union

import numpy as np

from dsp.modules import LM
from dsp.utils.utils import dotdict


class DummyLM(LM):
    """Dummy language model for unit testing purposes."""

    def __init__(self, answers: Union[list[str], dict[str, str]], follow_examples: bool = False):
        """Initializes the dummy language model.

        Parameters:
        - answers: A list of strings or a dictionary with string keys and values.
        - follow_examples: If True, and the prompt contains an example exactly equal to the prompt,
                           the dummy model will return the next string in the list for each request.
        If a list is provided, the dummy model will return the next string in the list for each request.
        If a dictionary is provided, the dummy model will return the value corresponding to the key tha
        """
        super().__init__("dummy-model")
        self.provider = "dummy"
        self.answers = answers
        self.follow_examples = follow_examples

    def basic_request(self, prompt, n=1, **kwargs) -> dict[str, list[dict[str, str]]]:
        """Generates a dummy response based on the prompt."""
        dummy_response = {"choices": []}
        for _ in range(n):
            answer = None

            if self.follow_examples:
                prefix = prompt.split("\n")[-1]
                _instructions, _format, *examples, _output = prompt.split("\n---\n")
                examples_str = "\n".join(examples)
                possible_answers = re.findall(prefix + r"\s*(.*)", examples_str)
                if possible_answers:
                    # We take the last answer, as the first one is just from
                    # the "Follow the following format" section.
                    answer = possible_answers[-1]
                    print(f"DummyLM got found previous example for {prefix} with value {answer=}")
                else:
                    print(f"DummyLM couldn't find previous example for {prefix=}")
```

```
import random
import uuid

from dsp.utils import dotdict
from dspy import Example


class Dataset:
    def __init__(self, train_seed=0, train_size=None, eval_seed=0, dev_size=None, test_size=None
        self.train_size = train_size
        self.train_seed = train_seed
        self.dev_size = dev_size
        self.dev_seed = eval_seed
        self.test_size = test_size
        self.test_seed = eval_seed
        self.input_keys = input_keys

        self.do_shuffle = True

        self.name = self.__class__.__name__

    def reset_seeds(self, train_seed=None, train_size=None, eval_seed=None, dev_size=None, tes
        self.train_size = train_size if train_size is not None else self.train_size
        self.train_seed = train_seed if train_seed is not None else self.train_seed
        self.dev_size = dev_size if dev_size is not None else self.dev_size
        self.dev_seed = eval_seed if eval_seed is not None else self.dev_seed
        self.test_size = test_size if test_size is not None else self.test_size
        self.test_seed = eval_seed if eval_seed is not None else self.test_seed

        if hasattr(self, '_train_'):
            del self._train_

        if hasattr(self, '_dev_'):
            del self._dev_

        if hasattr(self, '_test_'):
            del self._test_

    @property
    def train(self):
        if not hasattr(self, '_train_'):
            self._train_ = self._shuffle_and_sample('train', self._train, self.train_size, self.train_seed)

        return self._train_

    @property
    def dev(self):
```

```
from .aggregation import majority
from .chain_of_thought import ChainOfThought
from .chain_of_thought_with_hint import ChainOfThoughtWithHint
from .knn import KNN
from .multi_chain_comparison import MultiChainComparison
from .predict import Predict
from .program_of_thought import ProgramOfThought
from .react import ReAct
from .retry import Retry
```

```
import random

from datasets import load_dataset

from dspy.datasets.dataset import Dataset


class HotPotQA(Dataset):
    def __init__(self, *args, only_hard_examples=True, keep_details='dev_titles', unofficial_dev=Tru
        super().__init__(*args, **kwargs)
        assert only_hard_examples, "Care must be taken when adding support for easy examples." \
                                   "Dev must be all hard to match official dev, but training can be flexible."

        hf_official_train = load_dataset("hotpot_qa", 'fullwiki', split='train', trust_remote_code=True)
        hf_official_dev = load_dataset("hotpot_qa", 'fullwiki', split='validation', trust_remote_code=True

        official_train = []
        for raw_example in hf_official_train:
            if raw_example['level'] == 'hard':
                if keep_details is True:
                    keys = ['id', 'question', 'answer', 'type', 'supporting_facts', 'context']
                elif keep_details == 'dev_titles':
                    keys = ['question', 'answer', 'supporting_facts']
                else:
                    keys = ['question', 'answer']

                example = {k: raw_example[k] for k in keys}

                if 'supporting_facts' in example:
                    example['gold_titles'] = set(example['supporting_facts']['title'])
                    del example['supporting_facts']

                official_train.append(example)

        rng = random.Random(0)
        rng.shuffle(official_train)

        self._train = official_train[:len(official_train)*75//100]

        if unofficial_dev:
            self._dev = official_train[len(official_train)*75//100:]
        else:
            self._dev = None

        for example in self._train:
            if keep_details == 'dev_titles':
                del example['gold_titles']
```

```
import random

import tqdm
from datasets import load_dataset


class GSM8K:
    def __init__(self) -> None:
        super().__init__()
        self.do_shuffle = False

        dataset = load_dataset("gsm8k", 'main')

        hf_official_train = dataset['train']
        hf_official_test = dataset['test']
        official_train = []
        official_test = []

        for example in tqdm.tqdm(hf_official_train):
            question = example['question']

            answer = example['answer'].strip().split()
            assert answer[-2] == '####'

            gold_reasoning = ' '.join(answer[:-2])
            answer = str(int(answer[-1].replace(',', '')))

            official_train.append(dict(question=question, gold_reasoning=gold_reasoning, answer=answ

        for example in tqdm.tqdm(hf_official_test):
            question = example['question']

            answer = example['answer'].strip().split()
            assert answer[-2] == '####'

            gold_reasoning = ' '.join(answer[:-2])
            answer = str(int(answer[-1].replace(',', '')))

            official_test.append(dict(question=question, gold_reasoning=gold_reasoning, answer=answ

        rng = random.Random(0)
        rng.shuffle(official_train)

        rng = random.Random(0)
        rng.shuffle(official_test)

        trainset = official_train[:200]
```

```
import random

from dspy.datasets.dataset import Dataset

### A bunch of colors, originally from matplotlib
all_colors = ['alice blue', 'dodger blue', 'light sky blue', 'deep sky blue', 'sky blue', 'steel blue', 'light st

class Colors(Dataset):
    def __init__(self, sort_by_suffix=True, *args, **kwargs) -> None:
        super().__init__(*args, **kwargs)

        self.sort_by_suffix = sort_by_suffix
        colors = self.sorted_by_suffix(all_colors)

        train_size = int(len(colors) * 0.6) # chosen to ensure that similar colors aren't repeated betwee
        train_colors, dev_colors = colors[:train_size], colors[train_size:]

        self._train = [dict(color=color) for color in train_colors]
        self._dev = [dict(color=color) for color in dev_colors]

        random.Random(0).shuffle(self._train)
        random.Random(0).shuffle(self._dev)

    def sorted_by_suffix(self, colors):
        if not self.sort_by_suffix:
            return colors

        if isinstance(colors[0], str):
            sorted_colors = sorted(colors, key=lambda x: x[::-1])
        else:
            sorted_colors = sorted(colors, key=lambda x: x['color'][::-1])

        return sorted_colors
```

```
import random
from collections.abc import Mapping
from typing import List, Tuple, Union

from datasets import load_dataset

import dspy
from dspy.datasets.dataset import Dataset


class DataLoader(Dataset):
    def __init__(self,):
        pass

    def from_huggingface(
        self,
        dataset_name: str,
        *args,
        input_keys: Tuple[str] = (),
        fields: Tuple[str] = None,
        **kwargs,
    ) -> Union[Mapping[str, List[dspy.Example]], List[dspy.Example]]:
        if fields and not isinstance(fields, tuple):
            raise ValueError("Invalid fields provided. Please provide a tuple of fields.")

        if not isinstance(input_keys, tuple):
            raise ValueError("Invalid input keys provided. Please provide a tuple of input keys.")

        dataset = load_dataset(dataset_name, *args, **kwargs)

        if isinstance(dataset, list) and isinstance(kwargs["split"], list):
            dataset = {split_name:dataset[idx] for idx, split_name in enumerate(kwargs["split"])}

        try:
            returned_split = {}
            for split_name in dataset.keys():
                if fields:
                    returned_split[split_name] = [dspy.Example({field:row[field] for field in fields}).with_inpu
                else:
                    returned_split[split_name] = [dspy.Example({field:row[field] for field in row.keys()}).with

            return returned_split
        except AttributeError:
            if fields:
                return [dspy.Example({field:row[field] for field in fields}).with_inputs(*input_keys) for row i
            else:
                return [dspy.Example({field:row[field] for field in row.keys()}).with_inputs(*input_keys) for
```

```
import inspect
import json
import typing
from typing import Annotated, Callable, List, Tuple, Union  # noqa: UP035

import pydantic
import ujson
from pydantic.fields import FieldInfo

import dspy
from dsp.templates import passages2text
from dspy.primitives.prediction import Prediction
from dspy.signatures.signature import ensure_signature, make_signature


def predictor(*args: tuple, **kwargs) -> Callable[..., dspy.Module]:
    def _predictor(func) -> dspy.Module:
        """Decorator that creates a predictor module based on the provided function."""
        signature = _func_to_signature(func)
        *_, output_key = signature.output_fields.keys()
        return _StripOutput(TypedPredictor(signature, **kwargs), output_key)

    # if we have only a single callable argument, the decorator was invoked with no key word argum
    #  so we just return the wrapped function
    if len(args) == 1 and callable(args[0]) and len(kwargs) == 0:
        return _predictor(args[0])
    return _predictor


def cot(*args: tuple, **kwargs) -> Callable[..., dspy.Module]:
    def _cot(func) -> dspy.Module:
        """Decorator that creates a chain of thought module based on the provided function."""
        signature = _func_to_signature(func)
        *_, output_key = signature.output_fields.keys()
        return _StripOutput(TypedChainOfThought(signature, **kwargs), output_key)

    # if we have only a single callable argument, the decorator was invoked with no key word argum
    #  so we just return the wrapped function
    if len(args) == 1 and callable(args[0]) and len(kwargs) == 0:
        return _cot(args[0])
    return _cot


class _StripOutput(dspy.Module):
    def __init__(self, predictor, output_key):
        super().__init__()
        self.predictor = predictor
```

```
from .aggregation import majority
from .chain_of_thought import ChainOfThought
from .chain_of_thought_with_hint import ChainOfThoughtWithHint
from .knn import KNN
from .multi_chain_comparison import MultiChainComparison
from .predict import Predict
from .program_of_thought import ProgramOfThought
from .react import ReAct
from .retry import Retry
```

```
import logging
from typing import Optional

import dspy

try:
    from llama_index.core.base.base_retriever import BaseRetriever
except ImportError:
    err = "The 'llama_index' package is required to use LlamaIndexRM. Install it with 'pip install llama
    raise ImportError(err) from None

NO_TOP_K_WARNING = "The underlying LlamaIndex retriever does not support top k retrieval. Ig


class LlamaIndexRM(dspy.Retrieve):
    """Implements a retriever which wraps over a LlamaIndex retriever.

    This is done to bridge LlamaIndex and DSPy and allow the various retrieval
    abstractions in LlamaIndex to be used in DSPy.

    To-do (maybe):
        - Async support (DSPy lacks this entirely it seems, so not a priority until the rest of the repo ca
        - Text/video retrieval (Available in LI, not sure if this will be a priority in DSPy)

    Args:
        retriever (BaseRetriever): A LlamaIndex retriever object - text based only
        k (int): Optional; the number of examples to retrieve (similarity_top_k)

        If the underlying LI retriever does not have the property similarity_top_k, k will be ignored.

    Returns:
        DSPy RM Object - this is a retriever object that can be used in DSPy
    """

    retriever: BaseRetriever

    def __init__(
        self,
        retriever: BaseRetriever,
        k: Optional[int] = None,
    ):
        self.retriever = retriever

        if k:
            self.k = k

    @property
```

```
import os
import warnings
from typing import Any, Literal, Optional, Union

import requests

import dspy
from dsp.utils import dotdict


class YouRM(dspy.Retrieve):
    """Retriever for You.com's Search and News API.

    [API reference](https://documentation.you.com/api-reference/)

    Args:
        ydc_api_key: you.com API key, if `YDC_API_KEY` is not set in the environment
        k: If ``endpoint="search"``, the max snippets to return per search hit.
            If ``endpoint="news"``, the max articles to return.
        endpoint: you.com endpoints
        num_web_results: The max number of web results to return, must be under 20
        safesearch: Safesearch settings, one of "off", "moderate", "strict", defaults to moderate
        country: Country code, ex: 'US' for United States, see API reference for more info
        search_lang: (News API) Language codes, ex: 'en' for English, see API reference for more inf
        ui_lang: (News API) User interface language for the response, ex: 'en' for English.
                        See API reference for more info
        spellcheck: (News API) Whether to spell check query or not, defaults to True
    """

    def __init__(
        self,
        ydc_api_key: Optional[str] = None,
        k: int = 3,
        endpoint: Literal["search", "news"] = "search",
        num_web_results: Optional[int] = None,
        safesearch: Optional[Literal["off", "moderate", "strict"]] = None,
        country: Optional[str] = None,
        search_lang: Optional[str] = None,
        ui_lang: Optional[str] = None,
        spellcheck: Optional[bool] = None,
    ):
        super().__init__(k=k)

        # Data validation
        if not ydc_api_key and not os.environ.get("YDC_API_KEY"):
            raise RuntimeError('You must supply `ydc_api_key` or set environment variable "YDC_API_
```

```
"""
Retriever model for Pinecone
Author: Dhar Rawal (@drawal1)
"""

from typing import List, Optional, Union

import backoff

import dspy
from dsp.utils import dotdict

try:
    import pinecone
except ImportError:
    pinecone = None

if pinecone is None:
    raise ImportError(
        "The pinecone library is required to use PineconeRM. Install it with `pip install dspy-ai[pinecon
    )

import openai

try:
    OPENAI_LEGACY = int(openai.version.__version__[0]) == 0
except Exception:
    OPENAI_LEGACY = True

try:
    import openai.error
    ERRORS = (openai.error.RateLimitError, openai.error.ServiceUnavailableError, openai.error.AP
except Exception:
    ERRORS = (openai.RateLimitError, openai.APIError)

class PineconeRM(dspy.Retrieve):
    """
    A retrieval module that uses Pinecone to return the top passages for a given query.

    Assumes that the Pinecone index has been created and populated with the following metadata:
        - text: The text of the passage

    Args:
        pinecone_index_name (str): The name of the Pinecone index to query against.
        pinecone_api_key (str, optional): The Pinecone API key. Defaults to None.
        pinecone_env (str, optional): The Pinecone environment. Defaults to None.
        local_embed_model (str, optional): The local embedding model to use. A popular default is "se
```

```
"""
Retriever model for chromadb
"""

from typing import List, Optional, Union

import backoff
import openai

import dspy
from dsp.utils import dotdict

try:
    import openai.error
    ERRORS = (openai.error.RateLimitError, openai.error.ServiceUnavailableError, openai.error.AP
except Exception:
    ERRORS = (openai.RateLimitError, openai.APIError)

try:
    import chromadb
    import chromadb.utils.embedding_functions as ef
    from chromadb.api.types import (
        Embeddable,
        EmbeddingFunction,
    )
    from chromadb.config import Settings
    from chromadb.utils import embedding_functions
except ImportError:
    raise ImportError(
        "The chromadb library is required to use ChromadbRM. Install it with `pip install dspy-ai[chrom
    )


class ChromadbRM(dspy.Retrieve):
    """
    A retrieval module that uses chromadb to return the top passages for a given query.

    Assumes that the chromadb index has been created and populated with the following metadata:
        - documents: The text of the passage

    Args:
        collection_name (str): chromadb collection name
        persist_directory (str): chromadb persist directory
        embedding_function (Optional[EmbeddingFunction[Embeddable]]): Optional function to use to
        k (int, optional): The number of top passages to retrieve. Defaults to 7.
        client(Optional[chromadb.Client]): Optional chromadb client provided by user, default to None
```

```
import json
import os
from collections import defaultdict
from typing import List, Optional, Union

import requests

import dspy
from dsp.utils import dotdict

START_SNIPPET = "<%START%>"
END_SNIPPET = "<%END%>"

def remove_snippet(s: str) -> str:
    return s.replace(START_SNIPPET, "").replace(END_SNIPPET, "")

class VectaraRM(dspy.Retrieve):
    """
    A retrieval module that uses Vectara to return the top passages for a given query.

    Assumes that a Vectara corpora have been created and populated with the following payload:
        - document: The text of the passage

    Args:
        vectara_customer_id (str): Vectara Customer ID. defaults to VECTARA_CUSTOMER_ID envi
        vectara_corpus_id (str): Vectara Corpus ID. defaults to VECTARA_CORPUS_ID environment
        vectara_api_key (str): Vectara API Key. defaults to VECTARA_API_KEY environment variable
        k (int, optional): The default number of top passages to retrieve. Defaults to 3.

    Examples:
        Below is a code snippet that shows how to use Vectara as the default retriver:
        ```python
        from vectara_client import vectaraClient

        llm = dspy.OpenAI(model="gpt-3.5-turbo")
        retriever_model = vectaraRM("<VECTARA_CUSTOMER_ID>", "<VECTARA_CORPUS_ID>",
        dspy.settings.configure(lm=llm, rm=retriever_model)
        ```

        Below is a code snippet that shows how to use Vectara in the forward() function of a module
        ```python
        self.retrieve = vectaraRM("<VECTARA_CUSTOMER_ID>", "<VECTARA_CORPUS_ID>", "<V
        ```
    """

    def __init__(
        self,
```

```
import json
import os
from collections import defaultdict
from typing import List, Union

import requests

import dspy
from dspy.primitives.prediction import Prediction


class DatabricksRM(dspy.Retrieve):
    """
    A retrieval module that uses Databricks Vector Search Endpoint to return the top-k embeddings f

    Args:
        databricks_index_name (str): Databricks vector search index to query
        databricks_endpoint (str): Databricks index endpoint url
        databricks_token (str): Databricks authentication token
        columns (list[str]): Column names to include in response
        filters_json (str, optional): JSON string for query filters
        k (int, optional): Number of top embeddings to retrieve. Defaults to 3.
        docs_id_column_name (str, optional): Column name for retrieved doc_ids to return.
        text_column_name (str, optional): Column name for retrieved text to return.

    Examples:
        Below is a code snippet that shows how to configure Databricks Vector Search endpoints:

        (example adapted from "Databricks: How to create and query a Vector Search Index:
        https://docs.databricks.com/en/generative-ai/create-query-vector-search.html#create-a-vector-

        ```python
        from databricks.vector_search.client import VectorSearchClient

        #Creating Vector Search Client

        client = VectorSearchClient()

        client.create_endpoint(
            name="your_vector_search_endpoint_name",
            endpoint_type="STANDARD"
        )

        #Creating Vector Search Index using Python SDK
        #Example for Direct Vector Access Index

        index = client.create_direct_access_index(
```
```

```
from collections import defaultdict
from typing import Optional, Union

import dspy
from dsp.modules.sentence_vectorizer import BaseSentenceVectorizer, FastEmbedVectorizer
from dsp.utils import dotdict

try:
    from qdrant_client import QdrantClient, models
except ImportError as e:
    raise ImportError(
        "The 'qdrant' extra is required to use QdrantRM. Install it with `pip install dspy-ai[qdrant]`",
    ) from e


class QdrantRM(dspy.Retrieve):
    """A retrieval module that uses Qdrant to return the top passages for a given query.

    Args:
        qdrant_collection_name (str): The name of the Qdrant collection.
        qdrant_client (QdrantClient): An instance of `qdrant_client.QdrantClient`.
        k (int, optional): The default number of top passages to retrieve. Default: 3.
        document_field (str, optional): The key in the Qdrant payload with the content. Default: `"docu
        vectorizer (BaseSentenceVectorizer, optional): An implementation `BaseSentenceVectorizer`.
                                                       Default: `FastEmbedVectorizer`.
        vector_name (str, optional): Name of the vector in the collection. Default: The first available ve

    Examples:
        Below is a code snippet that shows how to use Qdrant as the default retriver:
        ```python
        from qdrant_client import QdrantClient

        llm = dspy.OpenAI(model="gpt-3.5-turbo")
        qdrant_client = QdrantClient()
        retriever_model = QdrantRM("my_collection_name", qdrant_client=qdrant_client)
        dspy.settings.configure(lm=llm, rm=retriever_model)
        ```

        Below is a code snippet that shows how to use Qdrant in the forward() function of a module
        ```python
        self.retrieve = QdrantRM("my_collection_name", qdrant_client=qdrant_client, k=num_passage
        ```
    """

    def __init__(
        self,
        qdrant_collection_name: str,
```

```
from .aggregation import majority
from .chain_of_thought import ChainOfThought
from .chain_of_thought_with_hint import ChainOfThoughtWithHint
from .knn import KNN
from .multi_chain_comparison import MultiChainComparison
from .predict import Predict
from .program_of_thought import ProgramOfThought
from .react import ReAct
from .retry import Retry
```

```
import functools
import os
from typing import List, Optional

import openai

import dspy
from dsp.modules.cache_utils import CacheMemory, NotebookCacheMemory, cache_turn_on
from dsp.utils import dotdict

# Check for necessary libraries and suggest installation if not found.
try:
    import clickhouse_connect
except ImportError:
    raise ImportError(
        "The 'myscale' extra is required to use MyScaleRM. Install it with `pip install dspy-ai[myscale]`
    )

# Verify the compatibility of the OpenAI library version installed.
try:
    major, minor, _ = map(int, openai.__version__.split('.'))
    OPENAI_VERSION_COMPATIBLE = major >= 1 and minor >= 16
except Exception:
    OPENAI_VERSION_COMPATIBLE = False

if not OPENAI_VERSION_COMPATIBLE:
    raise ImportError(
        "An incompatible OpenAI library version is installed. Ensure you have version 1.16.1 or later.",
    )

# Attempt to handle specific OpenAI errors; fallback to general ones if necessary.
try:
    import openai.error
    ERRORS = (openai.error.RateLimitError, openai.error.ServiceUnavailableError, openai.error.AP
except Exception:
    ERRORS = (openai.RateLimitError, openai.APIError)


class MyScaleRM(dspy.Retrieve):
    """
    A retrieval module that uses MyScaleDB to return the top passages for a given query.

    MyScaleDB is a fork of ClickHouse that focuses on vector similarity search and full
    text search.  MyScaleRM is designed to facilitate easy retrieval of information from
    MyScaleDB using embeddings.  It supports embedding generation through either a local
    model or the OpenAI API. This class abstracts away the complexities of connecting to
    MyScaleDB, managing API keys, and processing queries to return semantically
```

```
"""Clarifai as retriver to retrieve hits"""
import os
from concurrent.futures import ThreadPoolExecutor
from typing import List, Optional, Union

import requests

import dspy
from dsp.utils import dotdict

try:
    from clarifai.client.search import Search
except ImportError as err:
    raise ImportError(
        "Clarifai is not installed. Install it using `pip install clarifai`",
    ) from err


class ClarifaiRM(dspy.Retrieve):
    """
    Retrieval module uses clarifai to return the Top K relevant pasages for the given query.
    Assuming that you have ingested the source documents into clarifai App, where it is indexed and

    Args:
        clarifai_user_id (str): Clarifai unique user_id.
        clarfiai_app_id (str): Clarifai App ID, where the documents are stored.
        clarifai_pat (str): Clarifai PAT key.
        k (int): Top K documents to retrieve.

    Examples:
        TODO
    """

    def __init__(
        self,
        clarifai_user_id: str,
        clarfiai_app_id: str,
        clarifai_pat: Optional[str] = None,
        k: int = 3,
    ):
        self.app_id = clarfiai_app_id
        self.user_id = clarifai_user_id
        self.pat = (
            clarifai_pat if clarifai_pat is not None else os.environ["CLARIFAI_PAT"]
        )
        self.k = k
        self.clarifai_search = Search(
```

```
from typing import Optional, Union

import dspy
from dsp.utils.utils import dotdict

try:
    from ragatouille import RAGPretrainedModel
except ImportError:
    raise Exception(
        "You need to install RAGAtouille library, Please use the command: pip install ragatouille",
    )


class RAGatouilleRM(dspy.Retrieve):
    """A retrieval model that uses RAGatouille library to return the top passages for a given query.

    Assumes that you already have an index created with RAGatouille.
    Reference: https://github.com/bclavie/RAGatouille

    Args:
        index_root (str): Folder path where you index is stored.
        index_name (str): Name of the index you want to retrieve from.
        k (int, optional): The default number of passages to retrieve. Defaults to 3.

    Examples:
        Below is a code snippet that shows how to use RAGatouille index as the default retriver:
        ```python
        llm = dspy.OpenAI(model="gpt-3.5-turbo")
        rm = RAGatouilleRM(index_root="ragatouille/colbert/indexes", index_name="my_index")
        dspy.settings.configure(lm=llm, rm=rm)
        ```
    """

    def __init__(self, index_root: str, index_name: str, k: int = 3):
        super().__init__(k=k)
        self.index_path = f"{index_root}/{index_name}"
        try:
            self.model = RAGPretrainedModel.from_index(index_path=self.index_path)
        except FileNotFoundError as e:
            raise FileNotFoundError(f"Index not found: {e}")

    def forward(self, query_or_queries: Union[str, list[str]], k: Optional[int]) -> dspy.Prediction:
        """Search with RAGAtouille based index for self.k top passages for query

        Args:
            query_or_queries (Union[str, List[str]]): The query or queries to search for.
            k (Optional[int]): The number of top passages to retrieve. Defaults to self.k.
```

```
import json
from typing import Optional, Union

import requests
from requests.auth import HTTPBasicAuth

import dspy
from dsp.utils import dotdict


class WatsonDiscoveryRM(dspy.Retrieve):
    """A retrieval module that uses Watson Discovery to return the top passages for a given query.

    Args:
        apikey (str): apikey for authentication purposes,
        url (str): endpoint URL that includes the service instance ID
        version (str): Release date of the version of the API you want to use. Specify dates in YYYY-M
        project_id (str): The Universally Unique Identifier (UUID) of the project.
        collection_ids (list): An array containing the collections on which the search will be executed.
        k (int, optional): The number of top passages to retrieve. Defaults to 5.

    Returns:
        dspy.Prediction: An object containing the retrieved passages.
    """

    def __init__(
        self,
        apikey: str,
        url:str,
        version:str,
        project_id:str,
        collection_ids:list=None,
        k: int = 5,
    ):
        if collection_ids is None:
            collection_ids = []
        self.apikey=apikey
        self.url=url,
        self.version:version
        self.project_id:project_id
        self.collection_ids=collection_ids
        self.k: k
        self.query_url=url + "/v2/projects/" + project_id + "/query?version=" + version

        super().__init__(k=k)

    def forward(self, query_or_queries: Union[str, list[str]], k: Optional[int]= None) -> dspy.Prediction:
```

```
import os
from typing import Any, List, Optional, Union

import backoff
from openai import (
    APITimeoutError,
    InternalServerError,
    OpenAI,
    RateLimitError,
    UnprocessableEntityError,
)

import dspy
from dsp.utils import dotdict

try:
    from neo4j import GraphDatabase
    from neo4j.exceptions import (
        AuthError,
        ServiceUnavailable,
    )
except ImportError:
    raise ImportError(
        "Please install the neo4j package by running `pip install dspy-ai[neo4j]`",
    )


class Embedder:
    def __init__(self, provider: str, model: str):
        if provider == "openai":
            api_key = os.getenv("OPENAI_API_KEY")
            if not api_key:
                raise ValueError("Environment variable OPENAI_API_KEY must be set")
            self.client = OpenAI()
            self.model = model

    @backoff.on_exception(
        backoff.expo,
        (
            APITimeoutError,
            InternalServerError,
            RateLimitError,
            UnprocessableEntityError,
        ),
        max_time=15,
    )
    def __call__(self, queries) -> Any:
```

```
from typing import Optional, Union

import dspy
from dsp.utils import dotdict

try:
    from snowflake.snowpark import Session
    from snowflake.snowpark import functions as snow_fn
    from snowflake.snowpark.functions import col, function, lit
    from snowflake.snowpark.types import VectorType

except ImportError:
    raise ImportError(
        "The snowflake-snowpark-python library is required to use SnowflakeRM. Install it with dspy-a
    )


class SnowflakeRM(dspy.Retrieve):
    """A retrieval module that uses Weaviate to return the top passages for a given query.

    Assumes that a Snowflake table has been created and populated with the following payload:
        - content: The text of the passage

    Args:
        snowflake_credentials: connection parameters for initializing Snowflake client.
        snowflake_table_name (str): The name of the Snowflake table containing document embeddir
        embeddings_field (str): The field in the Snowflake table with the content embeddings
        embeddings_text_field (str): The field in the Snowflake table with the content.
        k (int, optional): The default number of top passages to retrieve. Defaults to 3.
    """

    def __init__(
        self,
        snowflake_table_name: str,
        snowflake_credentials: dict,
        k: int = 3,
        embeddings_field: str = "chunk_vec",
        embeddings_text_field: str = "chunk",
        embeddings_model: str = "e5-base-v2",
    ):
        self.snowflake_table_name = snowflake_table_name
        self.embeddings_field = embeddings_field
        self.embeddings_text_field = embeddings_text_field
        self.embeddings_model = embeddings_model
        self.client = self._init_cortex(credentials=snowflake_credentials)

        super().__init__(k=k)
```

```
import os
from typing import Any, List

import backoff
from openai import (
    APITimeoutError,
    InternalServerError,
    OpenAI,
    RateLimitError,
    UnprocessableEntityError,
)

import dspy

try:
    from pymongo import MongoClient
    from pymongo.errors import (
        ConfigurationError,
        ConnectionFailure,
        InvalidURI,
        OperationFailure,
        ServerSelectionTimeoutError,
    )
except ImportError:
    raise ImportError(
        "Please install the pymongo package by running `pip install dspy-ai[mongodb]`",
    )


def build_vector_search_pipeline(
    index_name: str, query_vector: List[float], num_candidates: int, limit: int,
) -> List[dict[str, Any]]:
    return [
        {
            "$vectorSearch": {
                "index": index_name,
                "path": "embedding",
                "queryVector": query_vector,
                "numCandidates": num_candidates,
                "limit": limit,
            },
        },
        {"$project": {"_id": 0, "text": 1, "score": {"$meta": "vectorSearchScore"}}},
    ]


class Embedder:
```

```
import warnings
from typing import Callable, Optional

import dspy

try:
    import psycopg2
    from pgvector.psycopg2 import register_vector
    from psycopg2 import sql
except ImportError:
    raise ImportError(
        "The 'pgvector' extra is required to use PgVectorRM. Install it with `pip install dspy-ai[pgvector
    )
try:
    import openai
except ImportError:
    warnings.warn(
        "`openai` is not installed. Install it with `pip install openai` to use OpenAI embedding models.",
        category=ImportWarning,
    )


class PgVectorRM(dspy.Retrieve):
    """
    Implements a retriever that (as the name suggests) uses pgvector to retrieve passages,
    using a raw SQL query and a postgresql connection managed by psycopg2.

    It needs to register the pgvector extension with the psycopg2 connection

    Returns a list of dspy.Example objects

    Args:
        db_url (str): A PostgreSQL database URL in psycopg2's DSN format
        pg_table_name (Optional[str]): name of the table containing passages
        openai_client (openai.OpenAI): OpenAI client to use for computing query embeddings. Either
        embedding_func (Callable): A function to use for computing query embeddings. Either openai_
        content_field (str = "text"): Field containing the passage text. Defaults to "text"
        k (Optional[int]): Default number of top passages to retrieve. Defaults to 20
        embedding_field (str = "embedding"): Field containing passage embeddings. Defaults to "emb
        fields (List[str] = ['text']): Fields to retrieve from the table. Defaults to "text"
        embedding_model (str = "text-embedding-ada-002"): Field containing the OpenAI embedding

    Examples:
        Below is a code snippet that shows how to use PgVector as the default retriever

        ```python
        import dspy
```
```

```
"""
Retriever module for Azure AI Search
Author: Prajapati Harishkumar Kishorkumar (@HARISHKUMAR1112001)
"""

import warnings
from typing import Any, Callable, List, Optional, Union

import dspy
from dsp.utils.utils import dotdict

try:
    from azure.core.credentials import AzureKeyCredential
    from azure.search.documents import SearchClient
    from azure.search.documents._paging import SearchItemPaged
    from azure.search.documents.models import QueryType, VectorFilterMode, VectorizedQuery
except ImportError:
    raise ImportError(
        "You need to install azure-search-documents library"
        "Please use the command: pip install azure-search-documents==11.6.0b1",
    )

try:
    import openai
except ImportError:
    warnings.warn(
        "`openai` is not installed. Install it with `pip install openai` to use AzureOpenAI embedding mo
        category=ImportWarning,
    )


class AzureAISearchRM(dspy.Retrieve):

    """
    A retrieval module that utilizes Azure AI Search to retrieve top passages for a given query.

    Args:
        search_service_name (str): The name of the Azure AI Search service.
        search_api_key (str): The API key for accessing the Azure AI Search service.
        search_index_name (str): The name of the search index in the Azure AI Search service.
        field_text (str): The name of the field containing text content in the search index. This field will
        field_vector (Optional[str]): The name of the field containing vector content in the search index
        k (int, optional): The default number of top passages to retrieve. Defaults to 3.
        azure_openai_client (Optional[openai.AzureOpenAI]): An instance of the AzureOpenAI client.
        openai_embed_model (Optional[str]): The name of the OpenAI embedding model. Defaults to
        embedding_func (Optional[Callable]): A function for generating embeddings. Either openai_cli
        semantic_ranker (bool, optional): Whether to use semantic ranking. Defaults to False.
```

```
from typing import List, Optional, Union

import dspy
from dsp.utils import dotdict
from dspy.primitives.prediction import Prediction

try:
    import weaviate
except ImportError as err:
    raise ImportError(
        "The 'weaviate' extra is required to use WeaviateRM. Install it with `pip install dspy-ai[weaviate
    ) from err


class WeaviateRM(dspy.Retrieve):
    """A retrieval module that uses Weaviate to return the top passages for a given query.

    Assumes that a Weaviate collection has been created and populated with the following payload:
        - content: The text of the passage

    Args:
        weaviate_collection_name (str): The name of the Weaviate collection.
        weaviate_client (WeaviateClient): An instance of the Weaviate client.
        k (int, optional): The default number of top passages to retrieve. Default to 3.

    Examples:
        Below is a code snippet that shows how to use Weaviate as the default retriever:
        ```python
        import weaviate

        llm = dspy.Cohere(model="command-r-plus", api_key=api_key)
        weaviate_client = weaviate.connect_to_[local, wcs, custom, embedded]("your-path-here")
        retriever_model = WeaviateRM("my_collection_name", weaviate_client=weaviate_client)
        dspy.settings.configure(lm=llm, rm=retriever_model)

        retrieve = dspy.Retrieve(k=1)
        topK_passages = retrieve("what are the stages in planning, sanctioning and execution of publi
        ```


        Below is a code snippet that shows how to use Weaviate in the forward() function of a module
        ```python
        self.retrieve = WeaviateRM("my_collection_name", weaviate_client=weaviate_client, k=num_p
        ```
    """

    def __init__(
        self,
```

```
"""Retriever model for faiss: https://github.com/facebookresearch/faiss.
Author: Jagane Sundar: https://github.com/jagane.
"""

from typing import Optional, Union

import numpy as np

import dspy
from dsp.modules.sentence_vectorizer import SentenceTransformersVectorizer
from dsp.utils import dotdict

try:
    import faiss
except ImportError:
    faiss = None

if faiss is None:
    raise ImportError(
        """
        The faiss package is required. Install it using `pip install dspy-ai[faiss-cpu]`
        """,
    )


class FaissRM(dspy.Retrieve):
    """A retrieval module that uses an in-memory Faiss to return the top passages for a given query.

    Args:
        document_chunks: the input text chunks
        vectorizer: an object that is a subclass of BaseSentenceVectorizer
        k (int, optional): The number of top passages to retrieve. Defaults to 3.

    Returns:
        dspy.Prediction: An object containing the retrieved passages.

    Examples:
        Below is a code snippet that shows how to use this as the default retriver:
        ```python
        import dspy
        from dspy.retrieve import faiss_rm

        document_chunks = [
            "The superbowl this year was played between the San Francisco 49ers and the Kanasas C
            "Pop corn is often served in a bowl",
            "The Rice Bowl is a Chinese Restaurant located in the city of Tucson, Arizona",
            "Mars is the fourth planet in the Solar System",
```
```

```
from collections import defaultdict
from typing import List, Union

import dspy
from dsp.utils import dotdict

try:
    import marqo
except ImportError:
    raise ImportError(
        "The 'marqo' extra is required to use MarqoRM. Install it with `pip install dspy-ai[marqo]`",
    )

class MarqoRM(dspy.Retrieve):
    """
    A retrieval module that uses Marqo to return the top passages for a given query.

    Assumes that a Marqo index has been created and populated with the following payload:
        - document: The text of the passage

    Args:
        marqo_index_name (str): The name of the marqo index.
        marqo_client (marqo.client.Client): A marqo client instance.
        k (int, optional): The number of top passages to retrieve. Defaults to 3.
        page_content (str, optional): The name of the field in the marqo index that contains the text of
        filter_string (str, optional): A filter string to use when searching. Defaults to None.
        **kwargs: Additional keyword arguments to pass to the marqo search function.

    Examples:
        Below is a code snippet that shows how to use Marqo as the default retriver:
        ```python
        import marqo
        marqo_client = marqo.Client(url="http://0.0.0.0:8882")

        llm = dspy.OpenAI(model="gpt-3.5-turbo")
        retriever_model = MarqoRM("my_index_name", marqo_client=marqo_client)
        dspy.settings.configure(lm=llm, rm=retriever_model)
        ```

        Below is a code snippet that shows how to use Marqo in the forward() function of a module
        ```python
        self.retrieve = MarqoRM("my_index_name", marqo_client=marqo_client, k=num_passages)
        ```
    """

    def __init__(
        self,
```

```
"""
Retriever model for Milvus or Zilliz Cloud
"""

from typing import Callable, List, Optional, Union

import dspy
from dsp.utils import dotdict

try:
    from pymilvus import MilvusClient
except ImportError:
    raise ImportError(
        "The pymilvus library is required to use MilvusRM. Install it with `pip install dspy-ai[milvus]`",
    )


def openai_embedding_function(texts: List[str]):
    from openai import OpenAI

    client = OpenAI()
    response = client.embeddings.create(
        input=texts,
        model="text-embedding-3-small",
    )
    return [x.embedding for x in response.data]


class MilvusRM(dspy.Retrieve):
    """
    A retrieval module that uses Milvus to return passages for a given query.

    Assumes that a Milvus collection has been created and populated with the following field:
        - text: The text of the passage

    Args:
        collection_name (str): The name of the Milvus collection to query against.
        uri (str, optional): The Milvus connection uri. Defaults to "http://localhost:19530".
        token (str, optional): The Milvus connection token. Defaults to None.
        db_name (str, optional): The Milvus database name. Defaults to "default".
        embedding_function (callable, optional): The function to convert a list of text to embeddings.
            The embedding function should take a list of text strings as input and output a list of embed
            Defaults to None. By default, it will get OpenAI client by the environment variable OPENAI_
            and use OpenAI's embedding model "text-embedding-3-small" with the default dimension.
        k (int, optional): The number of top passages to retrieve. Defaults to 3.

    Returns:
```

```
import random
from typing import Dict, List, Optional, Union

import dsp
from dspy.predict.parameter import Parameter
from dspy.primitives.prediction import Prediction


def single_query_passage(passages):
    passages_dict = {key: [] for key in list(passages[0].keys())}
    for docs in passages:
        for key, value in docs.items():
            passages_dict[key].append(value)
    if "long_text" in passages_dict:
        passages_dict["passages"] = passages_dict.pop("long_text")
    return Prediction(**passages_dict)


class Retrieve(Parameter):
    name = "Search"
    input_variable = "query"
    desc = "takes a search query and returns one or more potentially relevant passages from a corp

    def __init__(self, k=3):
        self.stage = random.randbytes(8).hex()
        self.k = k

    def reset(self):
        pass

    def dump_state(self):
        state_keys = ["k"]
        return {k: getattr(self, k) for k in state_keys}

    def load_state(self, state):
        for name, value in state.items():
            setattr(self, name, value)

    def __call__(self, *args, **kwargs):
        return self.forward(*args, **kwargs)

    def forward(
        self,
        query_or_queries: Union[str, List[str]],
        k: Optional[int] = None,
        by_prob: bool = True,
        with_metadata: bool = False,
```

```
"""
Retriever model for deeplake
"""

from collections import defaultdict
from typing import List, Optional, Union

import openai

import dspy
from dsp.utils import dotdict

try:
    import openai

    ERRORS = (
        openai.RateLimitError,
        openai.APIError,
    )
except Exception:
    ERRORS = (openai.RateLimitError, openai.APIError)


class DeeplakeRM(dspy.Retrieve):

    """
    A retriever module that uses deeplake to return the top passages for a given query.

    Assumes that a Deep Lake Vector Store has been created and populated with the following payl
        - text: The text of the passage

    Args:
        deeplake_vectorstore_name (str): The name or path of the Deep Lake Vector Store.
        deeplake_client (VectorStore): An instance of the Deep Lake client.
        k (int, optional): The default number of top passages to retrieve. Defaults to 3.

    Examples:
        Below is a code snippet that shows how to use Deep Lake as the default retriver:
        ```python
        from deeplake import VectorStore
        llm = dspy.OpenAI(model="gpt-3.5-turbo")
        deeplake_client = VectorStore
        retriever_model = DeeplakeRM("my_vectorstore_path", deeplake_client=deeplake_client)
        dspy.settings.configure(lm=llm, rm=retriever_model)
        ```

        Below is a code snippet that shows how to use Deep Lake in the forward() function of a modu
```

```
from collections import defaultdict  # noqa: F401
from typing import Dict, List, Union  # noqa: UP035

import dspy
from dsp.utils import dotdict

try:
    from pyepsilla import vectordb
except ImportError:
    raise ImportError(  # noqa: B904
        "The 'pyepsilla' extra is required to use EpsillaRM. Install it with `pip install dspy-ai[epsilla]`",
    )


class EpsillaRM(dspy.Retrieve):
    def __init__(
        self,
        epsilla_client: vectordb.Client,
        db_name: str,
        db_path: str,
        table_name: str,
        k: int = 3,
        page_content: str = "document",
    ):
        self._epsilla_client = epsilla_client
        self._epsilla_client.load_db(db_name=db_name, db_path=db_path)
        self._epsilla_client.use_db(db_name=db_name)
        self.page_content = page_content
        self.table_name = table_name

        super().__init__(k=k)

    def forward(self, query_or_queries: Union[str, List[str]], k: Union[int, None] = None, **kwargs) ->
        queries = [query_or_queries] if isinstance(query_or_queries, str) else query_or_queries
        queries = [q for q in queries if q]
        limit = k if k else self.k
        all_query_results: list = []

        passages: Dict = defaultdict(float)

        for result_dict in all_query_results:
            for result in result_dict:
                passages[result[self.page_content]] += result["@distance"]
        sorted_passages = sorted(passages.items(), key=lambda x: x[1], reverse=False)[:limit]
        return dspy.Prediction(passages=[dotdict({"long_text": passage}) for passage, _ in sorted_pas
```

```
import logging
import os
import pickle
import random
import sys
import textwrap
from collections import defaultdict

import numpy as np
import optuna

from dspy.evaluate.evaluate import Evaluate
from dspy.propose import GroundedProposer
from dspy.teleprompt.teleprompt import Teleprompter
from dspy.teleprompt.utils import (
    create_n_fewshot_demo_sets,
    eval_candidate_program,
    get_dspy_source_code,
    get_program_with_highest_avg_score,
    get_signature,
    get_task_model_history_for_full_example,
    print_full_program,
    save_candidate_program,
    save_file_to_log_dir,
    set_signature,
    setup_logging,
)

try:
    import wandb
except ImportError:
    wandb = None

"""
USAGE SUGGESTIONS:

The following code can be used to compile a optimized signature teleprompter using MIPRO, and e

``` python
from dspy.teleprompt import MIPROv2

teleprompter = MIPROv2(prompt_model=prompt_model, task_model=task_model, metric=metric, r
kwargs = dict(num_threads=NUM_THREADS, display_progress=True, display_table=0)
compiled_prompt_opt = teleprompter.compile(program, trainset=trainset[:TRAIN_NUM], num_batch
eval_score = evaluate(compiled_prompt_opt, devset=evalset[:EVAL_NUM], **kwargs)
```
```

```
import random

import dspy
from dspy.evaluate.evaluate import Evaluate
from dspy.teleprompt.teleprompt import Teleprompter

from .bootstrap import BootstrapFewShot
from .vanilla import LabeledFewShot

# TODO: Don't forget dealing with the raw demos.
# TODO: Deal with the (pretty common) case of having a metric for filtering and a separate metric f
# The metric itself may tell though by the presence of trace.

# TODO: This function should take a max_budget and max_teacher_budget. That's in the number
# In this case, max_student_budget is max_budget - max_teacher_budget.
# For max_teacher_budget, this will just limit the total number of things we bootstrap.
# This can end up implicitly defining the number of candidate programs (i.e., stop when runs out). C
# For max_student_budget, this will be a more upfront calculation.
# Right now, it can also just induce the number of candidate programs. Later, it could be used more
# for selective early stopping.
# Progressive elimination sounds about right: after 50 examples, drop bottom third, after 100, anoth
# until only 3--5 are left for the end. Could also be systematic and add (earlier) stopping based on e
# In general, though, the early filtering is just saying: either there are some really bad ones, or som
# good ones, or most things are pretty close. In all of these cases, dropping the bottom third is not


class BootstrapFewShotWithRandomSearch(Teleprompter):
    def __init__(
        self,
        metric,
        teacher_settings={},
        max_bootstrapped_demos=4,
        max_labeled_demos=16,
        max_rounds=1,
        num_candidate_programs=16,
        num_threads=6,
        max_errors=10,
        stop_at_score=None,
        metric_threshold=None,
    ):
        self.metric = metric
        self.teacher_settings = teacher_settings
        self.max_rounds = max_rounds

        self.num_threads = num_threads
        self.stop_at_score = stop_at_score
        self.metric_threshold = metric_threshold
```

```
import random

from .teleprompt import Teleprompter


class LabeledFewShot(Teleprompter):
    def __init__(self, k=16):
        self.k = k

    def compile(self, student, *, trainset, sample=True):
        self.student = student.reset_copy()
        self.trainset = trainset

        if len(self.trainset) == 0:
            return self.student

        rng = random.Random(0)

        for predictor in self.student.predictors():
            if sample:
                predictor.demos = rng.sample(self.trainset, min(self.k, len(self.trainset)))
            else:
                predictor.demos = self.trainset[: min(self.k, len(self.trainset))]

        return self.student


# NOTE: I believe templatev2 keeps rdemos as long as they have the last field.
# This may change later, especially with the introduction of required vs optional fields.
# NOTE: Since we're relying on downstream code to handle the demos, this sampling may be sub-
```

```
import inspect
import logging
import math
import os
import random
import shutil
import sys

import numpy as np

try:
    from IPython.core.magics.code import extract_symbols
except ImportError:
    # Won't be able to read code from juptyer notebooks
    extract_symbols = None

import dspy
from dspy.predict.parameter import Parameter
from dspy.teleprompt.bootstrap import BootstrapFewShot, LabeledFewShot

"""
This file consists of helper functions for our variety of optimizers.
"""

### OPTIMIZER TRAINING UTILS ###


def create_minibatch(trainset, batch_size=50):
    """Create a minibatch from the trainset."""

    # Ensure batch_size isn't larger than the size of the dataset
    batch_size = min(batch_size, len(trainset))

    # Randomly sample indices for the mini-batch
    sampled_indices = random.sample(range(len(trainset)), batch_size)

    # Create the mini-batch using the sampled indices
    minibatch = [trainset[i] for i in sampled_indices]

    return minibatch


def eval_candidate_program(batch_size, trainset, candidate_program, evaluate):
    """Evaluate a candidate program on the trainset, using the specified batch size."""
    # Evaluate on the full trainset
    if batch_size >= len(trainset):
        score = evaluate(candidate_program, devset=trainset, display_table=0)
```

```
from .copro_optimizer import COPRO

"""
==========================================================
DEPRECATED!!!
PLEASE USE COPRO INSTEAD.
==========================================================

USAGE SUGGESTIONS:

The following code can be used to compile a optimized signature teleprompter, and evaluate it on a

teleprompter = SignatureOptimizer(prompt_model=prompt_model, metric=metric, breadth=BREAD
kwargs = dict(num_threads=NUM_THREADS, display_progress=True, display_table=0)
compiled_prompt_opt = teleprompter.compile(program.deepcopy(), devset=devset[:DEV_NUM], ev
eval_score = evaluate(compiled_prompt_opt, devset=evalset[:EVAL_NUM], **kwargs)

Note that this teleprompter takes in the following parameters:

* prompt_model: The model used for prompt generation. When unspecified, defaults to the model s
* metric: The task metric used for optimization.
* breadth: The number of new prompts to generate at each iteration. Default=10.
* depth: The number of times we should ask our prompt model to generate new prompts, with the h
* init_temperature: The temperature used to generate new prompts. Higher roughly equals more cr
* verbose: Tells the method whether or not to print intermediate steps.
* track_stats: Tells the method whether or not to track statistics about the optimization process.
            If True, the method will track the following statistics:
               * results_best: The min,max,avg,stddev of top 10 scores for each predictor at each dep
               * results_latest: The min,max,avg,stddev of newest prompt scores for each predictor a
               * total_calls: The total number of calls to the task metric.
            These statistics will be returned as attributes of the best program.
"""


class SignatureOptimizer(COPRO):
    def __init__(
        self,
        prompt_model=None,
        metric=None,
        breadth=10,
        depth=3,
        init_temperature=1.4,
        verbose=False,
        track_stats=False,
    ):
        print(
            "\u001b[31m[WARNING] SignatureOptimizer has been deprecated and replaced with COPR
```

```
import os
import random
import time

import ujson
from datasets.fingerprint import Hasher

import dsp
from dspy.signatures.signature import signature_to_template

from .bootstrap import BootstrapFewShot

# from dspy.primitives import Example
from .teleprompt import Teleprompter

# from .vanilla import LabeledFewShot

# from dspy.evaluate.evaluate import Evaluate


if os.environ.get("DSP_NOTEBOOK_CACHEDIR"):
    training_data_directory = os.path.join(os.environ.get("DSP_NOTEBOOK_CACHEDIR"), "compile
    print(training_data_directory)
else:
    training_data_directory = "local_cache/compiler"

if not os.path.exists(training_data_directory):
    os.makedirs(training_data_directory)


"""
TODO: Reduce and document the dependencies.

# !pip install evaluate
# !pip install tensorboardX
# !pip install transformers[torch]
# !pip install accelerate -U
# !pip install rouge_score


fewshot_teleprompter = BootstrapFewShot(metric=lambda gold, prediction, trace: gold.answer == p
                                        max_bootstrapped_demos=3, max_labeled_demos=16,
                                        teacher_settings=dict(lm=turbo))

fewshot = fewshot_teleprompter.compile(MyMultiHop(passages_per_hop=2), trainset=trainset)

"""
```

```
from .aggregation import majority
from .chain_of_thought import ChainOfThought
from .chain_of_thought_with_hint import ChainOfThoughtWithHint
from .knn import KNN
from .multi_chain_comparison import MultiChainComparison
from .predict import Predict
from .program_of_thought import ProgramOfThought
from .react import ReAct
from .retry import Retry
```

```
class Teleprompter:
    def __init__(self):
        pass
```

```
import textwrap
from dataclasses import dataclass
from typing import Generic, Literal, TypeVar

import pydantic

import dspy
from dspy import BaseModel
from dspy.functional.functional import TypedChainOfThought, TypedPredictor
from dspy.signatures import Signature
from dspy.signatures.field import InputField, OutputField

# TODO:
# - Parallelize the generation of new signatures when we have multiple predictors
# - Consider generating multiple new signatures at once, which we can test in parallel
# - Consider using the prompt optimizer to optimize the prompt optimizer :O


def make_info(signature: type[Signature]) -> BaseModel:
    """Creates a SignatureInfo pydantic type, that describes the Signature.

    Returns an instnce of this type, with the instructions and field descriptions of the input type.
    """
    # First, create the SignatureInfo type
    fields = {
        "instructions": (str, pydantic.Field(description="The instructions for the task")),
    }
    for name in signature.fields:
        fields[name + "_prefix"] = (str, pydantic.Field(description=f"The prefix for {name}"))
        fields[name + "_desc"] = (str, pydantic.Field(description=f"The description for {name}"))
    SignatureInfo = pydantic.create_model(  # noqa: N806
        f"SignatureInfo[{signature.__name__}]",
        **fields,
    )

    # Add a method to convert the SignatureInfo back into a Signature
    def to_signature(info):
        new_signature = signature.with_instructions(info.instructions)
        for name in signature.fields:
            new_signature = new_signature.with_updated_fields(
                name,
                prefix=getattr(info, name + "_prefix"),
                desc=getattr(info, name + "_desc"),
            )
        return new_signature

    SignatureInfo.to_signature = to_signature
```

```
import random

from dspy.teleprompt.teleprompt import Teleprompter

"""
TODO: The EnsembledProgram should actually imitate the structure of the individual programs (IF
"""


class Ensemble(Teleprompter):
    def __init__(self, *, reduce_fn=None, size=None, deterministic=False):
        """A common reduce_fn is dspy.majority."""

        assert deterministic is False, "TODO: Implement example hashing for deterministic ensemble.

        self.reduce_fn = reduce_fn
        self.size = size
        self.deterministic = deterministic

    def compile(self, programs):
        size = self.size
        reduce_fn = self.reduce_fn

        import dspy

        class EnsembledProgram(dspy.Module):
            def __init__(self):
                super().__init__()
                self.programs = programs

            def forward(self, *args, **kwargs):
                programs = random.sample(self.programs, size) if size else self.programs
                outputs = [prog(*args, **kwargs) for prog in programs]

                if reduce_fn:
                    return reduce_fn(outputs)

                return outputs

        return EnsembledProgram()
```

```
from dspy.teleprompt.mipro_optimizer import MIPRO

"""
============================================================
DEPRECATED!!!
PLEASE USE MIPRO INSTEAD.
============================================================

USAGE SUGGESTIONS:

The following code can be used to compile a optimized signature teleprompter using the BayesianS

from dspy.teleprompt import BayesianSignatureOptimizer

teleprompter = BayesianSignatureOptimizer(prompt_model=prompt_model, task_model=task_mod
kwargs = dict(num_threads=NUM_THREADS, display_progress=True, display_table=0)
compiled_prompt_opt = teleprompter.compile(program, devset=devset[:DEV_NUM], optuna_trials_
eval_score = evaluate(compiled_prompt_opt, devset=evalset[:EVAL_NUM], **kwargs)

Note that this teleprompter takes in the following parameters:

* prompt_model: The model used for prompt generation. When unspecified, defaults to the model s
* task_model: The model used for prompt generation. When unspecified, defaults to the model set
* metric: The task metric used for optimization.
* n: The number of new prompts and sets of fewshot examples to generate and evaluate. Default=1
* init_temperature: The temperature used to generate new prompts. Higher roughly equals more cr
* verbose: Tells the method whether or not to print intermediate steps.
* track_stats: Tells the method whether or not to track statistics about the optimization process.
            If True, the method will track a dictionary with a key corresponding to the trial number,
            and a value containing a dict with the following keys:
               * program: the program being evaluated at a given trial
               * score: the last average evaluated score for the program
               * pruned: whether or not this program was pruned
            This information will be returned as attributes of the best program.
"""


class BayesianSignatureOptimizer(MIPRO):
    def __init__(
        self,
        prompt_model=None,
        task_model=None,
        teacher_settings={},
        n=10,
        metric=None,
        init_temperature=1.0,
        verbose=False,
```

```
import types
from typing import List, Optional

import dsp
from dspy.predict.knn import KNN
from dspy.teleprompt import BootstrapFewShot

from .teleprompt import Teleprompter


class KNNFewShot(Teleprompter):
    def __init__(self, k: int, trainset: List[dsp.Example], vectorizer: Optional[dsp.BaseSentenceVecto
        self.KNN = KNN(k, trainset, vectorizer=vectorizer)
        self.few_shot_bootstrap_args = few_shot_bootstrap_args

    def compile(self, student, *, teacher=None, trainset=None, valset=None):
        student_copy = student.reset_copy()

        def forward_pass(_, **kwargs):
            knn_trainset = self.KNN(**kwargs)
            few_shot_bootstrap = BootstrapFewShot(**self.few_shot_bootstrap_args)
            compiled_program = few_shot_bootstrap.compile(
                student,
                teacher=teacher,
                trainset=knn_trainset,
            )
            return compiled_program(**kwargs)

        student_copy.forward = types.MethodType(forward_pass, student_copy)
        return student_copy
```

```
import random
import threading
from typing import Dict, Optional

import tqdm

import dsp
import dspy
from dspy.primitives import Example

from .teleprompt import Teleprompter
from .vanilla import LabeledFewShot

# TODO: metrics should return an object with __bool__ basically, but fine if they're more complex.
# They can also be sortable.

# TODO: Switch here from dsp.Example to dspy.Example. Right now, it's okay because it's internal
# NOTE: Notice the places where we don't shuffle examples. I do like that this one doesn't shuffle.
# Other ones that consider options may want to use both unshuffled and then shuffle a few times, w
# considering candidates.

# TODO: the max_rounds via branch_idx to get past the cache, not just temperature.
# In principle, we can also sample multiple outputs from the final generation step
# (or even each step, in case the validation function just wants *one* thing that works, but nah)
# and try them all. Having a pretty solid guess on the "final step" of each example isn't hard by the s
# in the sense that we have the trace from the first round. (Yes it may change but that's an edge ca
# won't hurt our "best effort" guarantees.)

# TODO: When this bootstraps for another teleprompter like finetune, we want all demos we gather
# But when it's for direct use we may want to sample ONE demo per predictor--example pair.
# This is important for "multi-use" modules.

# TODO: Add baselines=[...]

class BootstrapFewShot(Teleprompter):
    def __init__(
        self,
        metric=None,
        metric_threshold=None,
        teacher_settings: Optional[Dict]=None,
        max_bootstrapped_demos=4,
        max_labeled_demos=16,
        max_rounds=1,
        max_errors=5,
    ):
        """
        A Teleprompter class that composes a set of demos/examples to go into a predictor's prompt.
```

```
import optuna

from dspy.evaluate.evaluate import Evaluate
from dspy.teleprompt.teleprompt import Teleprompter

from .bootstrap import BootstrapFewShot


class BootstrapFewShotWithOptuna(Teleprompter):
    def __init__(
        self,
        metric,
        teacher_settings={},
        max_bootstrapped_demos=4,
        max_labeled_demos=16,
        max_rounds=1,
        num_candidate_programs=16,
        num_threads=6,
    ):
        self.metric = metric
        self.teacher_settings = teacher_settings
        self.max_rounds = max_rounds

        self.num_threads = num_threads

        self.min_num_samples = 1
        self.max_num_samples = max_bootstrapped_demos
        self.num_candidate_sets = num_candidate_programs
        # self.max_num_traces = 1 + int(max_bootstrapped_demos / 2.0 * self.num_candidate_sets)

        # Semi-hacky way to get the parent class's _bootstrap function to stop early.
        # self.max_bootstrapped_demos = self.max_num_traces
        self.max_labeled_demos = max_labeled_demos

        print("Going to sample between", self.min_num_samples, "and", self.max_num_samples, "trac
        # print("Going to sample", self.max_num_traces, "traces in total.")
        print("Will attempt to train", self.num_candidate_sets, "candidate sets.")

    def objective(self, trial):
        program2 = self.student.reset_copy()
        for (name, compiled_predictor), (_, program2_predictor) in zip(
            self.compiled_teleprompter.named_predictors(), program2.named_predictors(),
        ):
            all_demos = compiled_predictor.demos
            demo_index = trial.suggest_int(f"demo_index_for_{name}", 0, len(all_demos) - 1)
            selected_demo = dict(all_demos[demo_index])
            program2_predictor.demos = [selected_demo]
```

```
import math
import random
import sys
import textwrap
from collections import defaultdict
from typing import Any

import optuna

import dsp
import dspy
from dspy.evaluate.evaluate import Evaluate
from dspy.signatures import Signature
from dspy.signatures.signature import signature_to_template
from dspy.teleprompt import BootstrapFewShot
from dspy.teleprompt.teleprompt import Teleprompter

"""
USAGE SUGGESTIONS:

The following code can be used to compile a optimized signature teleprompter using MIPRO, and e

``` python
from dspy.teleprompt import MIPRO

teleprompter = MIPRO(prompt_model=prompt_model, task_model=task_model, metric=metric, nur
kwargs = dict(num_threads=NUM_THREADS, display_progress=True, display_table=0)
compiled_prompt_opt = teleprompter.compile(program, trainset=trainset[:TRAIN_NUM], num_trials
eval_score = evaluate(compiled_prompt_opt, devset=evalset[:EVAL_NUM], **kwargs)
```

Note that this teleprompter takes in the following parameters:

* prompt_model: The model used for prompt generation. When unspecified, defaults to the model s
* task_model: The model used for prompt generation. When unspecified, defaults to the model set
* metric: The task metric used for optimization.
* num_candidates: The number of new prompts and sets of fewshot examples to generate and eva
* init_temperature: The temperature used to generate new prompts. Higher roughly equals more cr
* verbose: Tells the method whether or not to print intermediate steps.
* track_stats: Tells the method whether or not to track statistics about the optimization process.
            If True, the method will track a dictionary with a key corresponding to the trial number,
            and a value containing a dict with the following keys:
                * program: the program being evaluated at a given trial
                * score: the last average evaluated score for the program
                * pruned: whether or not this program was pruned
            This information will be returned as attributes of the best program.
"""
```

```
from collections import defaultdict

import dsp
import dspy
from dspy.evaluate.evaluate import Evaluate
from dspy.signatures import Signature
from dspy.teleprompt.teleprompt import Teleprompter

"""
USAGE SUGGESTIONS:

The following code can be used to compile a optimized signature teleprompter, and evaluate it on a

teleprompter = COPRO(prompt_model=prompt_model, metric=metric, breadth=BREADTH, depth=
kwargs = dict(num_threads=NUM_THREADS, display_progress=True, display_table=0)
compiled_prompt_opt = teleprompter.compile(program.deepcopy(), trainset=trainset[:DEV_NUM], e
eval_score = evaluate(compiled_prompt_opt, devset=evalset[:EVAL_NUM], **kwargs)

Note that this teleprompter takes in the following parameters:

* prompt_model: The model used for prompt generation. When unspecified, defaults to the model s
* metric: The task metric used for optimization.
* breadth: The number of new prompts to generate at each iteration. Default=10.
* depth: The number of times we should ask our prompt model to generate new prompts, with the h
* init_temperature: The temperature used to generate new prompts. Higher roughly equals more cr
* track_stats: Tells the method whether or not to track statistics about the optimization process.
                If True, the method will track the following statistics:
                    * results_best: The min,max,avg,stddev of top 10 scores for each predictor at each dep
                    * results_latest: The min,max,avg,stddev of newest prompt scores for each predictor a
                    * total_calls: The total number of calls to the task metric.
                These statistics will be returned as attributes of the best program.
"""


class BasicGenerateInstruction(Signature):
    """You are an instruction optimizer for large language models. I will give you a ``signature`` of fie

    basic_instruction = dspy.InputField(desc="The initial instructions before optimization")
    proposed_instruction = dspy.OutputField(desc="The improved instructions for the language mod
    proposed_prefix_for_output_field = dspy.OutputField(
        desc="The string at the end of the prompt, which will help the model start solving the task",
    )


class GenerateInstructionGivenAttempts(dspy.Signature):
    """You are an instruction optimizer for large language models. I will give some task instructions I'
```

```
import copy
import random

from langchain_core.pydantic_v1 import Extra
from langchain_core.runnables import Runnable

import dsp
import dspy
from dspy.predict.parameter import Parameter
from dspy.predict.predict import Predict
from dspy.primitives.prediction import Prediction
from dspy.signatures.field import OldInputField, OldOutputField
from dspy.signatures.signature import infer_prefix

# TODO: This class is currently hard to test, because it hardcodes gpt-4 usage:
# gpt4T = dspy.OpenAI(model='gpt-4-1106-preview', max_tokens=4000, model_type='chat')

class Template2Signature(dspy.Signature):
    """You are a processor for prompts. I will give you a prompt template (Python f-string) for an arbi
Your job is to prepare three modular pieces: (i) any essential task instructions or guidelines, (ii) a lis

    template = dspy.InputField(format=lambda x: f"```\n\n{x.strip()}\n\n```\n\nLet's now prepare three
    essential_instructions = dspy.OutputField()
    input_keys = dspy.OutputField(desc='comma-separated list of valid variable names')
    output_key = dspy.OutputField(desc='a valid variable name')


class ShallowCopyOnly:
    def __init__(self, obj): self.obj = obj
    def __getattr__(self, item): return getattr(self.obj, item)
    def __deepcopy__(self, memo): return ShallowCopyOnly(copy.copy(self.obj))


class LangChainPredict(Predict, Runnable): #, RunnableBinding):
    class Config: extra = Extra.allow  # Allow extra attributes that are not defined in the model

    def __init__(self, prompt, llm, **config):
        Runnable.__init__(self)
        Parameter.__init__(self)

        self.langchain_llm = ShallowCopyOnly(llm)

        try: langchain_template = '\n'.join([msg.prompt.template for msg in prompt.messages])
        except AttributeError: langchain_template = prompt.template

        self.stage = random.randbytes(8).hex()
        self.signature, self.output_field_key = self._build_signature(langchain_template)
```

```
import random

from pydantic import BaseModel

import dsp
from dspy.predict.parameter import Parameter
from dspy.primitives.prediction import Prediction
from dspy.signatures.signature import ensure_signature, signature_to_template


class Predict(Parameter):
    def __init__(self, signature, **config):
        self.stage = random.randbytes(8).hex()
        self.signature = ensure_signature(signature)
        self.config = config
        self.reset()

    def reset(self):
        self.lm = None
        self.traces = []
        self.train = []
        self.demos = []

    def dump_state(self):
        state_keys = ["lm", "traces", "train"]
        state = {k: getattr(self, k) for k in state_keys}

        state["demos"] = []
        for demo in self.demos:
            demo = demo.copy()

            for field in demo:
                if isinstance(demo[field], BaseModel):
                    demo[field] = demo[field].model_dump_json()

            state["demos"].append(demo)

        # Cache the signature instructions and the last field's name.
        state["signature_instructions"] = self.signature.instructions

        *_, last_key = self.signature.fields.keys()
        state["signature_prefix"] = self.signature.fields[last_key].json_schema_extra["prefix"]

        return state

    def load_state(self, state):
        for name, value in state.items():
```

```
import copy

import dsp
import dspy

from .predict import Predict


class Retry(Predict):
    def __init__(self, module):
        super().__init__(module.signature)
        self.module = module
        self.original_signature = module.extended_signature if isinstance(module, dspy.ChainOfThou
        self.original_forward = module.forward
        self.new_signature = self._create_new_signature(self.original_signature)

    def _create_new_signature(self, signature):
        # Add "Past" input fields for each output field
        for key, value in signature.output_fields.items():
            actual_prefix = value.json_schema_extra["prefix"].split(":")[0] + ":"
            signature = signature.append(f"past_{key}", dspy.InputField(
                prefix="Previous " + actual_prefix,
                desc=f"past {actual_prefix} with errors",
                format=value.json_schema_extra.get("format"),
            ))

        signature = signature.append("feedback", dspy.InputField(
            prefix="Instructions:",
            desc="Some instructions you must satisfy",
            format=str,
        ))

        return signature

    def forward(self, *, past_outputs, **kwargs):
        # Take into account the possible new signature, as in TypedPredictor
        new_signature = kwargs.pop("new_signature", None)
        if new_signature:
            self.original_signature = new_signature
            self.new_signature = self._create_new_signature(self.original_signature)

        # Convert the dict past_outputs={"answer": ...} to kwargs
        # {past_answer=..., ...}
        for key, value in past_outputs.items():
            past_key = f"past_{key}"
            if past_key in self.new_signature.input_fields:
                kwargs[past_key] = value
```

```
from .aggregation import majority
from .chain_of_thought import ChainOfThought
from .chain_of_thought_with_hint import ChainOfThoughtWithHint
from .knn import KNN
from .multi_chain_comparison import MultiChainComparison
from .predict import Predict
from .program_of_thought import ProgramOfThought
from .react import ReAct
from .retry import Retry
```

```
import dsp
import dspy
from dspy.signatures.signature import ensure_signature

from .predict import Predict

# TODO: FIXME: Insert this right before the *first* output field. Also rewrite this to use the new sign

# TODO: This shouldn't inherit from Predict. It should be a module that has one or two predictors.
# Let's focus on the activated case. It's a predictor with the expanded signature.
# Now, when deactivated, it's a predictor with the original signature.
# When activate is None, though, we need the expanded one but during forward we need to pass t

"""
class ChainOfThought(dspy.Module):
    def __init__(self, signature):

        input_fields, output_fields = dspy.process_signature(signature)
        output_fields = dict(rationale=dspy.OutputField(prefix="Reasoning: Let's think step by step."),
        self.signature = dspy.Signature(input_fields, output_fields)

        self.predict = dspy.Predict(self.signature)

    def forward(self, **kwargs):
        return self.predict(**kwargs)

# How this should look like. But with also passing signature=simpler_signature to the predict modul
"""


class ChainOfThought(Predict):
    def __init__(self, signature, rationale_type=None, activated=True, **config):
        super().__init__(signature, **config)

        self.activated = activated

        signature = ensure_signature(self.signature)
        *_keys, last_key = signature.output_fields.keys()

        rationale_type = rationale_type or dspy.OutputField(
            prefix="Reasoning: Let's think step by step in order to",
            desc="${produce the " + last_key + "}. We ...",
        )

        self.extended_signature = signature.prepend("rationale", rationale_type, type_=str)

    def forward(self, **kwargs):
```

```
import re
from copy import deepcopy
from typing import Any, Callable, Dict, List, Optional

from llama_index.core.base.llms.base import BaseLLM
from llama_index.core.base.llms.generic_utils import (
    prompt_to_messages,
)
from llama_index.core.base.llms.types import ChatMessage
from llama_index.core.base.query_pipeline.query import InputKeys, OutputKeys, QueryComponen
from llama_index.core.callbacks.base import CallbackManager
from llama_index.core.prompts import BasePromptTemplate, PromptTemplate
from llama_index.core.query_pipeline import QueryPipeline

import dsp
import dspy
from dspy import Predict
from dspy.signatures.field import InputField, OutputField
from dspy.signatures.signature import ensure_signature, make_signature, signature_to_template


def get_formatted_template(predict_module: Predict, kwargs: Dict[str, Any]) -> str:
    """Get formatted template from predict module."""
    # Extract the three privileged keyword arguments.
    signature = ensure_signature(predict_module.signature)
    demos = predict_module.demos

    # All of the other kwargs are presumed to fit a prefix of the signature.
    # That is, they are input variables for the bottom most generation, so
    # we place them inside the input - x - together with the demos.
    x = dsp.Example(demos=demos, **kwargs)

    # Switch to legacy format for dsp.generate
    template = signature_to_template(signature)

    return template(x)


def replace_placeholder(text: str) -> str:
    # Use a regular expression to find and replace ${...} with ${{...}}
    return re.sub(r'\$\{([^\{\}]*)\}', r'${{\1}}', text)


def _input_keys_from_template(template: dsp.Template) -> InputKeys:
    """Get input keys from template."""
    # get only fields that are marked OldInputField and NOT OldOutputField
    # template_vars = list(template.kwargs.keys())
```

```
class Parameter:
    pass

class Hyperparameter:
    pass
```

```
from typing import List, Optional

import numpy as np

import dsp


class KNN:
    def __init__(self, k: int, trainset: List[dsp.Example], vectorizer: Optional[dsp.BaseSentenceVecto
        self.k = k
        self.trainset = trainset
        self.vectorizer = vectorizer or dsp.SentenceTransformersVectorizer()
        trainset_casted_to_vectorize = [
            " | ".join([f"{key}: {value}" for key, value in example.items() if key in example._input_keys])
            for example in self.trainset
        ]
        self.trainset_vectors = self.vectorizer(trainset_casted_to_vectorize).astype(np.float32)

    def __call__(self, **kwargs) -> List[dsp.Example]:
        with dsp.settings.context(vectorizer=self.vectorizer):
            input_example_vector = self.vectorizer([" | ".join([f"{key}: {val}" for key, val in kwargs.items()
            scores = np.dot(self.trainset_vectors, input_example_vector.T).squeeze()
            nearest_samples_idxs = scores.argsort()[-self.k :][::-1]
            train_sampled = [self.trainset[cur_idx] for cur_idx in nearest_samples_idxs]
            return train_sampled
```

```
import dsp
import dspy
from dspy.signatures.signature import ensure_signature

from ..primitives.program import Module
from .predict import Predict

# TODO: Simplify a lot.
# TODO: Divide Action and Action Input like langchain does for ReAct.

# TODO: There's a lot of value in having a stopping condition in the LM calls at `\n\nObservation:`


class ReAct(Module):
    def __init__(self, signature, max_iters=5, num_results=3, tools=None):
        super().__init__()
        self.signature = signature = ensure_signature(signature)
        self.max_iters = max_iters

        self.tools = tools or [dspy.Retrieve(k=num_results)]
        self.tools = {tool.name: tool for tool in self.tools}

        self.input_fields = self.signature.input_fields
        self.output_fields = self.signature.output_fields

        assert len(self.output_fields) == 1, "ReAct only supports one output field."

        inputs_ = ", ".join([f"`{k}`" for k in self.input_fields.keys()])
        outputs_ = ", ".join([f"`{k}`" for k in self.output_fields.keys()])

        instr = []

        if self.signature.instructions is not None:
            instr.append(f"{self.signature.instructions}\n")

        instr.extend([
            f"You will be given {inputs_} and you will respond with {outputs_}.\n",
            "To do this, you will interleave Thought, Action, and Observation steps.\n",
            "Thought can reason about the current situation, and Action can be the following types:\n",
        ])

        self.tools["Finish"] = dspy.Example(
            name="Finish",
            input_variable=outputs_.strip("`"),
            desc=f"returns the final {outputs_} and finishes the task",
        )
```

```
import dsp
import dspy

from .predict import Predict

# TODO: FIXME: Insert this right before the *first* output field. Also rewrite this to use the new sign

class ChainOfThoughtWithHint(Predict):
    def __init__(self, signature, rationale_type=None, activated=True, **config):
        super().__init__(signature, **config)
        self.activated = activated
        signature = self.signature

        *keys, last_key = signature.fields.keys()
        rationale_type = rationale_type or dspy.OutputField(
            prefix="Reasoning: Let's think step by step in order to",
            desc="${produce the " + last_key + "}. We ...",
        )
        self.extended_signature1 = self.signature.insert(-2, "rationale", rationale_type, type_=str)

        DEFAULT_HINT_TYPE = dspy.OutputField()
        self.extended_signature2 = self.extended_signature1.insert(-2, "hint", DEFAULT_HINT_TYPE

    def forward(self, **kwargs):
        signature = self.signature

        if self.activated is True or (self.activated is None and isinstance(dsp.settings.lm, dsp.GPT3)):
            if 'hint' in kwargs and kwargs['hint']:
                signature = self.extended_signature2
            else:
                signature = self.extended_signature1

        return super().forward(signature=signature, **kwargs)


"""
TODO: In principle, we can update the field's prefix during forward too to fill any thing based on the

IF the user didn't overwrite our default rationale_type.
"""
```

```
from dsp.utils import normalize_text
from dspy.primitives.prediction import Completions, Prediction

default_normalize = lambda s: normalize_text(s) or None


def majority(prediction_or_completions, normalize=default_normalize, field=None):
    """
        Returns the most common completion for the target field (or the last field) in the signature.
        When normalize returns None, that completion is ignored.
        In case of a tie, earlier completion are prioritized.
    """

    assert any(isinstance(prediction_or_completions, t) for t in [Prediction, Completions, list])
    input_type = type(prediction_or_completions)

    # Get the completions
    if isinstance(prediction_or_completions, Prediction):
        completions = prediction_or_completions.completions
    else:
        completions = prediction_or_completions

    try:
        signature = completions.signature
    except:
        signature = None

    if not field:
        if signature:
            field = signature.output_fields[-1]
        else:
            field = list(completions[0].keys())[-1]

    # Normalize
    normalize = normalize if normalize else lambda x: x
    normalized_values = [normalize(completion[field]) for completion in completions]
    normalized_values_ = [x for x in normalized_values if x is not None]

    # Count
    value_counts = {}
    for value in (normalized_values_ or normalized_values):
        value_counts[value] = value_counts.get(value, 0) + 1

    majority_value = max(value_counts, key=value_counts.get)

    # Return the first completion with the majority value in the field
    for completion in completions:
```

```
import dspy
from dspy.signatures.signature import ensure_signature

from ..primitives.program import Module
from .predict import Predict


class MultiChainComparison(Module):
    def __init__(self, signature, M=3, temperature=0.7, **config):
        super().__init__()

        self.M = M
        signature = ensure_signature(signature)

        *_, self.last_key = signature.output_fields.keys()

        for idx in range(M):
            signature = signature.append(
                f"reasoning_attempt_{idx+1}",
                dspy.InputField(
                    prefix=f"Student Attempt #{idx+1}:", desc="${reasoning attempt}",
                ),
            )

        signature = signature.prepend(
            "rationale",
            dspy.OutputField(
                prefix="Accurate Reasoning: Thank you everyone. Let's now holistically",
                desc="${corrected reasoning}",
            ),
        )

        self.predict = Predict(signature, temperature=temperature, **config)

    def forward(self, completions, **kwargs):
        attempts = []

        for c in completions:
            rationale = c.rationale.strip().split("\n")[0].strip()
            answer = c[self.last_key].strip().split("\n")[0].strip()
            attempts.append(
                f"«I'm trying to {rationale} I'm not sure but my prediction is {answer}»",
            )

        assert len(attempts) == self.M, f"The number of attempts ({len(attempts)}) doesn't match the e

        kwargs = {
```

```
import re

import dspy
from dspy.signatures.signature import ensure_signature

from ..primitives.program import Module
from ..primitives.python_interpreter import CodePrompt, PythonInterpreter


class ProgramOfThought(Module):
    def __init__(self, signature, max_iters=3, import_white_list=None):
        super().__init__()
        self.signature = signature = ensure_signature(signature)
        self.max_iters = max_iters
        self.import_white_list = import_white_list

        self.input_fields = signature.input_fields
        self.output_fields = signature.output_fields

        assert len(self.output_fields) == 1, "PoT only supports one output field."

        self.output_field_name = next(iter(self.output_fields))
        inputs_ = ", ".join(
            [f"`{field_name}`" for field_name in self.input_fields.keys()],
        )
        outputs_ = f"`{self.output_field_name}`"

        assert len(self.output_fields) == 1, "PoT only supports one output field."

        instr = []
        instr.append(
            f"You will be given {inputs_} and you will respond with {outputs_}.",
        )
        instr.append(
            f"Generating executable Python code that programmatically computes the correct {outputs_
        )
        instr.append(
            f"After you're done with the computation, make sure the last line in your code evaluates to t
        )
        instr = "\n".join(instr)

        self.code_generate = dspy.ChainOfThought(
            dspy.Signature(
                self._generate_signature("generate").fields,
                self._generate_instruction("generate"),
            ),
        )
```