zilliz

使用 GPTCache, 向 LLM「省省省」私有化部署开发 say hi

-- 为 LLM 请求定制 Cache

讲师介绍

付邦

Zilliz 软件工程师

GPTCache 作者
Milvus 系统开发者

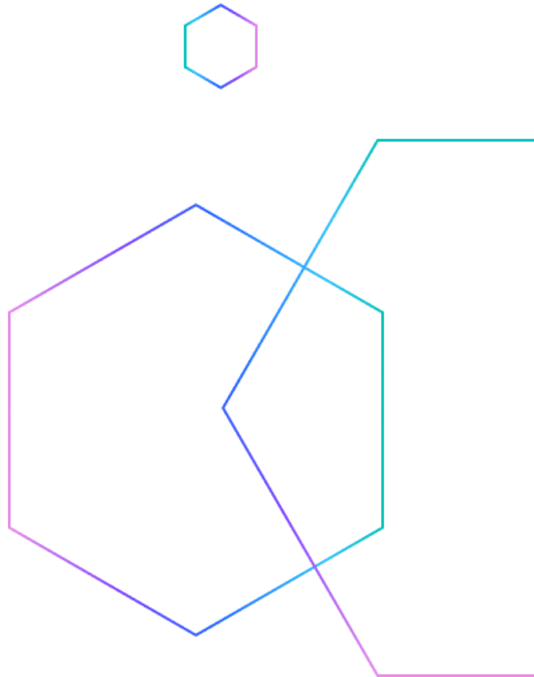
CONTENTS

01

项目介绍

项目介绍

1. 当前 AI 领域的现状
2. GPTCache 简介
3. GPTCache 作用



当前 AI 领域的现状

2022.11.30，也就是大概半年前，OpenAI 发布文章 Introducing ChatGPT，标志着正式迈入大模型时代，经过半年时间的发酵，ChatGPT 已经被大家熟知，其传播速度超过国内的 QQ 和微信。

今年2月，ChatGPT 也逐渐被国内所知，国内各个公司也开始布局语言模型赛道，如百度的文心一言、阿里的通义千问、复旦大学的 Moss、科大讯飞的星火等。

在观察近两个月的 GitHub Trending 榜，可以发现基本上每天都有关于 GPT 项目，如前段时间的 LangChain、llama_index、Auto-GPT 等，GPTCache 也连续上榜四天。AI 领域的论文也有许多。

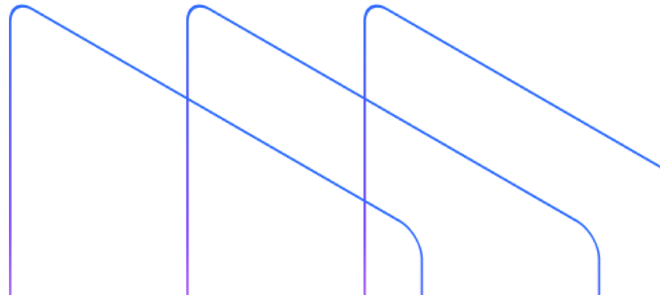
目前可以看出的，对于 ChatGPT 相关的开发，大概可以分为以下几种：

1. ChatGPT 模型平替，如 LLama、Dolly、Moss 等

2. 模型 FineTune，如 LawGPT、BioGPT、Huatuo-Llama-Med-Chinese 等

3. GPT 应用开发，包括桌面应用、反向代理在线 LLM 模型、基于 GPT 能力解决日常问题

4. GPT 能力增强，如 LLM 流程编排、token 限制突破、给 ChatGPT 提供外部数据源等



GPTCache 简介

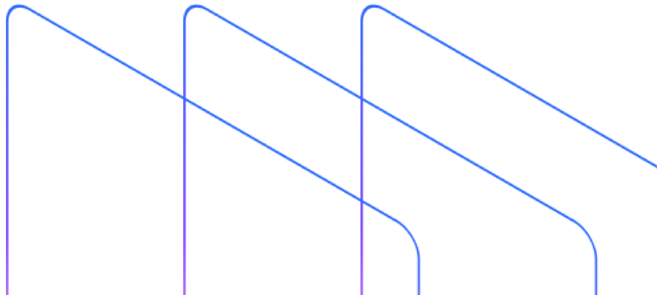
项目地址：https://github.com/zilliztech/GPTCache

GPTCache 由来：公司内部在进行 OSSChat 项目开发过程中，发现 ChatGPT 可能会成为阻碍 OSSChat 提升性能的瓶颈。一，不稳定的 ChatGPT 服务会拉低 OSSChat 响应速度；二，每次调用 ChatGPT 接口，都会产生新的费用，这导致 OSSChat 的使用成本不断拉升。

GPTCache 是什么?

ChatGPT 和其他大语言模型，可以被广泛应用于各种开发场景中。随着应用处理请求的增加，在固定资源的情况下将增加请求延时，或者为了保证请求延时满足用户需求，则需要增加计算资源，也将间接增加应用开发成本。假如使用类似 ChatGPT 的在线服务，也会有请求速率限制，同时请求数量的增加也必定导致成本的增加。

GPTCache 则是为了解决上述问题诞生的，主要是为 LLM 相关应用构建相似语义缓存，相似的问题请求多次，直接从缓存中获取，这样将减少请求响应时间，同时也降低了 LLM 的使用成本。



GPTCache 简介

GPTCache 在 GitHub：

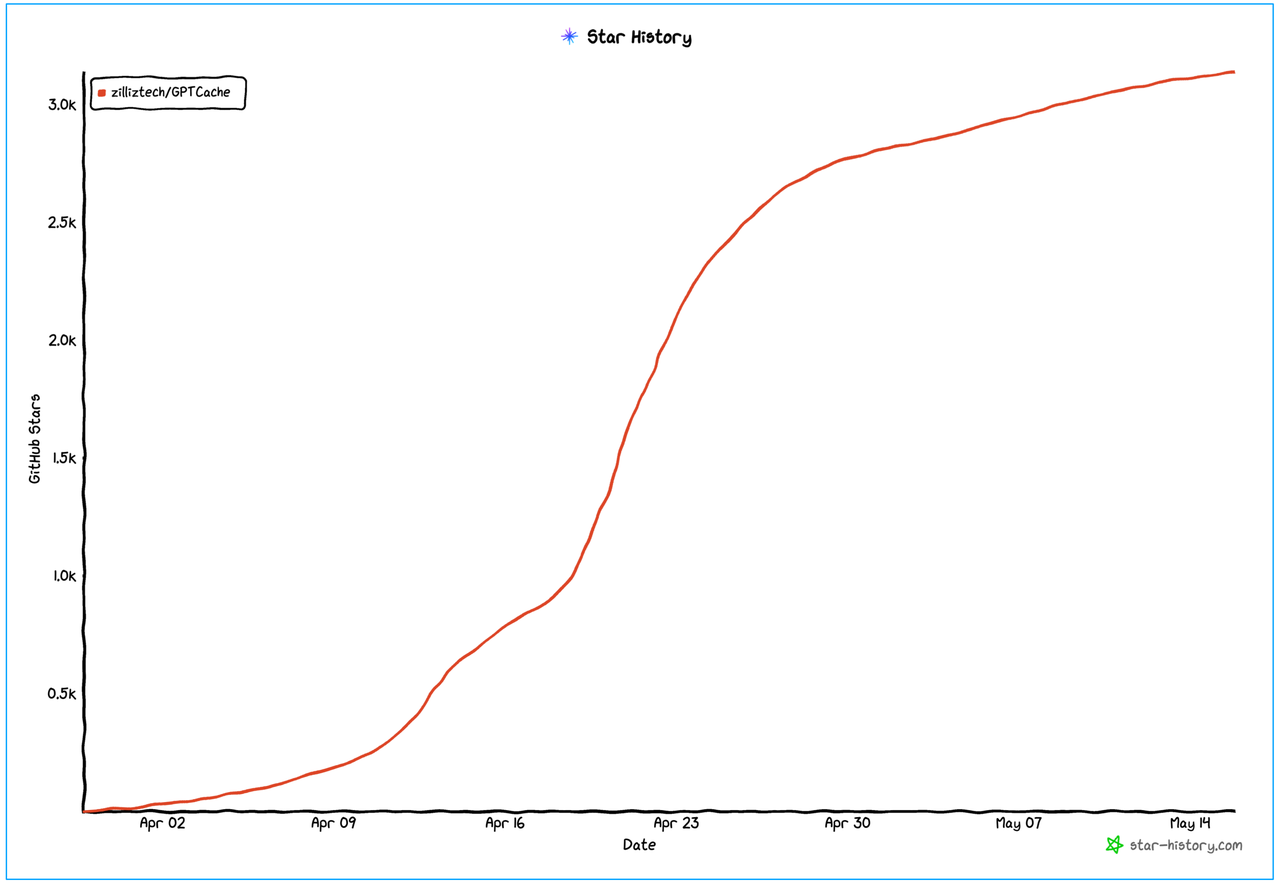
发布近一个月获得 **3.1k** star；

近 **200** 个 fork 数量；

统计的被引用数量，近 300 个；

PyPi 发布版本：0.1.1~0.1.23

pip 累计下载数量：超 **14w**



GPTCache 作用

GPTCache 作用是什么?

1. 降低 LLM 使用费用：当请求使用缓存结果，自然降低请求次数，减少使用成本

2. 性能优化：从缓存数据中获取时间将降低一个数量级

3. 兼容性强，多种应用场景：GPTCache 提供多种 LLM 的镜像接口，只需修改 import 路径，即可模型 LLM 请求；

4. 改善 LLM 服务的可扩展性和可用性：对于相似的问题使用缓存答案，将有效缓解服务无法响应这一问题。

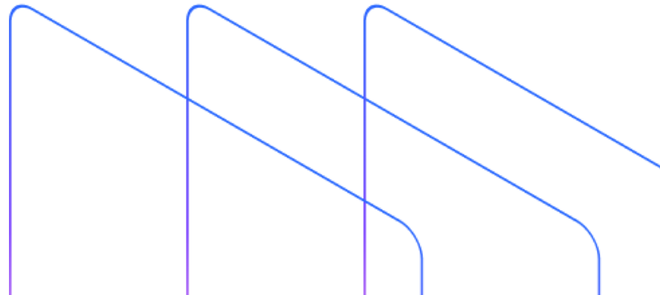
GPTCache 作用于哪些场景比较合适?

1. 某一垂直领域的 LLM 相关应用，如法律、生物、医学等；

2. 固定的 LLM 相关应用，如某公司内部或个人使用的 ChatBot；

3. 开发的 LLM 应用在某些时间内的请求具有高度相似性，如节日等；

4. 具有大用户群体的 LLM 应用，如果给用户群体进行分类，类似用户用同一缓存



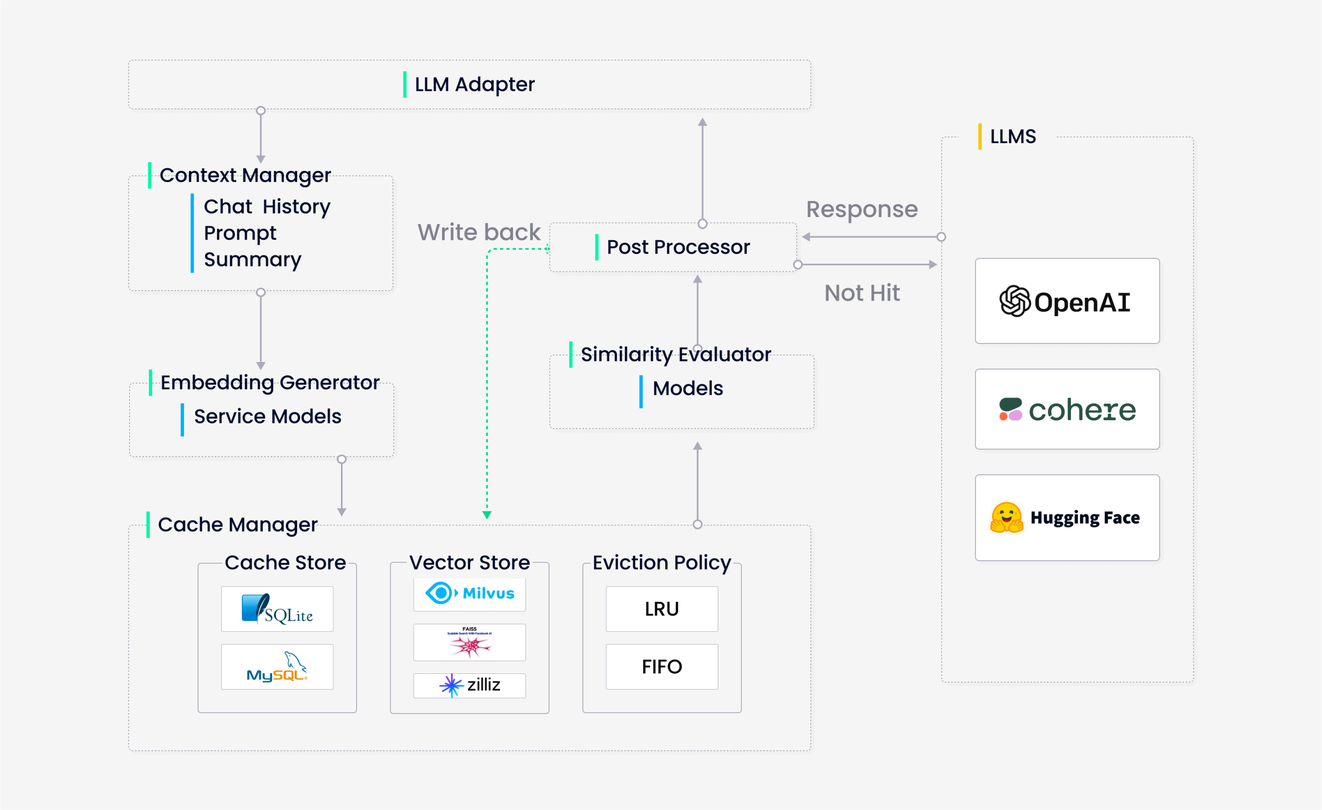
02

项目结构

项目结构

核心组件

1. LLM Adapter
2. Pre-Processor
3. Embedding
4. Cache Manager
5. Similarity Evaluation
6. Post-Processor
7. GPTCache Server



LLM Adapter

使用 GPTCache 代理 LLM 的请求，保证用户以最低的成本、风险最小的方式接入 GPTCache

Text-to-Text

1. OpenAI
2. Llama
3. Dolly
4. LangChain

MultiModel

1. StableDiffusion
2. Stability
3. Replicate
4. MiniGPT4 (VQA)

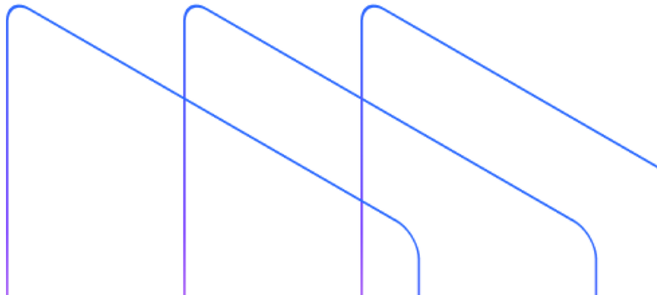
API

1. put
2. get
3. init_similar_cache
4. init_similar_cache_from_config

[Example]

```
from gptcache.adapter import openai

response = openai.ChatCompletion.create(
            model='gpt-3.5-turbo',
            messages=[
              {
                  'role': 'user',
                  'content': "what's github"
              }],
          )
```



Pre-Processor

预处理，处理 LLM 请求的输入，如对输入信息进行删减，也可进行增添，或者进行信息修改等，处理后的结果将会交由 Embedding 组件转换为向量

基础内置实现

last_content
last_content_without_prompt
all_content
nop
...

Context预处理

ContextProcess **Interface**
SummarizationContextProcess
SelectiveContextProcess

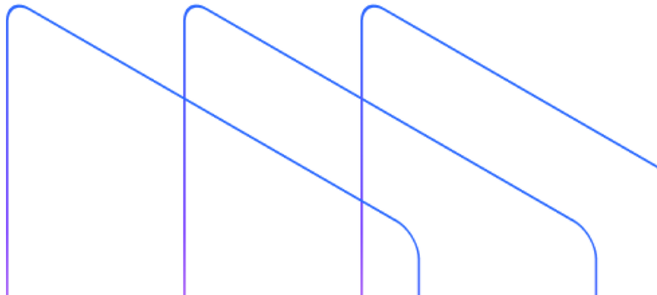
Example

```
from gptcache.processor.pre import last_content

cache.init(pre_embedding_func=last_content)

from
gptcache.processor.context.summarization_context
import SummarizationContextProcess

context_process = SummarizationContextProcess()
cache.init(pre_embedding_func=context_process.pre
_process)
```



Embedding

提取用户请求中的语义信息，输出为向量，用于后续相似搜索

文本 Embedding

Onnx
OpenAI
Cohere
SBERT
FastText

图片 Embedding

Timm
ViT

音频 Embedding

Data2VecAudio

第三方 Embedding

Huggingface
LangChain

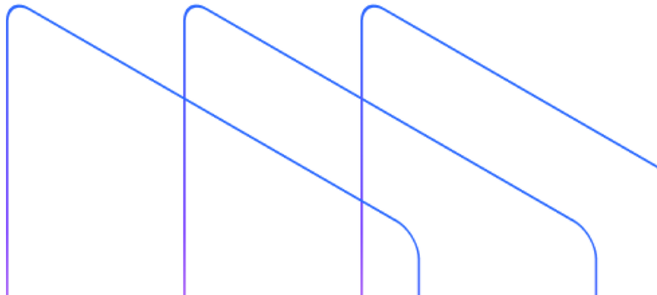
Example

```
from gptcache.embedding import Huggingface
from gptcache import cache

huggingface = Huggingface(model='uer/albert-base-chinese-cluecorpussmall')
cache.init(
    embedding_func=huggingface.to_embeddings,
    )
```

注意事项

1. 不同的语言有些模型是不适用的
2. 更改模型后，需要将之前的缓存文件删除
3. Embedding模型的维度注意与向量数据库进行关联



Cache Manager

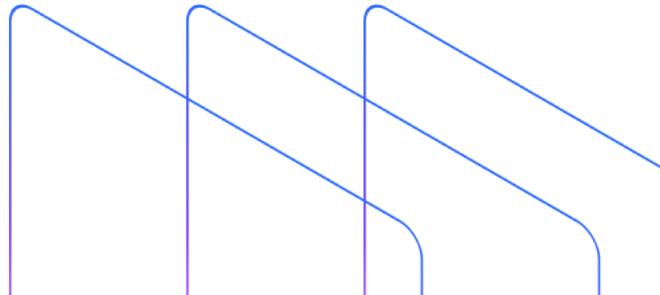
管理缓存数据，包括了存储、搜索、清理。使用传统数据库存储结构化数据，即字符串、数字等；使用向量数据库存储非结构化数据，主要是文本、图片、音频等通过 Embedding 模型得到的向量；使用对象存储存储文件类型数据，即图片、音频等。

传统数据库	向量数据库	对象存储
SQLite	Milvus	Local Storage
DuckDB	Zilliz Cloud	S3
PostgreSQL	Faiss	...
MySQL	Hnswlib	
MariaDB	PGVector	
SQL Server	Chroma	
Oracle	DocArray	
...	...	

Example

```
from gptcache.embedding import Huggingface
from gptcache.manager import manager_factory
from gptcache import cache

huggingface = Huggingface(model='uer/albert-base-chinese-cluecorpussmall')m = manager_factory("sqlite,faiss,local", data_dir="./dolly", vector_params={"dimension": huggingface.dimension})
cache.init(
    data_manager=m,
    huggingface = Huggingface(model='uer/albert-base-chinese-cluecorpussmall')
)
```



Similarity Evaluation

评估相似结果是否达到预期，将用户输入与从 cache 中相似搜索得到的结果进行相似评估，得到一个数值，然后与用户设定的相似阈值进行比较。这样可以有效保证缓存结果质量，以满足响应用户请求的要求。

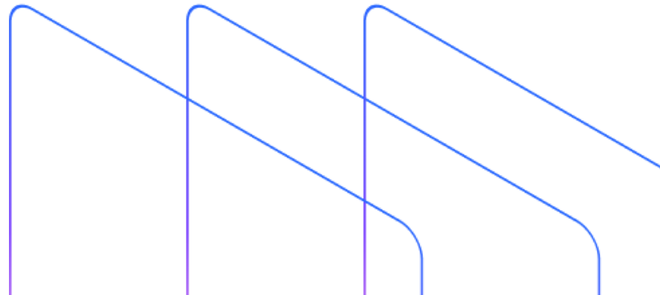
Evaluator

SearchDistanceEvaluation
ExactMatchEvaluation
NumpyNormEvaluation
OnnxModelEvaluation
KReciprocalEvaluation
...

Example

```
from gptcache import cache
from gptcache.similarity_evaluation.distance import
SearchDistanceEvaluation

evaluation = SearchDistanceEvaluation()
cache.init(
      similarity_evaluation=evaluation,
  )
```



Post-Processor

对符合相似阈值的结果列表进行处理，如果想保证cache并不是每次都一样，则可以这个组件进行二次开发。

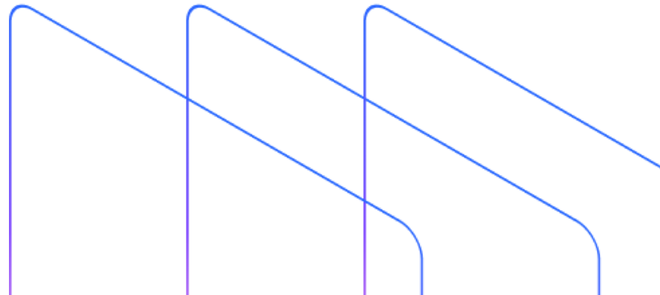
基础内置实现

first
random_one
temperature_softmax
nop

Example

```
from gptcache import cache
from gptcache.processor.post import first

cache.init(
        post_process_messages_func=first
    )
```



更多

Cache 初始化配置

similarity_threshold
log_time_func
prompts

Example

```
from gptcache import Config, cache

configs = Config(similarity_threshold=0.6)
cache.init(config=config)
```

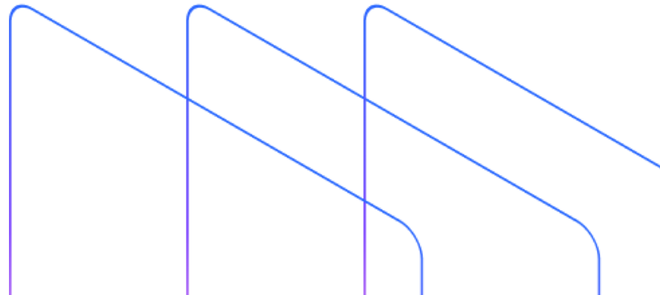
Cache 请求参数

cache_obj
cache_context
cache_skip
session
temperature

Example

```
from gptcache import Cache
from gptcache.adapter import openai

one_cache = Cache()
one_cache.init()
openai.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=[
        {"role": "user", "content": question}
    ],
    cache_obj=one_cache
)
```

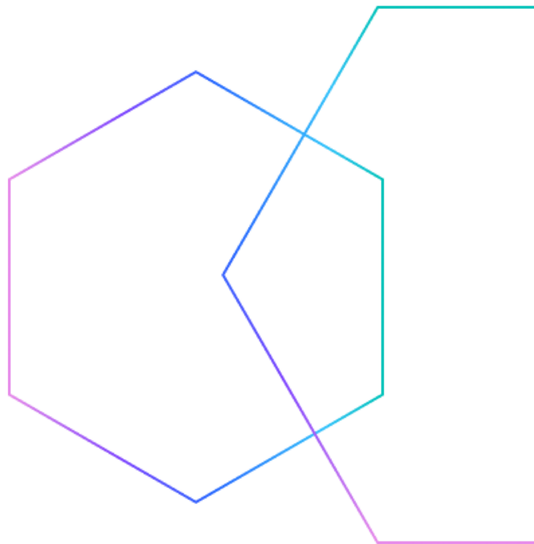


03

基本使用

基本使用

1. GPTCache 与内置集成 LLM 模型
2. GPTCache 与自定义 LLM 模型
3. GPTCache 与 LangChain
4. GPTCache 与 LlamaIndex



GPTCache 与内置集成 LLM 模型

使用 GPTCache，初始化后即可体验，其初始化过程中涉及到的各个组件可根据自身要求进行组装，接入 LLM 则只需要直接替换包名即可，更多LLM例子

Example

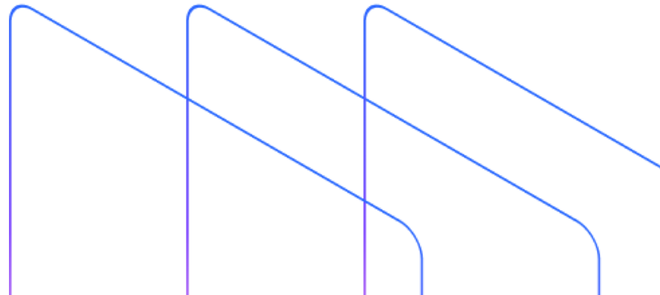
```
from gptcache.adapter.api import init_similar_cache
from gptcache.processor.pre import get_prompt
from gptcache.adapter.adapter import openai

init_similar_cache(pre_func=last_content)

response = openai.ChatCompletion.create(
      model='gpt-3.5-turbo',
      messages=[
        {
            'role': 'user',
            'content': question
        }
      ],
  )
```

Cache.init

```
class Cache:
    def init(
        self,
        cache_enable_func=cache_all,
        pre_embedding_func=last_content,
        embedding_func=string_embedding,
        data_manager: DataManager = get_data_manager(),
        similarity_evaluation=ExactMatchEvaluation(),
        post_process_messages_func=temperature_softmax,
        config=Config(),
        next_cache=None,
    ):
        pass
```



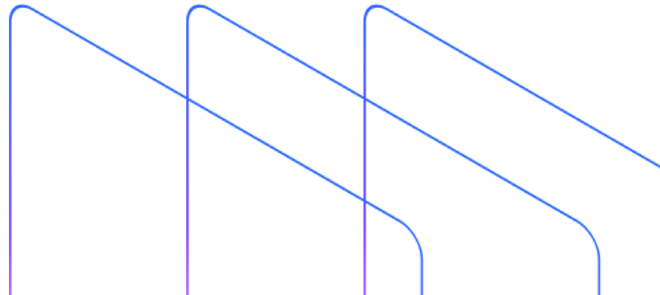
GPTCache 与自定义 LLM 模型

对于目前没有集成至 GPTCache 的大模型，可以通过 api 使用 GPTCache 的能力

Example

```
from gptcache.adapter.api import put, get, init_similar_cache

init_similar_cache()
put("hello", "foo")
print(get("hello"))
```



GPTCache 与 LangChain

LangChain 是旨在帮助开发人员将大型语言模型与其他计算或知识源结合使用，以构建功能强大的 Python 库。提供了一套工具和资源，用于构建问答、聊天机器人和代理等 LLM 应用程序。

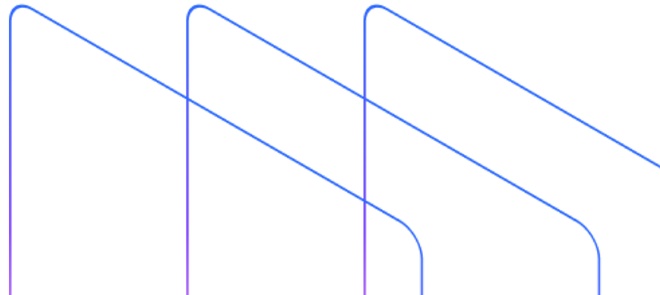
LangChain 提供了一个全局的 LLM_cache 对象，所以只需要将这个对象赋值成 GPTCache 对象即可，就可以使用 GPTCache 相关功能。

Example

```
from gptcache import Cache
from gptcache.adapter.api import init_similar_cache
from langchain.cache import GPTCache

def init_gptcache(cache_obj: Cache, llm str):
    init_similar_cache(cache_obj=cache_obj, data_dir=f"similar_cache_{llm}")

langchain.llm_cache = GPTCache(init_gptcache)
```



GPTCache 与 LlamaIndex

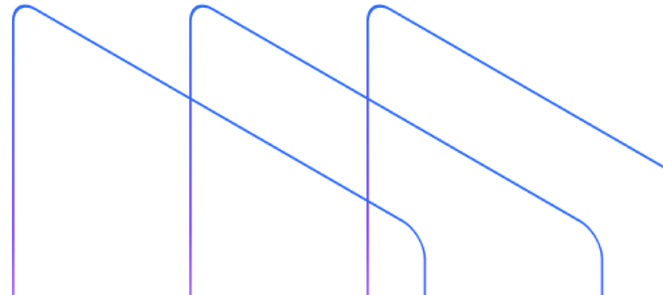
LlamaIndex 是一种数据索引工具，用于将私有数据与预先训练好的 LLM 模型相结合，实现在特定上下文中学习。
它提供数据连接器、索引和查询接口等工具，以便高效地将非结构化和结构化数据与 LLM 相结合，实现知识增强。

Example

```
from gptcache import Cache
from gptcache.adapter.api import init_similar_cache
from langchain.cache import GPTCache
from gpt_index.llm_predictor.structured import LLMPredictor
from gpt_index.prompts.default_prompts import DEFAULT_SIMPLE_INPUT_PROMPT

def init_gptcache(cache_obj: Cache, llm str):
    init_similar_cache(cache_obj=cache_obj, data_dir=f"similar_cache_{llm}")

predictor = LLMPredictor(llm, False, GPTCache(init_gptcache))
prompt = DEFAULT_SIMPLE_INPUT_PROMPT
llm_prediction, formatted_output = predictor.predict(
        prompt, query_str="hello world"
    )
```



更多

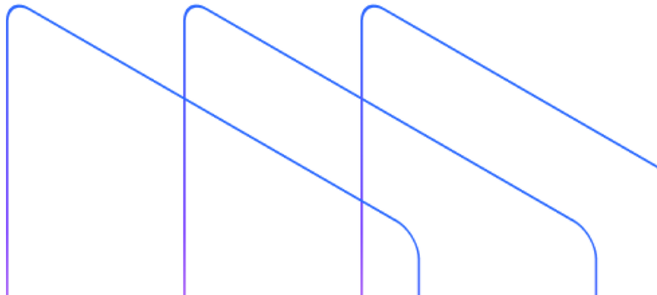
更多**使用案例**，可以参考 Bootcamp

针对**其他语言**，也可以通过 docker image 使用 GPTCache

启动 GPTCache Server
docker pull zilliz/gptcache:latest
docker run –p 8000:8000 –it zilliz/gptcache:latest

使用GPTCache Server
curl –X PUT –d "receive a hello message" "http://localhost:8000?prompt=hello"
curl –X GET "http://localhost:8000?prompt=hello"

更多信息参考：如何使用GPTCache Server

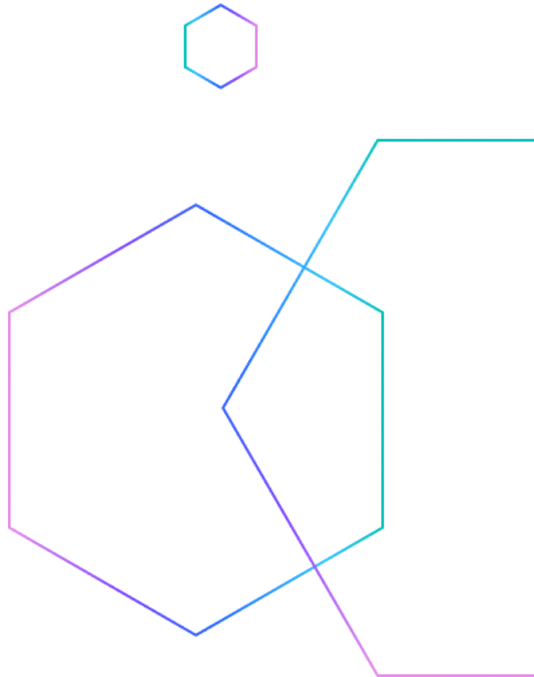


04

应用

应用

1. 私有化模型
2. GPT 应用
3. 多模态场景



私有化模型

安全

虽然 ChatGPT 的效果很好，但是使用 ChatGPT 时，数据将会被发送给 OpenAI 公司，这些数据可能会被模型用于回答其他用户的问题，从而导致数据泄漏。这种情况 OpenAI 无法完全杜绝，因此为了确保数据安全，许多 LLM 使用者开始考虑使用开源大模型进行私有化部署。

privateGPT：https://github.com/imartinez/privateGPT

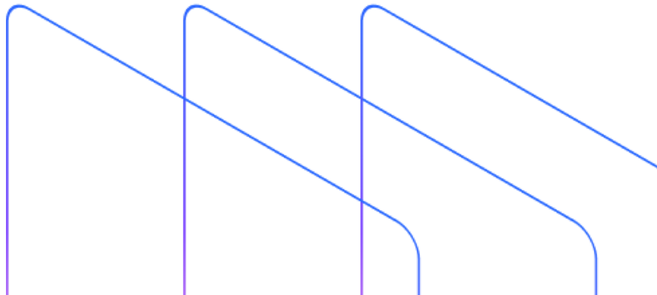
开源模型列表：https://github.com/eugeneyan/open-llms

成本

目前各个在线 LLM 模型基本上都是通过 token 数进行收费，长期以往，对于个人和企业也将是额外一笔开支，如果进行私有化部署，也将一定程度降低使用成本。

质量

目前 ChatGPT 其问题回答质量并不稳定，这与其训练数据相关，例如训练数据都是与生活，让其回答宇宙相关的问题，得到的答案则很有可能会胡乱捏造的。关于这部分，GitHub 上也有很多相关某一垂直领域资源，如医学相关的华驼、法律相关的 LawGPT 等，当然也可以搭建本地知识库，如公司内问答人、客服机器人、文档助手等。



私有化模型

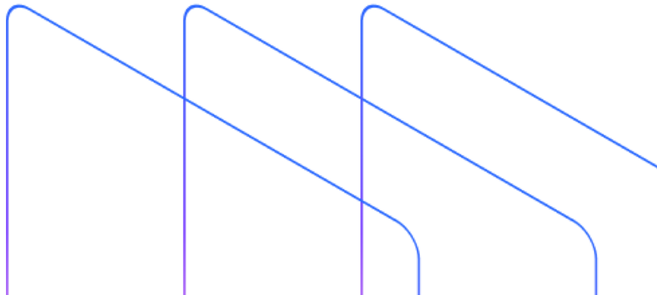
为什么要使用 GPTCache?

1. **接入成本极低。**使用内置模型只需两步：初始化和替换模型 SDK 的 import。

2. **降低 LLM 使用成本。**通过使用缓存获取相似问题的答案，可以减少 LLM 请求次数，从而降低服务器资源压力。

3. **减少响应时延。**相似问题的响应时延只需毫秒级别。如果对于缓存返回的结果不满意，支持绕过缓存重新生成。

4. **高度定制化。**可以根据业务场景组装合适的缓存，例如对于小数据量，只需要使用本地存储模型，例如SQLite + Faiss；对于大数据量场景，可以使用 MySQL/Oracle + Zilliz Cloud/Milvus。除了存储方面，其他模块也可以根据需要进行调整，例如预处理、Embedding、相似度评估和后处理等。



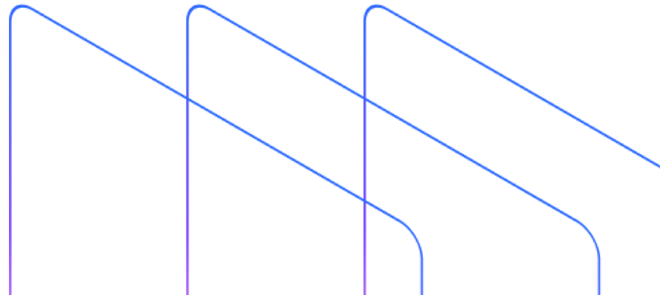
GPT应用

随着 ChatGPT 的出现，基于其 API 进行桌面应用或在线服务的开发也逐渐增多，这些应用可以降低用户使用门槛，同时也提高了 ChatGPT 的使用体验，例如通过定制化 UI、prompt 管理、数据导出等。

GPTCache 也适用于这种场景，它给个人用户带来的最直接好处是**成本降低💰和加速**🚀。如果缓存每次问答的结果，并达到一定的数据量，则可以离线处理任务。此外，对于应用服务提供者，根据用户画像对缓存数据进行精细化管理，可以使服务突破 LLM 模型的性能瓶颈，进一步改善服务的稳定性。

推荐资源列表

awesome-chatgpt-zh
awesome-open-gpt

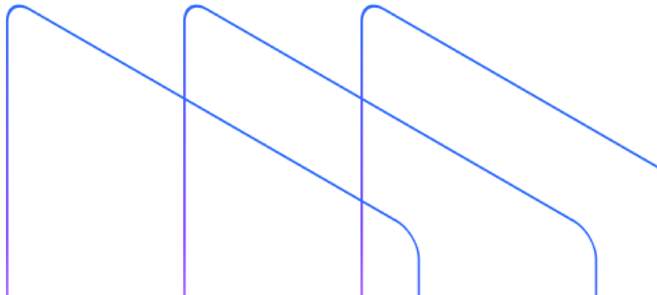


多模态场景

随着 ChatGPT 技术的不断普及和应用拓展，AIGC（内容生成）成为了其中一个重要的方向，包括文本生成、图片生成和语音生成等多个领域，这些技术的发展将进一步促进人工智能在各个领域的应用。

最新版的 GPTCache 已经可以支持多模态场景，包括了文本生成文本、文本生成图片、图文问答、文本生成语音等，更多例子参考 Bootcamp。

将 GPTCache 应用于这一场景中，最具吸引力的是它可以显著降低时延。然而，这种方法也存在一些问题。例如，有时候即使输入相似，也不一定需要相同的答案。为了解决这个问题，可以跳过缓存并重新生成模型，或者在获取缓存答案后，使用小模型进行微调。与重新生成模型相比，小模型微调也可以有效降低响应时间。因此，在实际应用场景中，需要根据具体情况来判断 GPTCache 是否会带来收益，并决定如何组合 GPTCache 的组件。



THANKS FOR WATCHING

Github

公众号

扫码并回复“技术交流”
加入用户交流群

https://github.com/zilliztech/GPTCache

https://zilliz.com

zilliz