Personal AI

Skoltech Applied AI Lab

нояб. 15, 2024

Оглавление

В ноутбуках ниже представлены примеры использования QA- и Mem-конвейеров с реализованными алгоритмами поиска информации в памяти (графе знаний) персонального ассистента.

ГЛАВА 1

Пример запуска персонального AI-ассистента с A*-алгоритмом поиска

[1]:
```
import sys
# TO CHANGE
BASEDIR = "../../../"
sys.path.insert(0, BASEDIR)
```

[ ]:
```
from src import PersonalAI, PersonalAIConfig, QAPipelineConfig,␣
↪MemPipelineConfig, \
      GraphModelConfig, EmbeddingsModelConfig, EmbedderModelConfig

from src.db_drivers import KeyValueDriverConfig, GraphDriverConfig,␣
↪VectorDriverConfig
from src.db_drivers.kv_driver import DEFAULT_INMEMORYKV_CONFIG
from src.db_drivers.graph_driver import DEFAULT_INMEMORYGRAPH_CONFIG
from src.db_drivers.vector_driver import VectorDBConnectionConfig

from src.qa_pipeline.knowledge_retriever import AStarGraphSearchConfig,␣
↪AStarMetricsConfig
from src.qa_pipeline import QueryLLMParserConfig, KnowledgeComparatorConfig,␣
↪KnowledgeRetrieverConfig, QALLMGeneratorConfig

from src.memorize_pipeline import LLMExtractorConfig, LLMUpdatorConfig

from src.utils import NodeType, Logger
```

1.1 1. Задаём конфигурацию графа знаний

Для создания объекта класса PersonalAI (верхнеуровневый класс персонального ассистента) необходимо выполнить конфигурацию трёх основных его компонент: * Графа знаний, который будет выступать в роли памяти ассистента и хранить поступающую информацию. * QA-конвейера, с помощью которого ассистент будет отвечать

на user-вопросы. * Memorize-конвейера, с помощью которого ассистент будет добавлять новую информацию в память и выполнять её актуализацию.

Граф знаний состоит из двух моделей хранения информации: векторной и графовой. Соответственно нам потребуется инициализировать два конфигурационных data-класса: `GraphModelConfig` и `EmbeddingModelConfig`.

В конфиге для графовой модели потребуется указать параметры подключения к одной из доступных графовых бд. В нашем случае мы подключаемся к самописной модели, которая хранить граф в оперативной памяти.

```
[ ]: # Graph model configuration
GRAPH_STORAGE_CONFIG = GraphDriverConfig(db_vendor='inmemory_graph', db_
 ↪config=DEFAULT_INMEMORYGRAPH_CONFIG)
GRAPH_MODEL_CONFIG = GraphModelConfig(driver_config=GRAPH_STORAGE_CONFIG)
```

В конфиге для векторной модели потребуется отдельно указать параметры подключения к двум векторным бд, где будут храниться эмбеддинги триплетов и вершин из графовой модели соответственно.

```
[ ]: # Vector model configuration
NODES_DB_PATH = '../../data/graph_structures/vectorized_nodes/testing'
TRIPLETS_DB_PATH = '../../data/graph_structures/vectorized_triplets/testing'
NEED_TO_CLEAR = True

VECTOR_NODES_STORAGE_CONFIG = VectorDriverConfig(db_
 ↪config=VectorDBConnectionConfig(path=NODES_DB_PATH, need_to_clear=NEED_TO_
 ↪CLEAR))
VECTOR_TRIPLETS_STIRAGE_CONFIG = VectorDriverConfig(db_
 ↪config=VectorDBConnectionConfig(path=TRIPLETS_DB_PATH, need_to_clear=NEED_
 ↪TO_CLEAR))
```

Также потребуется указать параметры для инициализации класса, с помощью которого будет выполняться перевод текста в его векторное представления (полечение эмбеддингов).

```
[ ]: DEVICE = 'cuda'
EMBEDDER_MODEL_PATH = '../../models/intfloat/multilingual-e5-small'
EMBEDDER_MODEL_CONFIG = EmbedderModelConfig(model_name_or_path=EMBEDDER_MODEL_
 ↪PATH, device=DEVICE)
```

```
[ ]: VECTOR_MODEL_CONFIG = EmbeddingsModelConfig(
    nodesdb_driver_config=VECTOR_NODES_STORAGE_CONFIG,
    tripletsdb_driver_config=VECTOR_TRIPLETS_STIRAGE_CONFIG,
    embedder_config=EMBEDDER_MODEL_CONFIG)
```

Далее нам потребуется сконфигурировать QA- и Memorize-пайплайны.

В рамках QA-конфигурации мы задаём алгоритм поиска, с помощью которого будет извлекаться информации из графа знаний. В нашем случае это будет A*-алгоритм. Для этого нам потребуется указать его идентификатор (в виде ключевого слова) и соответствующий ему data-класс `AStarGraphSearchConfig` с заданными гиперпараметрами.

```
[ ]: RETRIEVER_NAME = 'astar'

KV_STORAGE_CONFIG = KeyValueDriverConfig(db_vendor='inmemory_kv', db_
 ↪config=DEFAULT_INMEMORYKV_CONFIG)
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
RETRIEVER_HYPERP = AStarGraphSearchConfig(
    metrics_config=AStarMetricsConfig(h_metric_name='ip',kvdriver_config=KV_
↪STORAGE_CONFIG),
    max_depth=-1, max_passed_nodes=-1,
    accepted_node_types=[NodeType.object , NodeType.hyper, NodeType.episodic]
)
```

Для QueryParser- и QAGenerator- стадий мы указываем язык, который будет использоваться для генерации необходимой информации с помощью LLM-агента. В рамках библиотеки прддерживается два языка: русский и английский. Мы будем использовать английский.

[2]:
```
LANGUAGE = 'en'
```

[ ]:
```
# QA-pipeline configuration
QA_PIPELINE_CONFIG = QAPipelineConfig(
    query_parser_config=QueryLLMParserConfig(lang=LANGUAGE),
    knowledge_comparator_config=KnowledgeComparatorConfig(),
    knowledge_retriever_config=KnowledgeRetrieverConfig(
        retriever_method=RETRIEVER_NAME,retriever_config=RETRIEVER_HYPERP),
    answer_generator_config=QALLMGeneratorConfig(lang=LANGUAGE))
```

В рамках Memorize-конфигурации мы задаём только язык обрабатываемого текста, чтобы на шаге запуска LLM-агента выбрать болен оптимальные промпты и функцию парсинга ответа.

[ ]:
```
# Memorize-pipeline configuration
MEM_PIPELINE_CONFIG = MemPipelineConfig(
    xtractor_config=LLMExtractorConfig(lang=LANGUAGE),
    updator_config=LLMUpdatorConfig(lang=LANGUAGE))
```

Таким образом, у нас была получена конфигурация гиперпараметров для инициализации персонального ассистена.

[5]:
```
PERSONALAI_CONFIG = PersonalAIConfig(
    graph_struct_config=GRAPH_MODEL_CONFIG,
    embedds_struct_config=VECTOR_MODEL_CONFIG,
    qa_pipeline_config=QA_PIPELINE_CONFIG,
    mem_pipeline_config=MEM_PIPELINE_CONFIG,
    log=Logger('log/main'))
```

1.1.1 2. Инициализируем персонального ассистента

[ ]:
```
personalai = PersonalAI(config=PERSONALAI_CONFIG)
```

1.1.2 3. Пример работы Memorize-конвейера

Подготавливаем набор текстов на естественном языке (в нашем случае на английском) для их последующего сохранения в память ассистента.

[8]:
```
messages = [
    "Mikhail Menshchikov is currently a second-year master's student at ITMO.
↪",
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
    "Mikhail Menshchikov is studying in the Master's program 'Deep Learning␣
↪and Generative AI'",
    "Mikhail Menshchikov completed his bachelor's degree at Petrozavodsk␣
↪State University",
    "Petrozavodsk State University is where Mikhail Menshchikov received his␣
↪bachelor's degree.",
    "Mikhail Menshchikov studied at Petrozavodsk State University and␣
↪received a bachelor's degree."]
properties = [dict() for _ in range(len(messages))]
```

С помощью метода update_memory сохраняем сформированный набор информации в память.

```
[ ]: for text, prop in zip(messages, properties):
        _, info = personalai.update_memory(text, prop)
```

1.1.3 4. Пример работы QA-конвейера

Теперь мы можем задавать вопросы ассистенту по имеющимся у него знаниях в памяти и получать ответы.

```
[10]: answer, info = personalai.answer_question("What program is Mikhail␣
↪Menshchikov studying for his master's degree?")
print(answer)
```

```
Deep Learning and Generative AI
```

```
[11]: answer, info = personalai.answer_question("Where did Mikhail Menshchikov␣
↪receive his bachelor's degree?")
print(answer)
```

```
Petrozavodsk State University
```

ГЛАВА 2

Пример запуска персонального AI-ассистента с BFS-алгоритмом поиска

[1]:
```
import sys
# TO CHANGE
BASEDIR = "../../../"
sys.path.insert(0, BASEDIR)
```

[ ]:
```
from src import PersonalAI, PersonalAIConfig, QAPipelineConfig,␣
↪MemPipelineConfig, \
      GraphModelConfig, EmbeddingsModelConfig, EmbedderModelConfig

from src.db_drivers import GraphDriverConfig, VectorDriverConfig
from src.db_drivers.graph_driver import DEFAULT_INMEMORYGRAPH_CONFIG
from src.db_drivers.vector_driver import VectorDBConnectionConfig

from src.qa_pipeline.knowledge_retriever import BFSSearchConfig
from src.qa_pipeline import QueryLLMParserConfig, KnowledgeComparatorConfig,␣
↪KnowledgeRetrieverConfig, QALLMGeneratorConfig

from src.memorize_pipeline import LLMExtractorConfig, LLMUpdatorConfig

from src.utils import Logger
```

2.1 1. Задаём конфигурацию графа знаний

Для создания объекта класса PersonalAI (верхнеуровневый класс персонального ассистента) необходимо выполнить конфигурацию трёх основных его компонент: * Графа знаний, который будет выступать в роли памяти ассистента и хранить поступающую информацию. * QA-конвейера, с помощью которого ассистент будет отвечать на user-вопросы. * Memorize-конвейера, с помощью которого ассистент будет добавлять новую информацию в память и выполнять её актуализацию.

Граф знаний состоит из двух моделей хранения информации: векторной и графовой. Соответственно нам потребуется инициализировать два конфигурационных data-класса: `GraphModelConfig` и

`EmbeddingModelConfig`.

В конфиге для графовой модели потребуется указать параметры подключения к одной из доступных графовых бд. В нашем случае мы подключаемся к самописной модели, которая хранить граф в оперативной памяти.

```
[ ]: # Graph model configuration
GRAPH_STORAGE_CONFIG = GraphDriverConfig(db_vendor='inmemory_graph', db_
 ↪config=DEFAULT_INMEMORYGRAPH_CONFIG)
GRAPH_MODEL_CONFIG = GraphModelConfig(driver_config=GRAPH_STORAGE_CONFIG)
```

В конфиге для векторной модели потребуется отдельно указать параметры подключения к двум векторным бд, где будут храниться эмбеддинги триплетов и вершин из графовой модели соответственно.

```
[ ]: # Vector model configuration
NODES_DB_PATH = '../../data/graph_structures/vectorized_nodes/testing'
TRIPLETS_DB_PATH = '../../data/graph_structures/vectorized_triplets/testing'
NEED_TO_CLEAR = True

VECTOR_NODES_STORAGE_CONFIG = VectorDriverConfig(db_
 ↪config=VectorDBConnectionConfig(path=NODES_DB_PATH, need_to_clear=NEED_TO_
 ↪CLEAR))
VECTOR_TRIPLETS_STIRAGE_CONFIG = VectorDriverConfig(db_
 ↪config=VectorDBConnectionConfig(path=TRIPLETS_DB_PATH, need_to_clear=NEED_
 ↪TO_CLEAR))
```

Также потребуется указать параметры для инициализации класса, с помощью которого будет выполняться перевод текста в его векторное представления (полечение эмбеддингов).

```
[ ]: DEVICE = 'cuda'
EMBEDDER_MODEL_PATH = '../../models/intfloat/multilingual-e5-small'
EMBEDDER_MODEL_CONFIG = EmbedderModelConfig(model_name_or_path=EMBEDDER_MODEL_
 ↪PATH, device=DEVICE)

VECTOR_MODEL_CONFIG = EmbeddingsModelConfig(
    nodesdb_driver_config=VECTOR_NODES_STORAGE_CONFIG,
    tripletsdb_driver_config=VECTOR_TRIPLETS_STIRAGE_CONFIG,
    embedder_config=EMBEDDER_MODEL_CONFIG)
```

Далее нам потребуется сконфигурировать QA- и Memorize-пайплайны.

В рамках QA-конфигурации мы задаём алгоритм поиска, с помощью которого будет извлекаться информации из графа знаний. В нашем случае это будет BFS-алгоритм. Для этого нам потребуется указать его идентификатор (в виде ключевого слова) и соответствующий ему data-класс `BFSSearchConfig` с заданными гиперпараметрами.

```
[ ]: RETRIEVER_NAME = 'bfs'
RETRIEVER_HYPERP = BFSSearchConfig(
    strict_filter = False, hyper_episodic_num = 15,
    chain_triplets_num = 25, other_triplets_num = 6)
```

Для QueryParser- и QAGenerator- стадий мы указываем язык, который будет использоваться для генерации необходимой информации с помощью LLM-агента. В рамках библиотеки прддерживается два языка: русский и английский. Мы будем использовать английский.

```
[ ]: LANGUAGE = 'en'
```

[ ]:
```
# QA-pipeline configuration
QA_PIPELINE_CONFIG = QAPipelineConfig(
    query_parser_config=QueryLLMParserConfig(lang=LANGUAGE),
    knowledge_comparator_config=KnowledgeComparatorConfig(),
    knowledge_retriever_config=KnowledgeRetrieverConfig(
        retriever_method=RETRIEVER_NAME,retriever_config=RETRIEVER_HYPERP),
    answer_generator_config=QALLMGeneratorConfig(lang=LANGUAGE))
```

В рамках Memorize-конфигурации мы задаём только язык обрабатываемого текста, чтобы на шаге запуска LLM-агента выбрать болен оптимальные промпты и функцию парсинга ответа.

[ ]:
```
# Memorize-pipeline configuration
MEM_PIPELINE_CONFIG = MemPipelineConfig(
    xtractor_config=LLMExtractorConfig(lang=LANGUAGE),
    updator_config=LLMUpdatorConfig(lang=LANGUAGE))
```

Таким образом, у нас была получена конфигурация гиперпараметров для инициализации персонального ассистена.

[5]:
```
PERSONALAI_CONFIG = PersonalAIConfig(
    graph_struct_config=GRAPH_MODEL_CONFIG,
    embedds_struct_config=VECTOR_MODEL_CONFIG,
    qa_pipeline_config=QA_PIPELINE_CONFIG,
    mem_pipeline_config=MEM_PIPELINE_CONFIG,
    log=Logger('log/main'))
```

2.1.1 2. Инициализируем персонального ассистента

[ ]:
```
personalai = PersonalAI(config=PERSONALAI_CONFIG)
```

2.1.2 3. Пример работы Memorize-конвейера

Подготавливаем набор текстов на естественном языке (в нашем случае на английском) для их последующего сохранения в память ассистента.

[8]:
```
messages = [
    "Mikhail Menshchikov is currently a second-year master's student at ITMO.
↪",
    "Mikhail Menshchikov is studying in the Master's program 'Deep Learning␣
↪and Generative AI'",
    "Mikhail Menshchikov completed his bachelor's degree at Petrozavodsk␣
↪State University",
    "Petrozavodsk State University is where Mikhail Menshchikov received his␣
↪bachelor's degree.",
    "Mikhail Menshchikov studied at Petrozavodsk State University and␣
↪received a bachelor's degree."]
properties = [dict() for _ in range(len(messages))]
```

С помощью метода update_memory сохраняем сформированный набор информации в память.

[ ]:
```
for text, prop in zip(messages, properties):
    _, info = personalai.update_memory(text, prop)
```

2.1.3 4. Пример работы QA-конвейера

Теперь мы можем задавать вопросы ассистенту по имеющимся у него знаниях в памяти и получать ответы.

[10]:
```
answer, info = personalai.answer_question("What program is Mikhail␣
↪Menshchikov studying for his master's degree?")
print(answer)
```

```
Deep Learning and Generative AI
```

[11]:
```
answer, info = personalai.answer_question("Where did Mikhail Menshchikov␣
↪receive his bachelor's degree?")
print(answer)
```

```
Petrozavodsk State University
```

ГЛАВА 3

Пример запуска персонального AI-ассистента с Mixtured-алгоритмом поиска

[1]:
```
import sys
# TO CHANGE
BASEDIR = "../../../"
sys.path.insert(0, BASEDIR)
```

[ ]:
```
from src import PersonalAI, PersonalAIConfig, QAPipelineConfig,␣
↪MemPipelineConfig, \
      GraphModelConfig, EmbeddingsModelConfig, EmbedderModelConfig

from src.db_drivers import KeyValueDriverConfig, GraphDriverConfig,␣
↪VectorDriverConfig
from src.db_drivers.kv_driver import DEFAULT_INMEMORYKV_CONFIG
from src.db_drivers.graph_driver import DEFAULT_INMEMORYGRAPH_CONFIG
from src.db_drivers.vector_driver import VectorDBConnectionConfig

from src.qa_pipeline.knowledge_retriever import AStarGraphSearchConfig,␣
↪AStarMetricsConfig, BFSSearchConfig, MixturedGraphSearchConfig
from src.qa_pipeline import QueryLLMParserConfig, KnowledgeComparatorConfig,␣
↪KnowledgeRetrieverConfig, QALLMGeneratorConfig

from src.memorize_pipeline import LLMExtractorConfig, LLMUpdatorConfig

from src.utils import NodeType, Logger
```

3.1 1. Задаём конфигурацию графа знаний

Для создания объекта класса PersonalAI (верхнеуровневый класс персонального ассистента) необходимо выполнить конфигурацию трёх основных его компонент: * Графа знаний, который будет выступать в роли памяти ассистента и хранить поступающую информацию. * QA-конвейера, с помощью которого ассистент будет отвечать

на user-вопросы. * Memorize-конвейера, с помощью которого ассистент будет добавлять новую информацию в память и выполнять её актуализацию.

Граф знаний состоит из двух моделей хранения информации: векторной и графовой. Соответственно нам потребуется инициализировать два конфигурационных data-класса: `GraphModelConfig` и `EmbeddingModelConfig`.

В конфиге для графовой модели потребуется указать параметры подключения к одной из доступных графовых бд. В нашем случае мы подключаемся к самописной модели, которая хранить граф в оперативной памяти.

```
[ ]: # Graph model configuration
GRAPH_STORAGE_CONFIG = GraphDriverConfig(db_vendor='inmemory_graph', db_
 ↪config=DEFAULT_INMEMORYGRAPH_CONFIG)
GRAPH_MODEL_CONFIG = GraphModelConfig(driver_config=GRAPH_STORAGE_CONFIG)
```

В конфиге для векторной модели потребуется отдельно указать параметры подключения к двум векторным бд, где будут храниться эмбеддинги триплетов и вершин из графовой модели соответственно.

```
[ ]: # Vector model configuration
NODES_DB_PATH = '../../data/graph_structures/vectorized_nodes/testing'
TRIPLETS_DB_PATH = '../../data/graph_structures/vectorized_triplets/testing'
NEED_TO_CLEAR = True

VECTOR_NODES_STORAGE_CONFIG = VectorDriverConfig(db_
 ↪config=VectorDBConnectionConfig(path=NODES_DB_PATH, need_to_clear=NEED_TO_
 ↪CLEAR))
VECTOR_TRIPLETS_STIRAGE_CONFIG = VectorDriverConfig(db_
 ↪config=VectorDBConnectionConfig(path=TRIPLETS_DB_PATH, need_to_clear=NEED_
 ↪TO_CLEAR))
```

Также потребуется указать параметры для инициализации класса, с помощью которого будет выполняться перевод текста в его векторное представления (полечение эмбеддингов).

```
[ ]: DEVICE = 'cuda'
EMBEDDER_MODEL_PATH = '../../models/intfloat/multilingual-e5-small'
EMBEDDER_MODEL_CONFIG = EmbedderModelConfig(model_name_or_path=EMBEDDER_MODEL_
 ↪PATH, device=DEVICE)
```

```
[ ]: VECTOR_MODEL_CONFIG = EmbeddingsModelConfig(
    nodesdb_driver_config=VECTOR_NODES_STORAGE_CONFIG,
    tripletsdb_driver_config=VECTOR_TRIPLETS_STIRAGE_CONFIG,
    embedder_config=EMBEDDER_MODEL_CONFIG)
```

Далее нам потребуется сконфигурировать QA- и Memorize-пайплайны.

В рамках QA-конфигурации мы задаём алгоритм поиска, с помощью которого будет извлекаться информации из графа знаний. В нашем случае это будет Mixtured-алгоритм. Для этого нам потребуется указать его идентификатор (в виде ключевого слова) и соответствующий ему data-класс `MixturedGraphSearchConfig` с заданными гиперпараметрами. Так как Mixtured-алгоритм является комбнацией A*- и BFS-алгоритм нам также потребуется задать их конфигурации с помощью `AStarGraphSearchConfig`- и `BFSSearchConfig`-классов соответвенно.

```
[ ]: KV_STORAGE_CONFIG = KeyValueDriverConfig(db_vendor='inmemory_kv', db_
 ↪config=DEFAULT_INMEMORYKV_CONFIG)
ASTAR_RETRIEVER_CONFIG = AStarGraphSearchConfig(
```

(продолжается на следующей странице)

(продолжение с предыдущей страницы)

```
    metrics_config=AStarMetricsConfig(h_metric_name='ip', kvdriver_config=KV_
↪STORAGE_CONFIG),
    max_depth=20, max_passed_nodes=1000,
    accepted_node_types=[NodeType.object , NodeType.hyper, NodeType.episodic])

BFS_RETRIEVER_CONFIG = BFSSearchConfig(
    strict_filter = True, hyper_episodic_num = 15,
    chain_triplets_num = 25, other_triplets_num = 6)
```

[ ]:
```
RETRIEVER_NAME = 'mixture'
RETRIEVER_CONFIG = MixturedGraphSearchConfig(
    astar_config=ASTAR_RETRIEVER_CONFIG,
    bfs_config=BFS_RETRIEVER_CONFIG
)
```

Для QueryParser- и QAGenerator- стадий мы указываем язык, который будет использоваться для генерации необходимой информации с помощью LLM-агента. В рамках библиотеки прддерживается два языка: русский и английский. Мы будем использовать английский.

[ ]:
```
LANGUAGE = 'auto'
```

[ ]:
```
# QA-pipeline configuration
QA_PIPELINE_CONFIG = QAPipelineConfig(
    query_parser_config=QueryLLMParserConfig(lang=LANGUAGE),
    knowledge_comparator_config=KnowledgeComparatorConfig(),
    knowledge_retriever_config=KnowledgeRetrieverConfig(
        retriever_method=RETRIEVER_NAME,retriever_config=RETRIEVER_CONFIG),
    answer_generator_config=QALLMGeneratorConfig(lang=LANGUAGE))
```

В рамках Memorize-конфигурации мы задаём только язык обрабатываемого текста, чтобы на шаге запуска LLM-агента выбрать болен оптимальные промпты и функцию парсинга ответа.

[ ]:
```
# Memorize-pipeline configuration
MEM_PIPELINE_CONFIG = MemPipelineConfig(
    xtractor_config=LLMExtractorConfig(lang=LANGUAGE),
    updator_config=LLMUpdatorConfig(lang=LANGUAGE))
```

Таким образом, у нас была получена конфигурация гиперпараметров для инициализации персонального ассистена.

[5]:
```
PERSONALAI_CONFIG = PersonalAIConfig(
    graph_struct_config=GRAPH_MODEL_CONFIG,
    embedds_struct_config=VECTOR_MODEL_CONFIG,
    qa_pipeline_config=QA_PIPELINE_CONFIG,
    mem_pipeline_config=MEM_PIPELINE_CONFIG,
    log=Logger('log/main'))
```

3.2 2. Инициализируем персонального ассистента

[ ]:
```
personalai = PersonalAI(config=PERSONALAI_CONFIG)
```

3.3 3. Пример работы Memorize-конвейера

Подготавливаем набор текстов на естественном языке (в нашем случае на английском) для их последующего сохранения в память ассистента.

[8]:
```
messages = [
    "Mikhail Menshchikov is currently a second-year master's student at ITMO.
↪",
    "Mikhail Menshchikov is studying in the Master's program 'Deep Learning␣
↪and Generative AI'",
    "Mikhail Menshchikov completed his bachelor's degree at Petrozavodsk␣
↪State University",
    "Petrozavodsk State University is where Mikhail Menshchikov received his␣
↪bachelor's degree.",
    "Mikhail Menshchikov studied at Petrozavodsk State University and␣
↪received a bachelor's degree."]
properties = [dict() for _ in range(len(messages))]
```

С помощью метода update_memory сохраняем сформированный набор информации в память.

[ ]:
```
for text, prop in zip(messages, properties):
    _, info = personalai.update_memory(text, prop)
```

3.4 4. Пример работы QA-конвейера

Теперь мы можем задавать вопросы ассистенту по имеющимся у него знаниях в памяти и получать ответы.

[10]:
```
answer, info = personalai.answer_question("What program is Mikhail␣
↪Menshchikov studying for his master's degree?")
print(answer)
```

```
Deep Learning and Generative AI
```

[11]:
```
answer, info = personalai.answer_question("Where did Mikhail Menshchikov␣
↪receive his bachelor's degree?")
print(answer)
```

```
Petrozavodsk State University
```

ГЛАВА 4

Indices and tables

- genindex
- modindex
- search