Personal AI

Skoltech Applied AI Lab

нояб. 15, 2024

Contents:

ГЛАВА 1

Высокоуровневая архитектура

1.1 QA-конвейер

Общий конвейер по извлечению релевантной информации из графа знаний для генерации ответов на вопросы представлен на Рисунке 1.

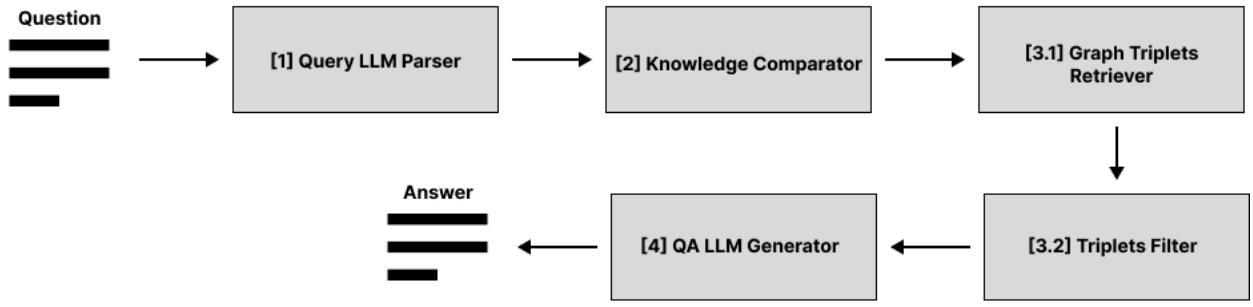


Рис. 1: Рисунок 1. Конвейер поиска информации в графе для генерации ответов на вопросы

Из Рисунка 1 видно, что конвейер разбит на четыре стадии, при этом третья стадия состоит из двух этапов. На вход конвейеру поступает пользовательский вопрос в строковом формате, который передаётся в модуль «Query LLM Parser». В данном модуле выполняется извлечение ключевых сущностей из вопроса. На второй стадии полученная информация передаётся в модуль «Knowledge Comparator» , где выполняется сопоставление (matching) сущностей из вопроса с объектами из графа знаний. На третей стадии выполняется поиск и фильтрация триплетов из графа знаний, релевантных вопросу, в рамках первого и второго этапов соответственно. На первом этапе с помощью модуля “Graph Triplets Retriever” выполняется запуск одного из реализованных алгоритмов обхода графа: в качестве стартовых выступают те вершины, которые были сопоставлены сущностям из вопроса в рамках второй стадии конвейера. В результате будет извлечён список потенциально-релевантных триплетов. На втором этапе с помощью модуля “Triplets Filter” выполняется ранжирование триплетов по степени их семантической близости к вопросу. В результате работы третей стадии будет получен список из N (гиперпараметр) триплетов, ближайших (семантически) к данному вопросу. На последней четвёртой стадии с помощью модуля «QA LLM Generator» выполняется генерация ответа на вопрос, обусловленная извлечённой информацией. Полученный ответ в строковом формате возвращается как результат работы конвейера.

Данная архитектура конвейера мотивирована следующими соображениями:

- Для получения хорошего начального приближения к подграфу с требуемой информацией необходимо сопоставить ключевые объекты из вопроса с похожими объектами из имеющегося набора знаний.
- Информация, которая требуется для генерации правильного ответа содержится в том же подграфе, что и ключевые сущности из вопроса.
- Извлечённые из графа знаний триплеты слабо обусловлены исходным вопросом. При этом каждый триплет имеет свою степень важной информации, которая требуется для генерации правильного ответа. Также у больших языковых моделей есть ограничение по максимальной длине текстовой последовательности, которую можно подать на вход.

1.2 Memorize-конвейер

Первым этапом запоминания информации является извлечение из текста триплетов с помощью ЛЛМ и специального промпта. Триплет состоит из субъекта, отношения и объекта. Например, из текста «Корабельщики в ответ: «Мы объехали весь свет; За морем житье не худо; В свете ж вот какое чудо» будут извлечены триплеты [корабельщики, объехали весь, свет], [житье, не худо за, море], [чудо, есть в, свет]. Далее субъекты и объекты триплетов становятся нодами графа знаний, а отношение между ними - простой связью в нем.

Далее по аналогичному сценарию происходит извлечение тезисов и их сущностей. Тезис, как и триплет, содержит некоторую атомарную информацию из текста, однако, в отличие от триплета, может объединять произвольное количество сущностей и всегда содержит законченную мысль, не зависящую от контекста. Например, из текста «За морем житье не худо; В свете ж вот какое чудо: Остров на море лежит, Град на острове стоит» будут извлечены тезисы «За морем житье не худо», «В свете чудо - остров на море лежит», «В свете чудо - град на острове стоит» со списками сущностей [море, житье], [свет, чудо, остров, море], [свет, чудо, град, остров] соответственно. Такая гиперсвязь также помещается в граф.

Наконец, весь исходный текст также помещается в граф как гиперсвязь между всеми сущностями, фигурирующими в извлеченных триплетах и тезисах. Такая гиперсвязь, по аналогии с когнитивными науками, называется эпизодической.

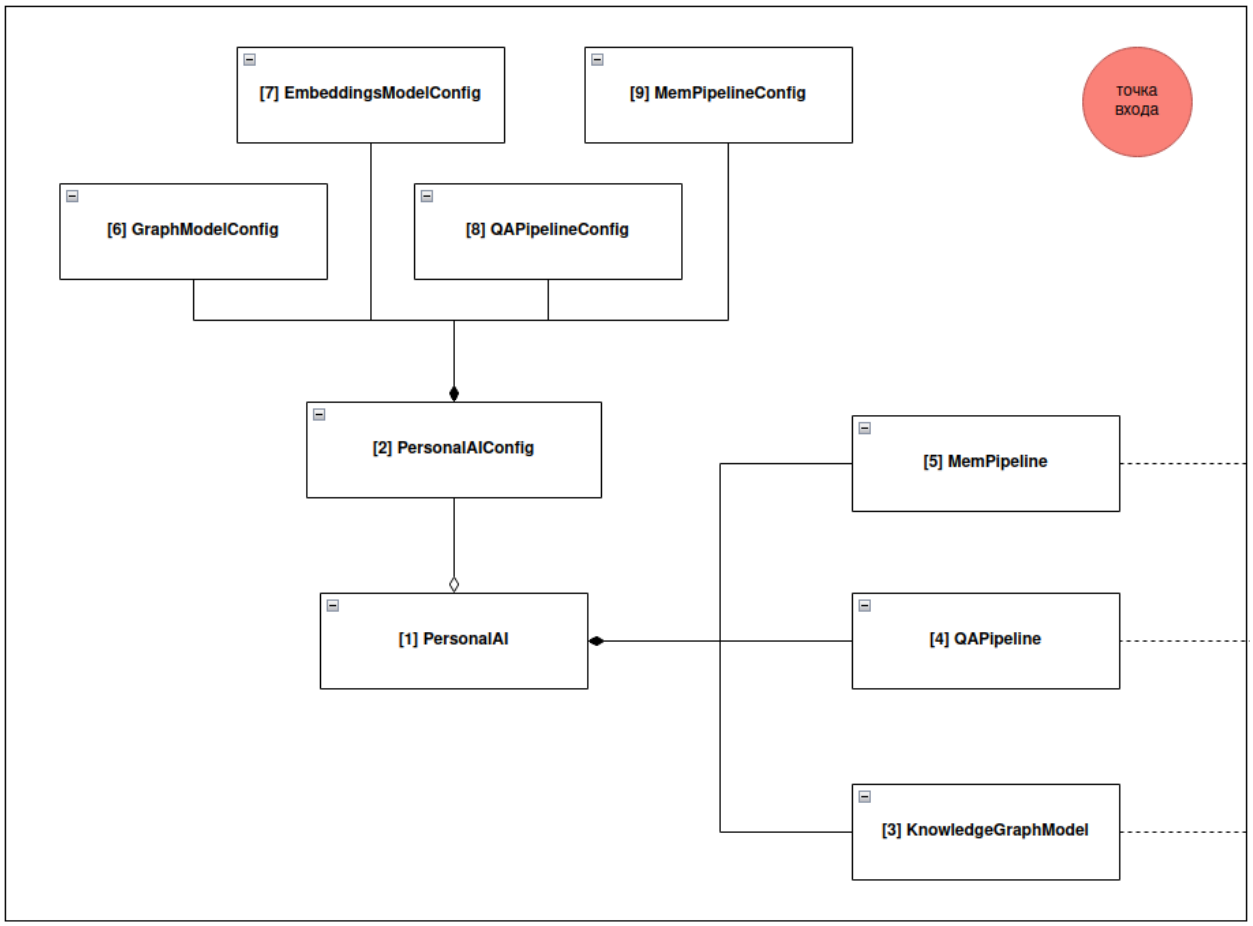
Поиск устаревшей информации в графе производится следующим образом. В начале множество сущнсотей, фигурирующих в только что извлеченных триплетах и тезисах сопоставялется с существующими вершинами графа для обнаржуения совпадений. Если таковые присутствуют, то они являются стартовым набором нод для поиска в ширину, который извлекает все связи и непресвязи, начинающиеся или заканчивающиеся в этих нодах. Далее, с помощью специального промпта ЛЛМ пытается заменить найденные знания только что извлеченными. Если какие-то триплеты или тезисы смогли быть заменены, то они удаляются из графа.

Для добавления структурированной информации реализован алгоритм, ожидающий список знаний в определенном формате, и добавляющий их в существующий граф знаний.

ГЛАВА 2

Проект подсистем

2.1 PersonalAI Main



2.1.1 src.personalai_main module

class src.personalai_main.**PersonalAIConfig**

Базовые классы: object

Конфигурация персонального ассистента.

Параметры

- **graph_struct_config** (GraphModelConfig) – Конфигурация модели, которая отвечает за представление знаний ассистента в графовом формате. Значение по умолчанию GraphModelConfig().
- **embedds_struct_config** (EmbeddingsModelConfig) – Конфигурация модели, которая отвечает за предтсавление знаний ассистента в векторном формате. Значение по умолчанию EmbeddingsModelConfig().
- **qa_pipeline_config** (QAPipelineConfig) – Конфигурация конвейера, который выполняет генерацию ответов на вопросы. Значение по умолчанию QAPipelineConfig().
- **mem_pipeline_config** (MemPipelineConfig) – Конфигурация конвейера, который выполняет изменение/обновление знаний в памяти ассистента. Значение по умолчанию MemPipelineConfig().
- **log** (Logger) – Отладочный класс для журналирования/мониторинга поведения инициализируемой комопненты. Значение по умолчанию Logger(RKG_LOG_PATH).
- **verbose** (*bool*) – Если True, то информация о поведении класса будет сохраняться в stdout и файл-журналирования (log), иначе только в файл. Значение по умолчанию False.

graph_struct_config: ***GraphModelConfig***

embedds_struct_config: ***EmbeddingsModelConfig***

qa_pipeline_config: ***QAPipelineConfig***

mem_pipeline_config: ***MemPipelineConfig***

log: ***Logger***

verbose: bool = False

class src.personalai_main.**PersonalAI**

Базовые классы: object

Верхнеуровневый класс персонального ассистента.

Параметры
config (PersonalAIConfig) – Конфигурация персонального ассистента.

answer_question(*question: str*) → Tuple[str, *ReturnInfo*]

Метод предназанчен для контекстуального поиска и извлечения релевантной информации из памяти (графа знаний) ассистента для генерации ответа на user-вопрос.

Параметры
question (*str*) – User-вопрос на естественном языке.

Результат
Кортеж из двух объектов: (1) сгенерированный ответ; (2) статус завершения операции с пояснительной информацией.

Тип результата
Tuple[str, *ReturnInfo*]

update_memory(*text: str*, *text_properties: Dict*) → Tuple[List[*Triplet*], *ReturnInfo*]

Метод предназначен для добавления новой информации в память (граф знаний) и её актуализацию.

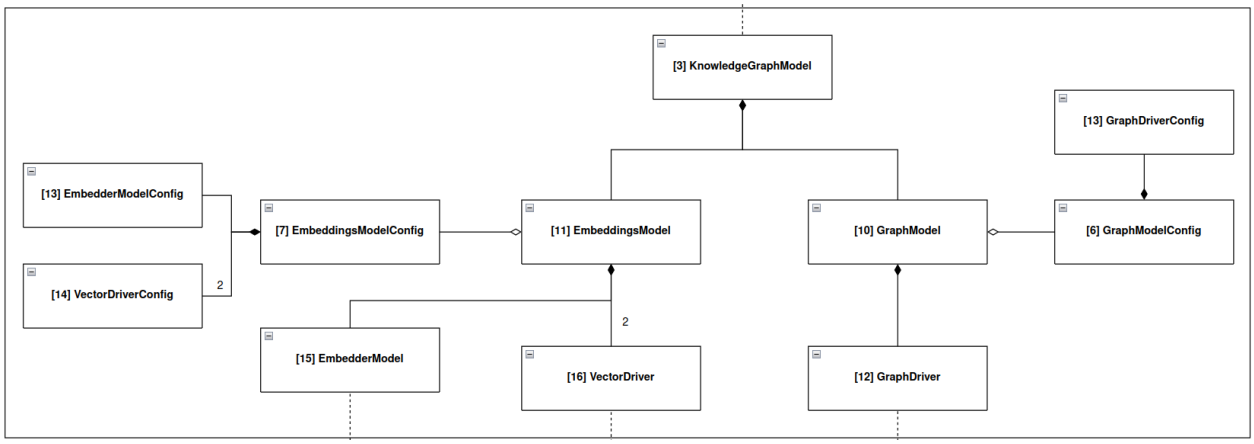
Параметры

- **text** (*str*) – Слабоструктурированный текст на естественном языке.
- **text_properties** (*Dict*) – Набор свойств данного текста, который необходимо дополнительно сохранить в память ассистента.

Результат
Кортеж из двух объектов: (1) список извлечённой из текста информации (в виде триплетов), который использовался для обновления/актуализации памяти ассистента; (2) статус завершения операции с пояснительной информацией.

Тип результата
Tuple[List[*Triplet*], *ReturnInfo*]

2.2 Knowledge Graph Model



2.2.1 src.knowledge_graph_model module

class src.knowledge_graph_model.**EmbeddingsModelConfig**

Базовые классы: object

Конфигурация векторной модели данных.

Параметры

- **nodesdb_driver_config** (*VectorDriverConfig*) – Конфигурация векторной базы данных, которая будет отвечать за хранение векторных представлений вершин из графовой модели. Значение по умолчанию NODES_DB_DEFAULT_DRIVER_CONFIG.
- **tripletsdb_driver_config** (*VectorDriverConfig*) – Конфигурация векторной базы данных, которая будет отвечать за хранение векторных представлений триплетов из графовой модели. Значение по умолчанию TRIPLETS_DB_DEFAULT_DRIVER_CONFIG.

- **tripletsdb_driver_config** – Конфигурация класса, отвечающего за приведения текста в его векторрное представление с помощью embedder-модели. Значение по умолчанию EmbedderModelConfig().
- **log** (Logger) – Отладочный класс для журналирования/мониторинга поведения инициализируемой комопненты. Значение по умолчанию Logger(EMBEDDINGS_MODEL_LOG_PATH).
- **verbose** (*bool*) – Если True, то информация о поведении класса будет сохраняться в stdout и файл-журналирования (log), иначе только в файл. Значение по умолчанию False.

nodesdb_driver_config: VectorDriverConfig

tripletsdb_driver_config: VectorDriverConfig

embedder_config: EmbedderModelConfig

log: *Logger*

verbose: bool = False

class src.knowledge_graph_model.**EmbeddingsModel**

Базовые классы: object

Модель хранения информации в векторной структуре данных.

Параметры

config (EmbeddingsModelConfig) – Конфигурация векторной модели данных. Значение по умолчанию EmbeddingsModelConfig().

create_triplets(*triplets: List[*Triplet*]*, *create_nodes: bool = True*, *batch_size: int = 128*) → None

Метод предназначен для добавления информации, представленной в виде списка триплетов, в векторную модель. Триплеты-дубликаты (по строковому представлению) в модель не добавляются.

Параметры

- **triplets** (*List[Triplet]*) – Набор триплетов для добавления в векторную модель данных.
- **create_nodes** (*bool, optional*) – Если True, то в векторную модель отдельно также будут добавлены вершины из триплетов. Вершины-дубликаты (по строковому представлению) не добавляются (отбрасываются), иначе False. Значение по умолчанию True.
- **batch_size** (*int, optional*) – Количество триплетов, которое за одну create-операцию добавляются в векторную модель. Значение по умолчанию 128.

delete_triplets(*triplets: List[*Triplet*]*, *delete_nodes: bool = True*) → None

Метод предназначен для удаления информации, представленной в виде списка триплетов, из векторной модели.

Параметры

- **triplets** (*List[Triplet]*) – Набора триплетов для удаления.
- **delete_nodes** (*bool, optional*) – Если True, то из векторной модели также будут удалены вершины, которые принадлежат данным триплетам, иначе False, Значение по умолчанию True.

create_stringified_triplets(*triplets_ids: List[str]*, *stringified_triplets: List[str]*, *nodes_ids: List[str] | None = None*, *stringified_nodes: List[str] | None = None*) → None

Метод предназначен для добавления строковых представлений триплетов/вершин в векторную модель.

Параметры

- **triplets_ids** (*List[str]*) – Идентификаторы триплетов, с которыми они будут сохранены в моделе.
- **stringified_triplets** (*List[str]*) – Строковые представления триплетов для сохранения в моделе.
- **nodes_ids** (*List[str], optional*) – Идентификаторы вершин, с которыми они будут сохранены в моделе. Значение по умолчанию None.
- **stringified_nodes** (*List[str], optional*) – Строковые представления вершин для сохранения в моделе. Значение по умолчанию None.

delete_stringified_triplets(*triplets_ids: List[str]*, *nodes_ids: List[str] | None = None*) → None

Метод предназначен для удаления строковых представлений триплетов/вершин из векторной модели.

Параметры

- **triplets_ids** (*List[str]*) – Идентификаторы триплетов на удаление из модели.
- **nodes_ids** (*List[str], optional*) – Идентификаторы вершин на удаление из модели. Значение по умолчанию None.

create_instances(*db_type: str*, *ids: List[str]*, *stringified_instances: List[str]*) → None

Метод предназначен для добавления набора объектов в одно из хранилищ данных векторной модели: для триплетов или вершин.

Параметры

- **db_type** (*str*) – Тип хранилища, в которое нужно добавить объекты. Принимает значение «triplets» или «nodes».
- **ids** (*List[str]*) – Идентификаторы объектов, с которыми они будут добавлены в хранилище.
- **stringified_instances** (*List[str]*) – Строковые представления объектов, которые будут сохранены в хранилище.

delete_instances(*db_type: str*, *ids: List[str]*) → None

Метод предназначен для удаления набора объектов из определённого хранилища данных векторной модели: из хранилища триплетов или вершин.

Параметры

- **db_type** (*str*) – Тип хранилища, из которого нужно удалить объекты. Принимает значение «triplets» или «nodes».
- **ids** (*List[str]*) – Идентификаторы объектов на удаление из хранилища.

read_embbeddings(*db_type: str*, *ids: List[str]*) → List[List[float]]

Метод предназначен для получения векторных представлений объектов из определённого хранилища векторной модели: из хранилища триплетов или вершин.

Параметры

- **db_type** (*str*) – Тип хранилища, в котором осуществялется поиск векторных представлений для заданных объектов. Принимает значени «triplets» или «nodes».

- **ids** (*List[str]*) – Идентификаторы объектов, для которых необходимо получить векторные представления.

Результат
Список полученных векторных представлений для заданных объектов.

Тип результата
List[List[float]]

class src.knowledge_graph_model.**GraphModelConfig**

Базовые классы: object

Конфигруация графовой модели.

Параметры

- **driver_config** (*GraphDriverConfig*) – Конфигурация графового хранилища данных.
- **log** (Logger) – Отладочный класс для журналирования/мониторинга поведения инициализируемой комопненты. Значение по умолчанию Logger(GRAPH_MODEL_LOG_PATH).
- **verbose** (*bool*) – Если, True, то информация о поведении класса будет сохраняться в stdout и файл-журналирования (log), иначе только в файл. Значение по умолчанию False.

driver_config: GraphDriverConfig

log: *Logger*

verbose: bool = False

class src.knowledge_graph_model.**GraphModel**

Базовые классы: object

Модель хранения информации в графовой структуре данных.

Параметры
config (GraphModelConfig) – Конфигурация графовой модели. Значение по умолчанию GraphModelConfig().

create_triplets(*triplets: List[Triplet]*, *batch_size: int = 64*) → None

Метод предназначен для сохранения информации, представленной в виде списка триплетов, в графовую модель.

Параметры

- **triplets** (*List[Triplet]*) – Набора триплетов для добавления в графовую модель.
- **batch_size** (*int, optional*) – Количество триплетов, которое за одну create-операцию добавляется в модель. Значение по умолчанию 64.

delete_triplets(*triplets: List[Triplet]*, *batch_size: int = 64*) → None

Метод предназначен для удаления информации, представленной в виде списка триплетов, из графовой модели.

Параметры

- **triplets** (*List[Triplet]*) – Набор триплетов для удаления из графовой модели.

- **batch_size** (*int, optional*) – Количество триплетов, которое за одну delete-операцию удаляется из графовой модели. Значение по умолчанию 64.

class src.knowledge_graph_model.**KnowledgeGraphModel**

Базовые классы: object

Модель памяти (графа знаний) ассистента.

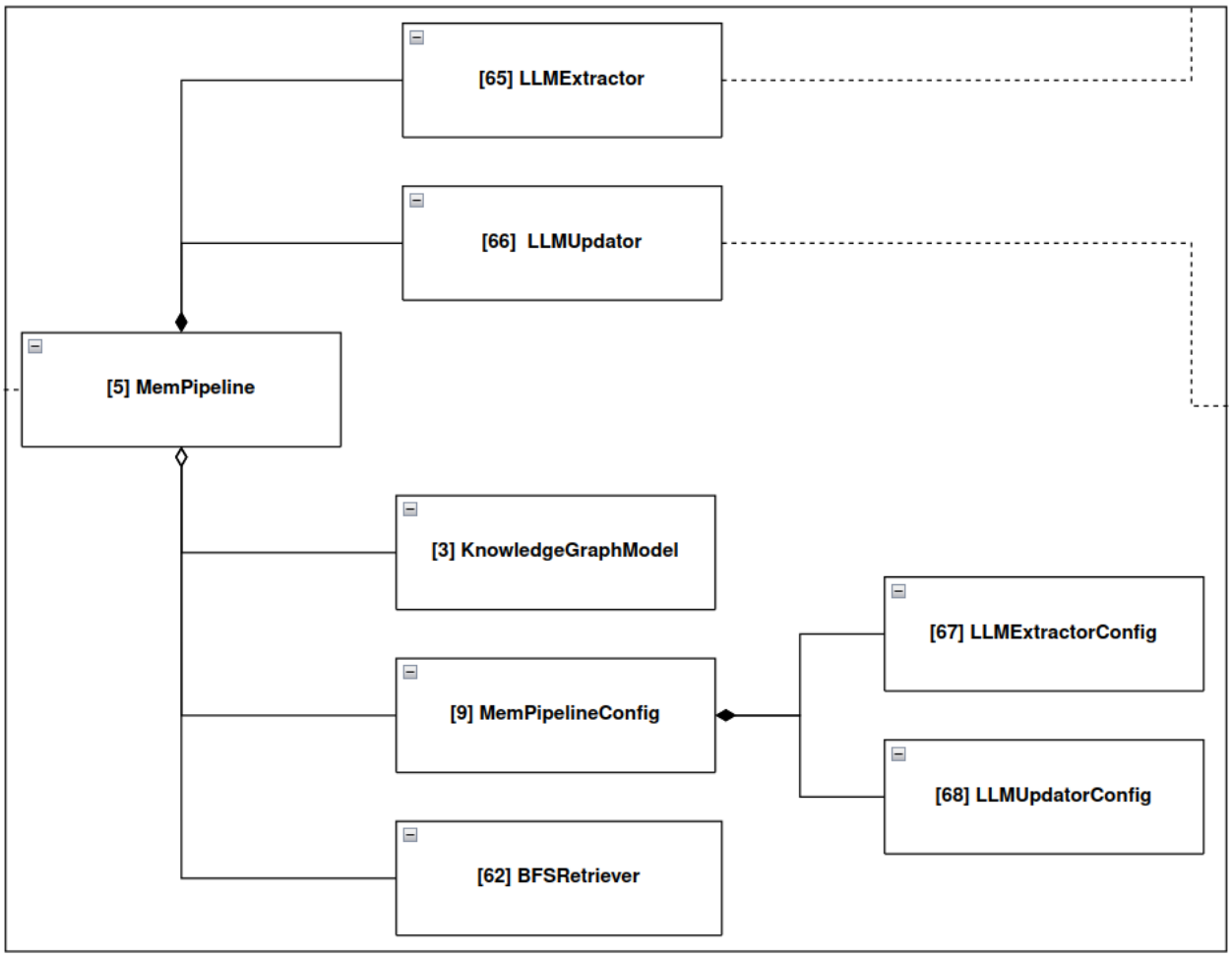
Параметры

- **graph_struct** (EmbeddingsModel) – Знания, хранящиеся в графовой структуре данных.
- **graph_struct** – Знания, хранящиеся в векторной структуре данных.

graph_struct: *GraphModel*

embeddings_struct: *EmbeddingsModel*

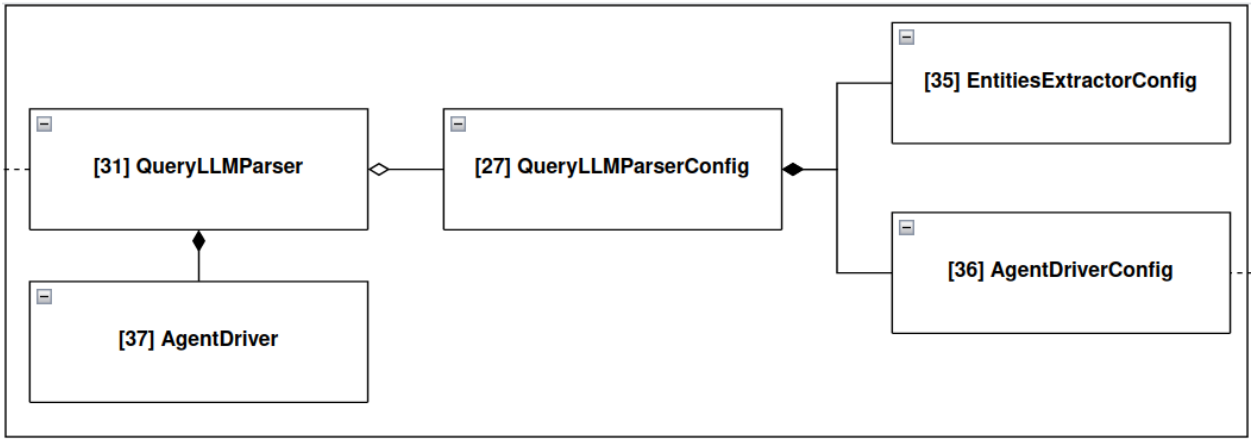
2.3 QA-pipeline



2.3.1 src.qa_pipeline package

Subpackages

Query Parser



src.qa_pipeline.query_parser package

Submodules

src.qa_pipeline.query_parser.QueryLLMParser module

class src.qa_pipeline.query_parser.QueryLLMParser.**QueryLLMParserConfig**

Базовые классы: `object`

Конфигурация «Query Parser»-стадии.

Параметры

- **lang** (*str*) – Язык, который будет использоваться в подаваемом на вход тексте. На основании выбранного языка будут использоваться соответствующие промпты. Если „auto“, то язык определяется автоматически. Значение по умолчанию „auto“.
- **ents_extr_config** (EntitiesExtractorConfig) – Конфигурация алгоритма по извлечению ключевых сущностей из текста. Значение по умолчанию EntitiesExtractorConfig().
- **agent_cofig** (*AgentDriverConfig*) – Конфигурация LLM-агента, который будет использоваться в рамках данной стадии. Значение по умолчанию AgentDriverConfig().
- **log** (Logger) – Отладочный класс для журналирования/мониторинга поведения инициализируемой компоненты. Значение по умолчанию Logger(QP_LOG_PATH).
- **verbose** (*bool*) – Если True, то информация о поведении класса будет сохраняться в stdout и файл-журналирования (log), иначе только в файл. Значение по умолчанию False.

lang: str = 'auto'

ents_extr_config: *EntitiesExtractorConfig*

agent_cofig: AgentDriverConfig

log: *Logger*

verbose: bool = False

class src.qa_pipeline.query_parser.QueryLLMParser.**QueryLLMParser**

Базовые классы: object

Верхнеуровневый класс первой стадии QA-конвейера для извлечения сущностей из user-вопроса.

Параметры

config (QueryLLMParserConfig) – Конфигурация «Query Parser»-стадии. Значение по умолчанию QueryLLMParserConfig().

extract_entities(*query: str*) → Tuple[*QueryInfo*, *ReturnInfo*]

Метод предназначен для извлечения ключевых сущностей из query-текста.

Параметры

query (*str*) – Текст на естественном языке.

Результат

Кортеж из двух объектов: (1) структура данных со списком извлечённых ключевых сущностей из query; (2) статус завершения операции с пояснительной информацией.

Тип результата

Tuple[*QueryInfo*, *ReturnInfo*]

src.qa_pipeline.query_parser.utils module

class src.qa_pipeline.query_parser.utils.**EntitiesExtractorConfig**(*user_prompt: ~typing.Dict = <factory>*, *system_prompt: ~typing.Dict = <factory>*, *entities_parse_func: ~typing.Dict = <factory>*)

Базовые классы: object

Конфигурация алгоритма по извлечению ключевых сущностей из текста.

Параметры

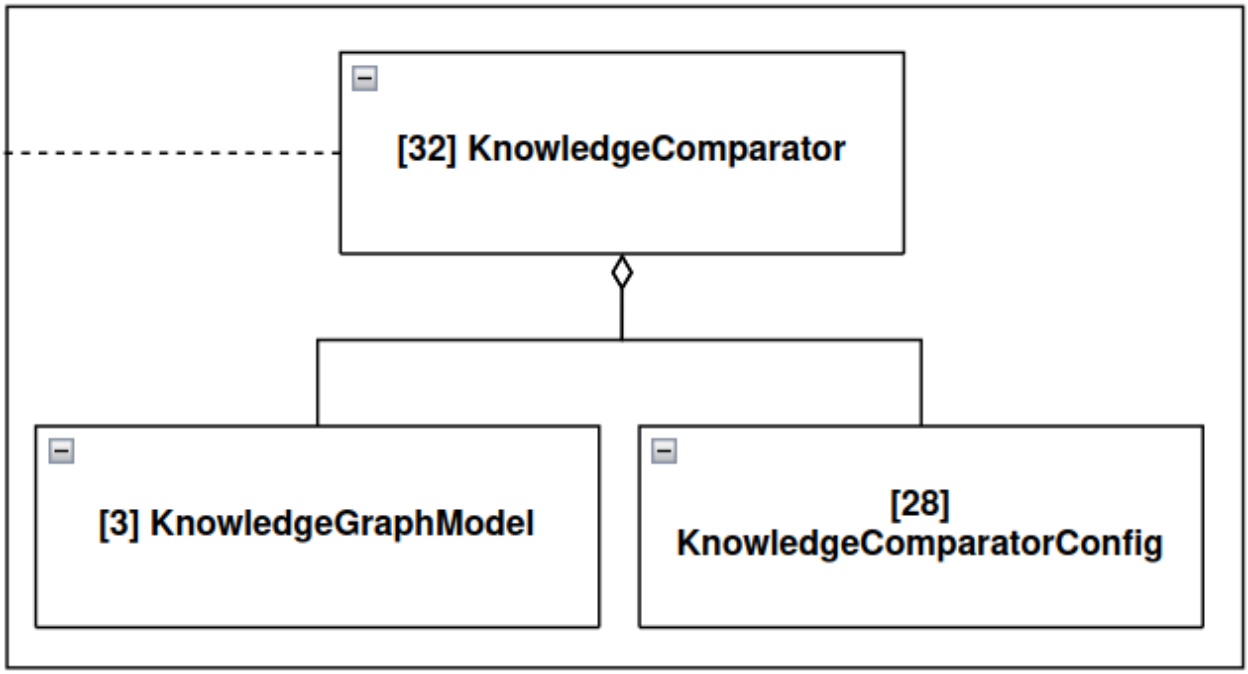
- **user_prompt** (*Dict*) – User-промпт для LLM-агента с описанием задачи по извлечению ключевых сущностей из текста.
- **system_prompt** – System-промпт с описание персоны, свойствам которой должен удовлетворять LLM-агент при генерации оветов.
- **entities_parse_func** (*Dict*) – Функция разбора результатов генерации LLM-агента.

user_prompt: Dict

system_prompt: Dict

entities_parse_func: Dict

Knowledge Comparator



src.qa_pipeline.knowledge_comparator package

Submodules

src.qa_pipeline.knowledge_comparator.KnowledgeComparator module

class src.qa_pipeline.knowledge_comparator.KnowledgeComparator.
KnowledgeComparatorConfig

Базовые классы: `object`

Конфигурация «Knowledge Comparator»-стадии.

Параметры

- **threshold** (*float*) – Нижний порог близости между эмбеддингами сущностей и вершин для их сопоставления. Значение по умолчанию 0.5.
- **fetch_n** (*int*) – лужебный гиперпараметр. Defaults to 20.
- **max_k** (*int*) – Максимальное количество вершин из графа знаний, которое может быть сопоставлено одной сущности. Значение по умолчанию 1.
- **k_compare** (*int*) – Значение по умолчанию 5.
- **log** (Logger) – Отладочный класс для журналирования/мониторинга поведения инициализируемой компопненты. Значение по умолчанию Logger(COMPARATOR_LOG_PATH).
- **verbose** (*bool*) – Если True, то информация о поведении класса будет сохраняться в stdout и файл-журналирования (log), иначе только в файл. Значение по умолчанию False.

threshold: float = 0.5

fetch_n: int = 20

max_k: int = 1

k_compare: int = 5

log: *Logger*

verbose: bool = False

class src.qa_pipeline.knowledge_comparator.KnowledgeComparator.**KnowledgeComparator**

Базовые классы: object

Верхнеуровневый класс второй стадии QA-конвейера для сопостовения информации из user-вопроса с имеющейся информацией в памяти (графе знаний) ассистента.

Параметры

- **kg_model** (KnowledgeGraphModel) – Модель памяти (графа знаний) ассистента.
- **config** (KnowledgeComparatorConfig) – Конфигурация «Knowledge Comparator»-стадии. Значение по умолчанию KnowledgeComparatorConfig().

link_kgnodes_to_query(*query_structure:* QueryInfo) → *ReturnInfo*

Метод предназначен для сопоставления (матчинга) сущностей, извлечённых из user-вопроса, с вершинами из графа знаний ассистента.

Параметры

query_structure (QueryInfo) – Структура данных, которая хранит user-вопрос и извлечённые из него сущности.

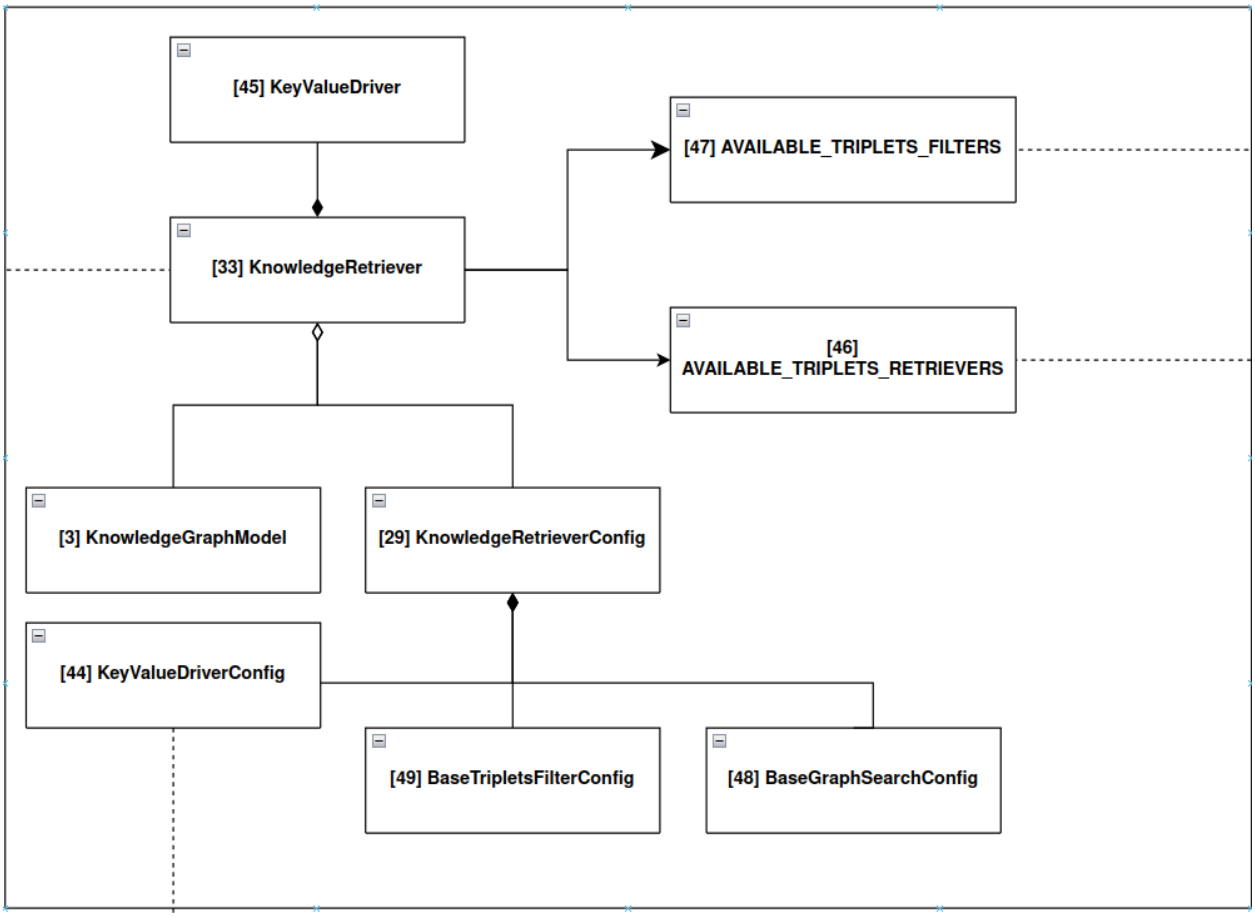
Результат

Статс завершения операции с пояснительной информацией.

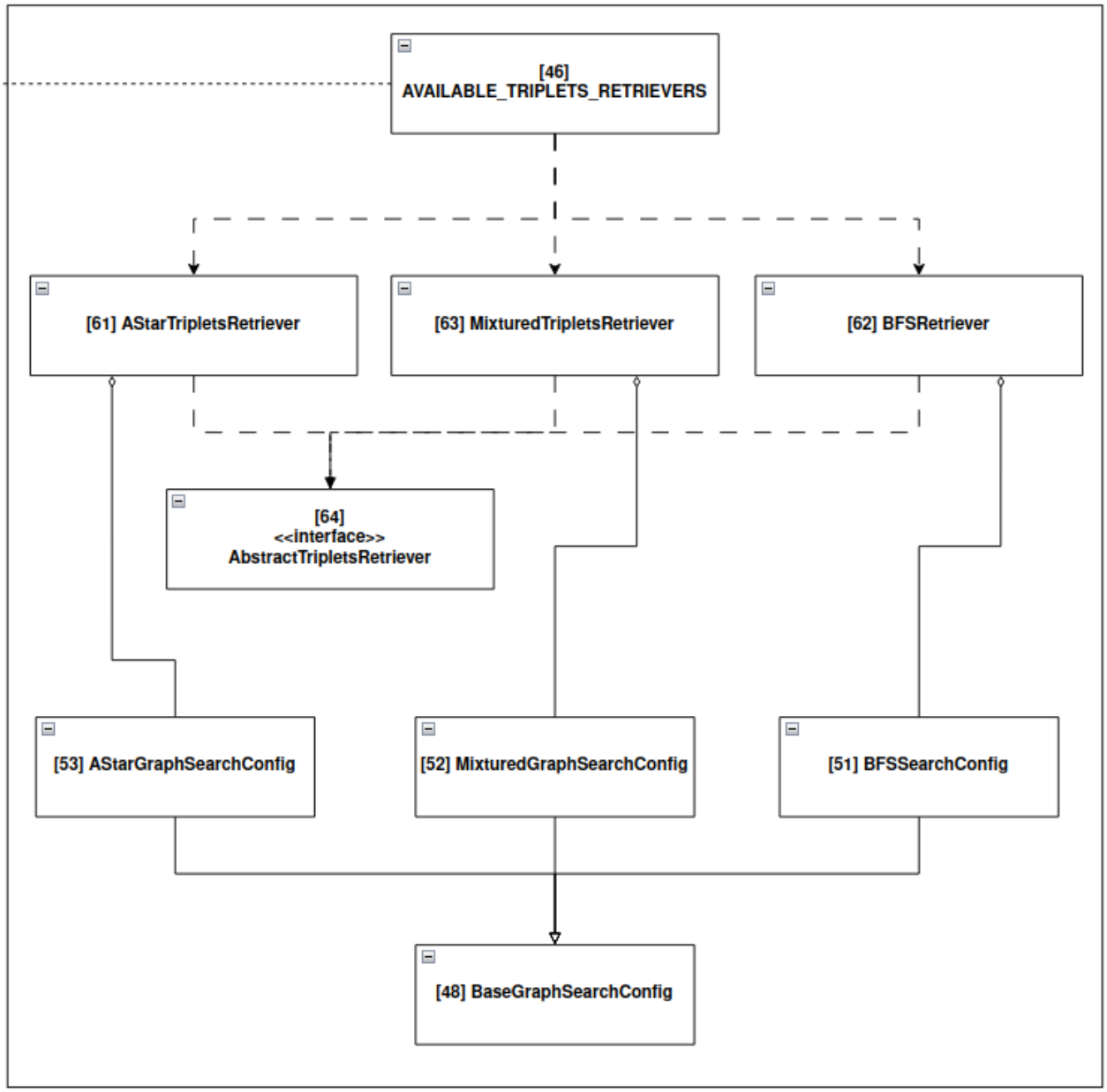
Тип результата

ReturnInfo

Knowledge Retriever



Available Triplets Retrievers



src.qa_pipeline.knowledge_retriever package

Submodules

src.qa_pipeline.knowledge_retriever.AStarTripletsRetriever module

class src.qa_pipeline.knowledge_retriever.AStarTripletsRetriever.**AStarMetricsConfig**

Базовые классы: `object`

Конфигурация класса для расчёта метрик, используемых в рамках A*-алгоритма.

Параметры

- **h_metric_name** (*str*) – Эвристическая метрика, которая будет использоваться для оценки расстояния между текущей и конечной вершинами. Данное поле принимает следующие значения: „ip“, „weight_with_short_path“, „avg_weighted_with_short_path“. Значение по умолчанию „ip“.
- **kvdriver_config** (*KeyValueDriverConfig*) – Конфигурация кеша для хранения рассчитанных h-оценок между вершинами. Значение по умолчанию None

h_metric_name: str = 'ip'

kvdriver_config: KeyValueDriverConfig = None

class src.qa_pipeline.knowledge_retriever.AStarTripletsRetriever.**AStarMetrics**

Базовые классы: object

Класс предназначен для расчёта d- и h-метрик, испрльзуемых в рамках A*-алгоритма поиска.

Параметры

- **kg_model** (KnowledgeGraphModel) – Модель памяти (графа знаний) ассистента.
- **config** (AStarMetricsConfig) – Конфигурация класса. Значение по умолчанию AStarMetricsConfig().
- **accepted_node_types** (*List[NodeType]*) – Типы вершин, которые можно использовать при расчёте метрик.
- **log** (Logger) – Отладочный класс для журналирования/мониторинга поведения инициализируемой компоненты. Значение по умолчанию Logger(RETRIEVER_LOG_PATH).
- **verbose** (*bool*) – Если True, то информация о поведении класса будет сохраняться в stdout и файл-журналирования (log), иначе только в файл. Значение по умолчанию False.

compute_h_metric(**args*, ***kwargs*) → float

get_nodes_path(*parent: Dict[str, str]*, *end_node_id: str*) → List[str]

precomputed_dist(*node1_id: str*, *node2_id: str*, **args*, ***kwargs*) → float

bfs(*s_node_id*, *e_node_id*)

precomputed_short_path(*node1_id: str*, *node2_id: str*) → float

weighted_short_path(*node1_id: str*, *node2_id: str*, **args*, ***kwargs*) → float

avg_weighted_short_path(*node1_id: str*, *node2_id: str*, *parent: Dict[str, str]*) → float

class src.qa_pipeline.knowledge_retriever.AStarTripletsRetriever.**AStarGraphSearchConfig**

Базовые классы: BaseGraphSearchConfig

Конфигурация класса, реализующего логику A*-алгоритма поиска по графу знаний.

Параметры

- **metrics_config** (AStarMetricsConfig) – Конфигурация класса, выполняющая расчёт необходимых матрик для A*-алгоритма. Значение по по умолчанию AStarMetricsConfig().

- **max_depth** (*int*) – Макимальная глубина обхода графа для поиска заданной вершины. Если указано значение -1, то данное ограничение выключается. Значение по умолчанию 10.
- **max_passed_nodes** (*int*) – Максимальное количество вершин, которое можно обойти для поиска заднной вершины в графе. Если указано значение -1, то данное ограничение выключается. Значение по умолчанию 500.
- **accepted_node_types** (*List[NodeType]*) – Типы вершин, которые можно обходить во время поиска заданной вершины. Значение по умолчанию [NodeType.object , NodeType.hyper, NodeType.episodic].

metrics_config: *AStarMetricsConfig*

max_depth: int = 10

max_passed_nodes: int = 500

accepted_node_types: List[*NodeType*]

class src.qa_pipeline.knowledge_retriever.AStarTripletsRetriever.**AStarGraphSearch**

Базовые классы: object

Класс предназначен для запуска A*-алгоритма для излвечения триплетов из графового хранилища триплетов.

Параметры

- **kg_model** (KnowledgeGraphModel) – Модель памяти (графа знаний) ассистента.
- **search_config** (AStarGraphSearchConfig) – Конфигурация A*-алгоритма поиска по графовому хранилищу триплетов. Значение по умолчанию AStarGraphSearchConfig().
- **log** (Logger) – Отладочный класс для журналирования/мониторинга поведения инициализируемой комопненты. Значение по умолчанию Logger(RETRIEVER_LOG_PATH).
- **verbose** (*bool*) – Если True, то информация о поведении класса будет сохраняться в stdout и файл-журналирования (log), иначе только в файл. Значение по умолчанию False.

search_path(*start_node_id: str*, *end_node_id: str*) → Tuple[List[str], List[str], Dict[str, int], Dict[str, str], str]

Реализация A*-алгоритма. Источник: https://www.redblobgames.com/pathfinding/a-star/implementation.html.

class src.qa_pipeline.knowledge_retriever.AStarTripletsRetriever.**AStarTripletsRetriever**

Базовые классы: AbstractTripletsRetriever

Класс предназначен для извлечения триплетов из графа знаний на основе A*-алгоритма поиска.

Параметры

- **kg_model** (KnowledgeGraphModel) – Модель памяти (графа знаний) ассистента.
- **search_config** (AStarGraphSearchConfig) – Конфигурация A*-алгоритма поиска по графовому хранилищу триплетов. Значение по умолчанию AStarGraphSearchConfig().

- **log** (Logger) – Отладочный класс для журналирования/мониторинга поведения инициализируемой компоненты. Значение по умолчанию Logger(RETRIEVER_LOG_PATH).
- **verbose** (*bool*) – Если True, то информация о поведении класса будет сохраняться в stdout и файл-журналирования (log), иначе только в файл. Значение по умолчанию False.

get_nodes_path(*parent: Dict[str, str]*, *end_node_id: str*, *spare_closest_node_id: str*) → List[str]

get_relevant_triplets(*query_info:* QueryInfo) → List[*Triplet*]

Метод предназначен для извлечения триплетов из графа знаний на основании информации из user-вопроса. На выходе список триплетов не содержит дубликатов (по строковому представлению).

Параметры
query_info (QueryInfo) – Структура данных с информацией о user-вопросе.

Результат
Набор триплетов, извлечённый из графа знаний.

Тип результата
List[*Triplet*]

src.qa_pipeline.knowledge_retriever.BFSTripletsRetriever module

class src.qa_pipeline.knowledge_retriever.BFSTripletsRetriever.**BFSSearchConfig**

Базовые классы: BaseGraphSearchConfig

summary

strict_filter: bool = True

hyper_episodic_num: int = 15

chain_triplets_num: int = 25

other_triplets_num: int = 6

class src.qa_pipeline.knowledge_retriever.BFSTripletsRetriever.**BFSRetriever**

Базовые классы: AbstractTripletsRetriever

Класс с реализацией алгоритма BFS (поиск в ширину) по графу

Параметры

- **kg_model** (KnowledgeGraphModel) – класс для извлечения триплетов из графа
- **log** (Logger) – класс для логирования
- **search_config** (BFSSearchConfig, *optional*) – конфигурация поиска по графу

parse_triplet_output(*direction: str*, *query: List[list | str]*, *another_entities1: List[str]*, *another_entities2: Dict[str, List[List[str]]]*, *chain: List[List[str]]*) → Tuple[Tuple[List[dict], List[Tuple[str]], List[List[dict]]], List[List[List[str]]], List[List[List[str]]]]

summary

Параметры

- **direction** (*str*) – направление поиска
- **query** (*List[Union[list, str]]*) – запрос для поиска

- **another_entities1** (*List[str]*) – сущности из вопроса
- **another_entities2** (*Dict[str, List[List[str]]]*) – сущности, найденные в процессе поиска
- **chain** (*List[List[str]]*) – путь в графе

Результат
информация об извлеченных триплетах и пути в графе

Тип результата
Tuple[List[List[dict], List[Tuple[str]], List[List[dict]]], List[List[List[str]]], List[List[List[str]]]]

add_chains (*inters_chains1: List[List[List[str]]]*, *inters_chains2: List[List[List[str]]]*, *cur_inters_chains1: List[List[List[str]]]*, *cur_inters_chains2: List[List[List[str]]]*) → Tuple[List[List[List[str]]], List[List[List[str]]]]

summary

Параметры

- **inters_chains1** (*List[List[List[str]]]*) – пути в графе, которые пересекаются с сущностями из вопроса
- **inters_chains2** (*List[List[List[str]]]*) – пути в графе, которые пересекаются с другими путями
- **cur_inters_chains1** (*List[List[List[str]]]*) – пути в графе, которые пересекаются с сущностями из вопроса, на текущем шаге поиска
- **cur_inters_chains2** (*List[List[List[str]]]*) – пути в графе, которые пересекаются с другими путями, на текущем шаге поиска

Результат
пути в графе, которые пересекаются с сущностями из вопроса

Тип результата
Tuple[List[List[List[str]]], List[List[List[str]]]]

make_triplet_key (*triplet: List[Dict[str, str]]*) → Tuple[Tuple[str], Tuple[str]]

summary

Параметры
triplet (*List[Dict[str, str]]*) – триплет

Результат
субъект, отношение и объект в триплете

Тип результата
Tuple[Tuple[str], Tuple[str]]

get_relevant_triplets (*query_info:* QueryInfo, *depth: int = 1*) → List[*Triplet*]

Метод предназначен для извлечения триплетов из графа знаний на основании информации из user-вопроса. На выходе список триплетов не содержит дубликатов (по строковому представлению).

Параметры
query_info (QueryInfo) – Структура данных с информацией о user-вопросе.

Результат
Набор триплетов, извлечённый из графа знаний.

Тип результата
List[*Triplet*]

extract_thesis_for_entities (*seed_entities: List[List[Tuple[str]]]*, *entities_list: List[Tuple[str]]*, *entity_type: str*, *texts_set: Set[str]*) → Tuple[List[Tuple[str, str, Dict[str, str], Dict[str, str], int, str]], Set[str]]

summary

Параметры

- **seed_entities** (*List[List[Tuple[str]]]*) – список сущностей из графа для сущностей из вопроса
- **entities_list** (*List[Tuple[str]]*) – список сущностей из графа для данной сущности из вопроса
- **entity_type** (*str*) – тип сущности («hyper» или «episodic»)
- **texts_set** (*Set[str]*) – набор текстов из триплетов

Результат
извлеченные тексты из триплетов типа hyper и episodic

Тип результата
Tuple[List[Tuple[str, str, Dict[str], Dict[str], int, str]], Set[str]]

extract_thesis (*seed_entities: List[List[Tuple[str]]]*, *same_types: bool*) → List[str]

summary

Параметры

- **seed_entities** (*List[List[Tuple[str]]]*) – список сущностей из графа для сущностей из вопроса
- **same_types** (*bool*) – принадлежат ли сущности из вопроса к одному и тому же типу

Результат
список текстов тезисов

Тип результата
List[str]

bfs (*seed_entities: List[List[Tuple[str]]]*, *depth: int = 1*, *subj_labels: List[str] | None = None*, *obj_labels: List[str] | None = None*, *use_rel_props: bool = False*) → Tuple[Dict[tuple, Tuple[List[dict], List[Tuple[str]], List[List[dict]]]], List[List[List[str]]], List[List[List[str]]]]

summary

Параметры

- **seed_entities** (*List[List[Tuple[str]]]*) – список сущностей из графа для сущностей из вопроса
- **depth** (*int, optional*) – глубина поиска
- **subj_labels** (*List[str], optional*) – список типов субъекта в триплетах, включаемых в пути в графе во время поиска
- **obj_labels** (*List[str], optional*) – список типов субъекта в триплетах, включаемых в пути в графе во время поиска
- **use_rel_props** (*bool, optional*) – использовать ли при поиске свойства relations

Результат
извлеченные триплеты и пути в графе

Тип результата
Tuple[Dict[tuple, List[List[dict], List[Tuple[str]], List[List[dict]]]], List[List[List[str]]], List[List[List[str]]]]

src.qa_pipeline.knowledge_retriever.BFSTripletsRetriever.**process_chain**(*chain: List[List[str]]*, *chain_subj_obj: List[Tuple[str]]*, *chain_triplets: List[List[str]]*) → Tuple[List[Tuple[str]], List[List[str]]]

summary

Параметры

- **chain** (*List[List[str]]*) – пути в графе, которые начинаются от сущностей из вопроса
- **chain_subj_obj** (*List[Tuple[str]]*) – список субъектов и объектов триплетов из путей в графе
- **chain_triplets** (*List[List[str]]*) – список триплетов из путей в графе

Результат
список субъектов и объектов триплетов из путей в графе, список триплетов из путей в графе

Тип результата
Tuple[List[Tuple[str]], List[List[str]]]

src.qa_pipeline.knowledge_retriever.BFSTripletsRetriever.**process_inters_chains1**(*inters_chains1: List[List[List[str*) → List[List[str]]

summary

Параметры
inters_chains1 (*List[List[List[str]]]*) – пути в графе, которые начинаются от сущностей из вопроса и пересекаются

Результат
список триплетов из путей в графе

Тип результата
List[List[str]]

src.qa_pipeline.knowledge_retriever.BFSTripletsRetriever.**process_inters_chains2**(*inters_chains2: List[List[List[str*) → List[List[str]]

summary

Параметры
inters_chains2 – пути в графе, которые начинаются от сущностей из вопроса и пересекаются

Результат
список триплетов из путей в графе

Тип результата
List[List[str]]

src.qa_pipeline.knowledge_retriever.MixturedTripletsRetriever module

class src.qa_pipeline.knowledge_retriever.MixturedTripletsRetriever.**MixturedGraphSearchConfig**

Базовые классы: BaseGraphSearchConfig

Конфигурация комбинированного алгоритма по исзлечению триплетов из графа знаний.

Параметры

- **astar_config** (AStarGraphSearchConfig) – Конфигурация A*-алгоритиа поиска. Значение по умолчанию AStarGraphSearchConfig().
- **bfs_config** (BFSSearchConfig) – Конфигурация BFS-алгоритма поиска. Значение по умолчанию BFSSearchConfig().

astar_config: ***AStarGraphSearchConfig***

bfs_config: ***BFSSearchConfig***

class src.qa_pipeline.knowledge_retriever.MixturedTripletsRetriever.**MixturedTripletsRetriever**

Базовые классы: AbstractTripletsRetriever

Класс предназначен для извлечения триплетов из графа знаний с помощью комбинации BFS- и A*-алгоритмов поиска.

Параметры

- **kg_model** (KnowledgeGraphModel) – Модель памяти (графа знаний) ассистента.
- **config** (MixturedGraphSearchConfig) – Конфигурация комбинирванного алгоритма поиска. Значение по умолчанию MixturedGraphSearchConfig().
- **log** (Logger) – Отладочный класс для журналирования/мониторинга поведения инициализируемой комопненты. Значение по умолчанию Logger(LOG_PATH).
- **verbose** (*bool*) – Если True, то информация о поведении класса будет сохраняться в stdout и файл-журналирования (log), иначе только в файл. Значение по умолчанию False.

get_relevant_triplets(*query_info:* QueryInfo) → List[*Triplet*]

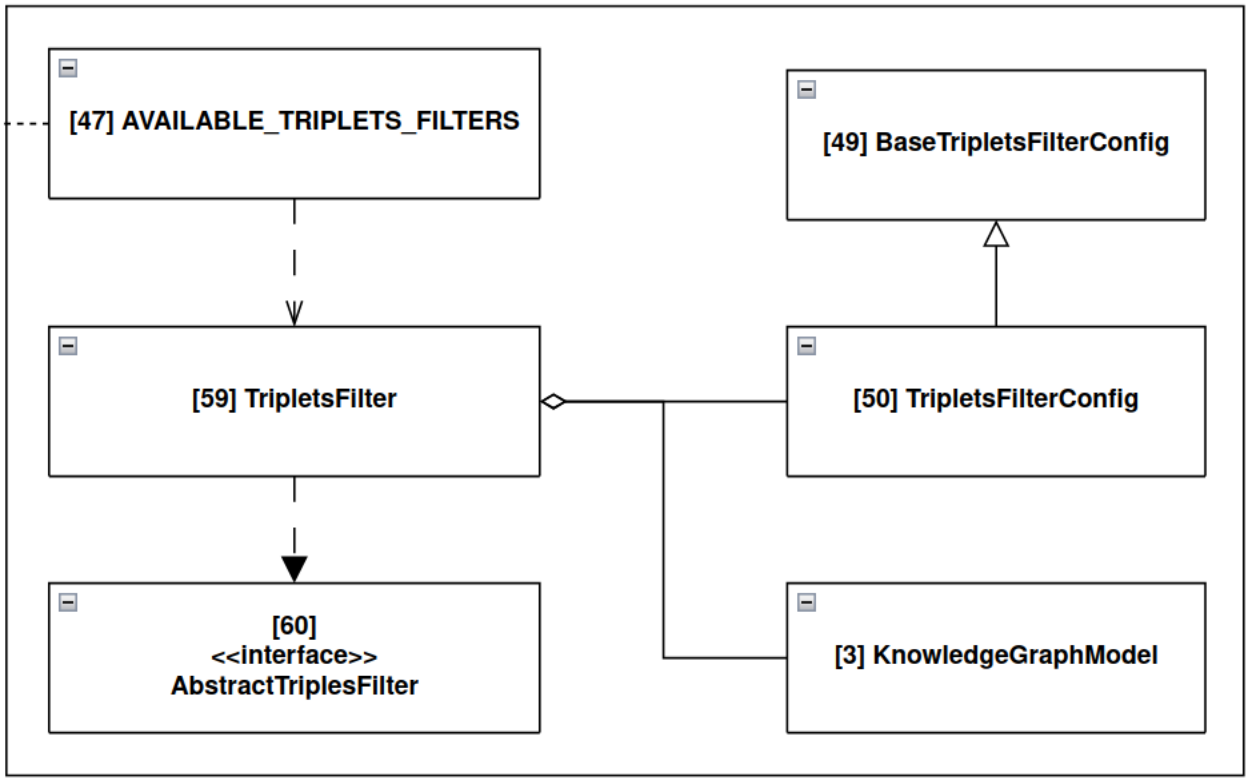
Метод предназначен для извлечения триплетов из графа знаний на основании информации из user-вопроса. На выходе список триплетов не содержит дубликатов (по строковому представлению).

Параметры
query_info (QueryInfo) – Структура данных с информацией о user-вопросе.

Результат
Набор триплетов, извлечённый из графа знаний.

Тип результата
List[*Triplet*]

Available Triplets Filters



src.qa_pipeline.knowledge_retriever package

src.qa_pipeline.knowledge_retriever.TripletsFilter module

class src.qa_pipeline.knowledge_retriever.TripletsFilter.**TripletsFilterConfig**

Базовые классы: BaseTripletsFilterConfig

Конфигурация наивного алгоритма раннжирования/фильтрации триплетов.

Параметры

max_k (*int*) – Первые k (по релевантности) триплетов, которые будут возвращены в результате операции ранжирования. Значение по умолчанию 50.

max_k: int = 50

class src.qa_pipeline.knowledge_retriever.TripletsFilter.**TripletsFilter**

Базовые классы: AbstractTriplesFilter

Класс реализует логику наивного ранжирования/фильтрации триплетов на основе их релевантности к user-вопросу.

Параметры

- **kg_model** (KnowledgeGraphModel) – Модель памяти (графа знаний) ассистента.
- **config** (TripletsFilterConfig) – Конфигурация наивного алгоритма фильтрации. Значение по умолчанию TripletsFilterConfig().
- **log** (Logger) – Отладочный класс для журналирования/мониторинга поведения инициализируемой компоненты. Значение по умолчанию Logger(LOG_PATH).

- **verbose** (*bool*) – Если True, то информация о поведении класса будет сохраняться в stdout и файл-журналирования (log), иначе только в файл. Значение по умолчанию False.

apply_filter (*query_info:* QueryInfo, *triplets: List[*Triplet*]*) → List[*Triplet*]

Метод предназначен для применения операциии ранжирования/фильтрации к набору триплетов на основе меры релевантности к user-вопросу.

Параметры

- **query_info** (QueryInfo) – Структура данных с user-вопросом.
- **triplets** (*List[Triplet]*) – Набор триплетов для ранжирования/отбора.

Результат
Набор триплетов, релевантных данному user-вопрос.

Тип результата
List[*Triplet*]

src.qa_pipeline.knowledge_retriever package

src.qa_pipeline.knowledge_retriever.KnowledgeRetriever module

class src.qa_pipeline.knowledge_retriever.KnowledgeRetriever.**KnowledgeRetrieverConfig**

Базовые классы: object

Конфигурация «Knowledge Retriever»-стадии.

Параметры

- **retriever_method** (*str*) – TODO. Значение по умолчанию „astar“.
- **retriever_config** (*BaseGraphSearchConfig*) – TODO. Значение по умолчанию AStarGraphSearchConfig().
- **filter_method** (*str*) – TODO. Значение по умолчанию „naive“.
- **filter_config** (*BaseTripletsFilterConfig*) – TODO. Значение по умолчанию TripletsFilterConfig().
- **log** (Logger) – Отладочный класс для журналирования/мониторинга поведения инициализируемой компонненты. Значение по умолчанию Logger(RETRIEVER_LOG_PATH).
- **verbose** (*bool*) – Если True, то информация о поведении класса будет сохраняться в stdout и файл-журналирования (log), иначе только в файл. Значение по умолчанию False.

retriever_method: str = 'astar'

retriever_config: BaseGraphSearchConfig

filter_method: str = 'naive'

filter_config: BaseTripletsFilterConfig

log: *Logger*

verbose: bool = False

class src.qa_pipeline.knowledge_retriever.KnowledgeRetriever.**KnowledgeRetriever**

Базовые классы: object

Верхнеуровневый класс третьей стадии QA-конвейера для извлечения релевантной к user-вопросу информации из памяти (графа знаний) ассистента.

Параметры

- **kg_model** (KnowledgeGraphModel) – Модель памяти (графа знаний) ассистента. Значение по умолчанию „astar“.
- **config** (KnowledgeRetrieverConfig) – Конфигурация „Knowledge Retriever“-стадии. Значение по умолчанию KnowledgeRetrieverConfig().

retrieve(*query_info:* QueryInfo) → Tuple[List[*Triplet*], *ReturnInfo*]

Метод предназначен для извлечения релевантных к user-вопросу триплетов из графа знаний.

Параметры

query_info (QueryInfo) – Структура данных, которая хранит user-вопрос и связанную с ним инфомрацию.

Результат

Кортеж из двух объектов: (1) список релевантных user-вопросу триплетов; (2) статус завершения операции с пояснительной информацией.

Тип результата

Tuple[List[*Triplet*], *ReturnInfo*]

src.qa_pipeline.knowledge_retriever.utils module

class src.qa_pipeline.knowledge_retriever.utils.**AbstractTriplesFilter**

Базовые классы: ABC

Интерфейс алгоритмов фильтрации/ранжирования триплетов.

abstract apply_filter(*query_info:* QueryInfo, *triplets: List[*Triplet*]*) → List[*Triplet*]

Метод предназначен для применения операциии ранжирования/фильтрации к набору триплетов на основе меры релевантности к user-вопросу.

Параметры

- **query_info** (QueryInfo) – Структура данных с user-вопросом.
- **triplets** (*List[*Triplet*]*) – Набор триплетов для ранжирования/отбора.

Результат

Набор триплетов, релевантных данному user-вопрос.

Тип результата

List[*Triplet*]

class src.qa_pipeline.knowledge_retriever.utils.**AbstractTripletsRetriever**

Базовые классы: ABC

Интерфейс алгоритмов извлечения триплетов из графа знаний.

abstract get_relevant_triplets(*query_info:* QueryInfo) → List[*Triplet*]

Метод предназначен для извлечения триплетов из графа знаний на основании информации из user-вопроса. На выходе список триплетов не содержит дубликатов (по строковому представлению).

Параметры
query_info (QueryInfo) – Структура данных с информацией о user-вопросе.

Результат
Набор триплетов, извлечённый из графа знаний.

Тип результата
List[*Triplet*]

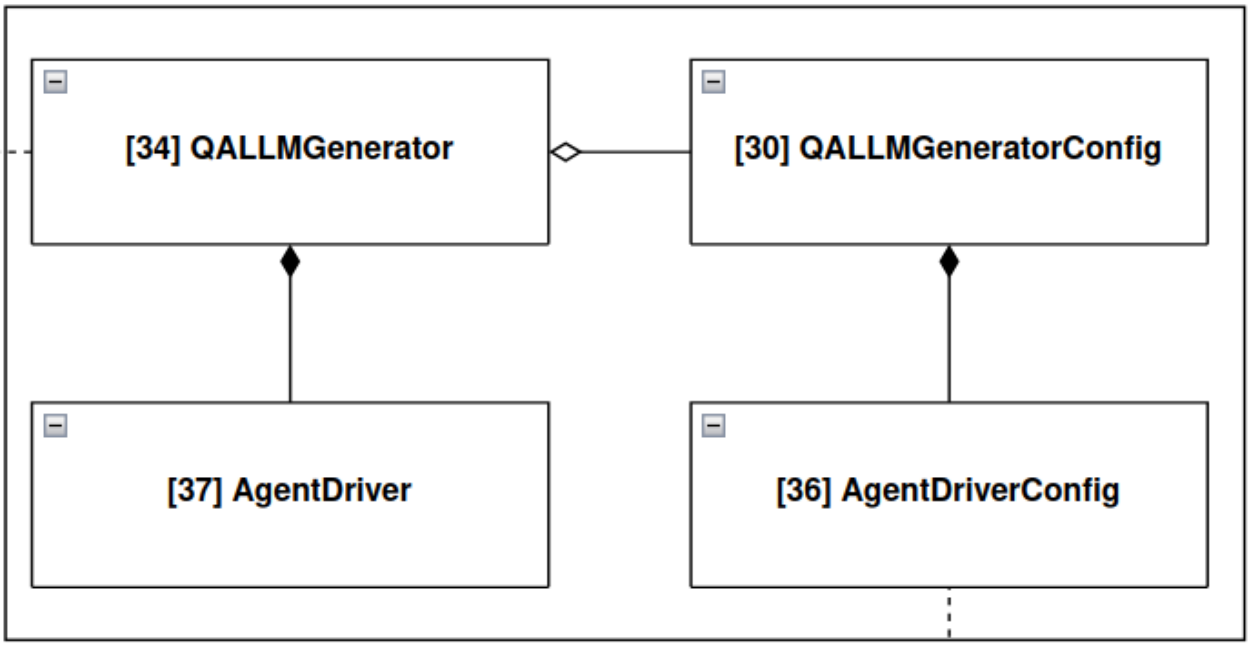
class src.qa_pipeline.knowledge_retriever.utils.**BaseGraphSearchConfig**
Базовые классы: object

Базовая конфигурация алгоритиов по извлечению триплетов из графа знаний.

class src.qa_pipeline.knowledge_retriever.utils.**BaseTripletsFilterConfig**
Базовые классы: object

Базовая конфигурация алгоритмоа по ранжированию/фильтрации триплетов.

Answer Generator



src.qa_pipeline.answer_generator package

Submodules

src.qa_pipeline.answer_generator.QALLMGenerator module

class src.qa_pipeline.answer_generator.QALLMGenerator.**QALLMGeneratorConfig**
Базовые классы: object

Конфигурация «Question Answering»-стадии.

Параметры

- **lang** (*str*) – Язык, который будет использоваться в подаваемом на вход тексте. На основании выбранного языка будут использоваться соответствующие промпты. Если „auto“, то язык определяется автоматически. Значение по умолчанию „auto“.

- **system_prompt** (*dict*) – System-промпт с описание персоны, свойствам которой должен удовлетворять LLM-агент при генерации оветов.
- **user_prompt** (*dict*) – User-промпт для LLM-агента с описанием QA-задачи.
- **answer_parse_func** (*dict*) – Функция разбора результатов генерации LLM-агента.
- **agent_cofig** (*AgentDriverConfig*) – Конфигурация LLM-агента, который будет использоваться в рамках данной стадии. Значение по умолчанию AgentDriverConfig().
- **relation_type** (*List[RelationType]*) – Типы триплетов, которые могут присутствовать в контексте для генерации ответа на user-вопрос. Значение по умолчанию [RelationType.simple, RelationType.hyper, RelationType.episodic].
- **log** (Logger) – Отладочный класс для журналирования/мониторинга поведения инициализируемой комопненты. Значение по умолчанию Logger(QA_LOG_PATH).
- **verbose** (*bool*) – Если True, то информация о поведении класса будет сохраняться в stdout и файл-журналирования (log), иначе только в файл. Значение по умолчанию False.

lang: str = 'auto'

system_prompt: dict

user_prompt: dict

answer_parse_func: dict

agent_cofig: AgentDriverConfig

relation_type: List[*RelationType*]

log: *Logger*

verbose: bool = False

class src.qa_pipeline.answer_generator.QALLMGenerator.**QALLMGenerator**

Базовые классы: object

Верхнеуровневый класс четвёртой стадии QA-конвейера для генерации ответа на user-вопрос, обусловленного извлечённой информацией из памяти (графа знаний) ассистента.

Параметры

config (QALLMGeneratorConfig) – Конфигурация «Answer-generation»-стадии. Значение по умолчанию QALLMGeneratorConfig().

formate_context (*triplets: List[*Triplet*]*) → str

Метод предназначен для предтсавления набора триплетов в виде ненумерованного списка с их строковыми представлениями на естественном языке.

Параметры

triplets (*List[Triplet]*) – Набор триплетов.

Результат

Ненумепованный список со строковыми представлениями триплетов.

Тип результата

str

generate (*query: str*, *context: str*) → Tuple[str, *ReturnInfo*]

Метод предназначен для условной генерации ответа на вопрос.

Параметры

- **query** (*str*) – Вопрос на естественном языке.
- **context** (*str*) – Ненумерованный список дополнительной информации на естественном языке для генерации ответа.

Результат

Кортеж из двух объектов: (1) сгенерированнвй ответ на вопрос; (2) статус выполнения операции с пояснительной информацией.

Тип результата

Tuple[str, *ReturnInfo*]

Submodules

src.qa_pipeline.QAPipeline module

class src.qa_pipeline.**QAPipelineConfig**

Базовые классы: object

Конфигурация QA-конвейера.

Параметры

- **query_parser_config** (QueryLLMParserConfig) – Конфигурация первой стадии QA-конвейера: извлечение сущностей из user-вопроса. Значение по умолчанию QueryLLMParserConfig().
- **knowledge_comparator_config** (KnowledgeComparatorConfig) – Конфигурация второй стадии QA-конвейера: сопоставление (match) сущностей из user-вопроса с информацией в графе знаний. Значение по умолчанию KnowledgeComparatorConfig().
- **knowledge_retriever_config** (KnowledgeRetrieverConfig) – Конфигурация третьей стадии QA-конвейера: извлечение релевантной информации из графа знаний для user-вопроса. Значение по умолчанию KnowledgeRetrieverConfig().
- **answer_generator_config** (QALLMGeneratorConfig) – Конфигурация четвёртой стадии QA-конвейера: условная генерация ответа на user-вопрос. Значение по умолчанию QALLMGeneratorConfig().
- **log** (Logger) – Отладочный класс для журналирования/мониторинга поведения инициализируемой комопненты. Значение по умолчанию Logger(LOG_PATH).
- **verbose** (*bool*) – Если True, то информация о поведении класса будет сохраняться в stdout и файл-журналирования (log), иначе только в файл. Значение по умолчанию False.

query_parser_config: *QueryLLMParserConfig*

knowledge_comparator_config: *KnowledgeComparatorConfig*

knowledge_retriever_config: *KnowledgeRetrieverConfig*

answer_generator_config: *QALLMGeneratorConfig*

log: *Logger*

verbose: bool = False

class src.qa_pipeline.**QAPipeline**

Базовые классы: object

Верхнеуровневый класс QA-конвейера, отвечающий за генерацию ответов на вопросы.

Параметры

- **kg_model** (KnowledgeGraphModel) – Модель памяти (графа знаний) ассистента.
- **config** (QAPipelineConfig) – Конфигурация QA-конвейера. Значение по умолчанию QAPipelineConfig().

answer(*query: str*) → Tuple[str, *ReturnInfo*]

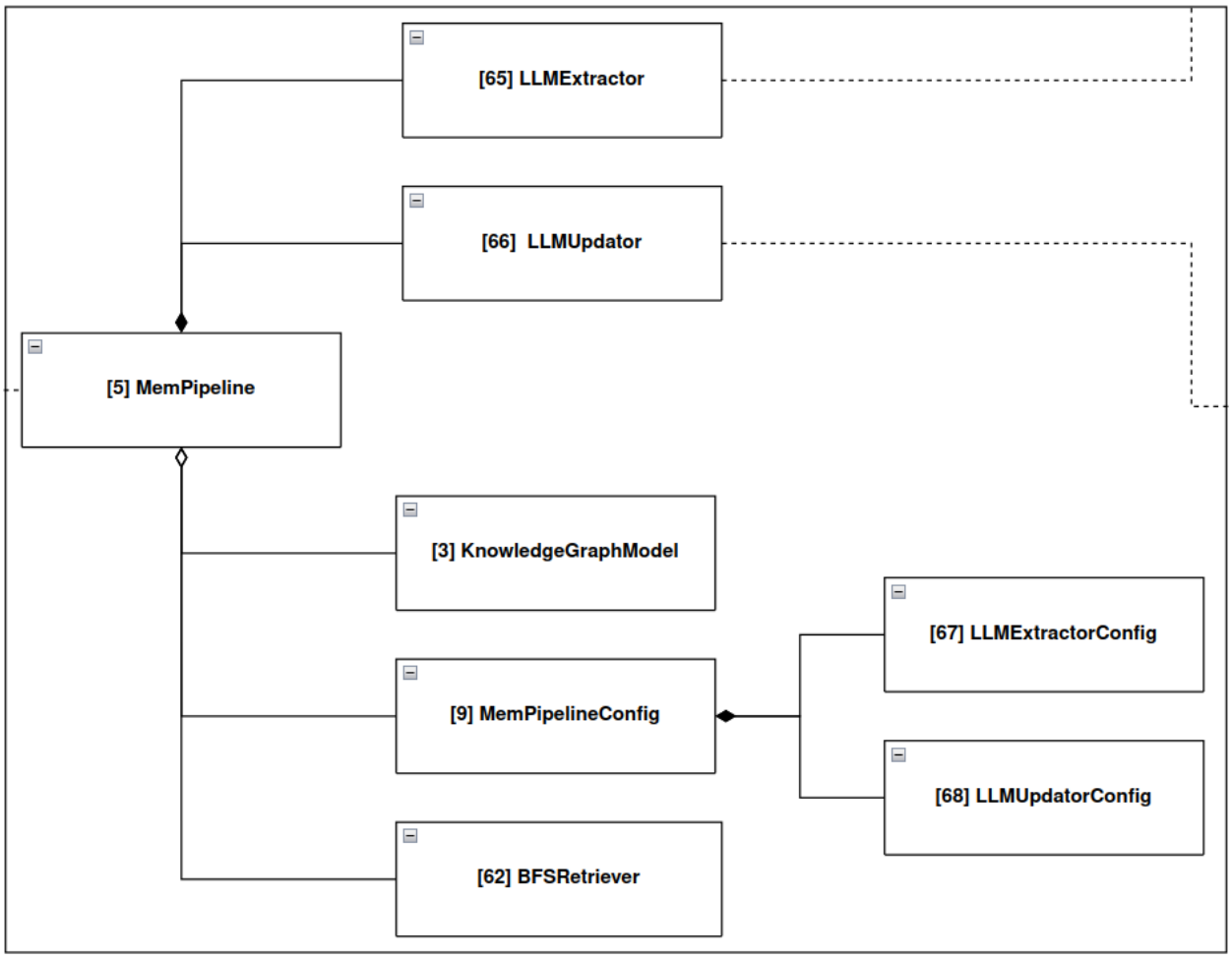
Метод предназначен для генерации ответа на user-вопрос. Ответ обуславливается на информацию из имеющегося графа знаний.

Параметры
query (*str*) – User-вопрос на естественном языке.

Результат
Кортеж из двух объектов: (1) сгенерированный ответ; (2) статус завершения операции с пояснительной информацией.

Тип результата
Tuple[str, *ReturnInfo*]

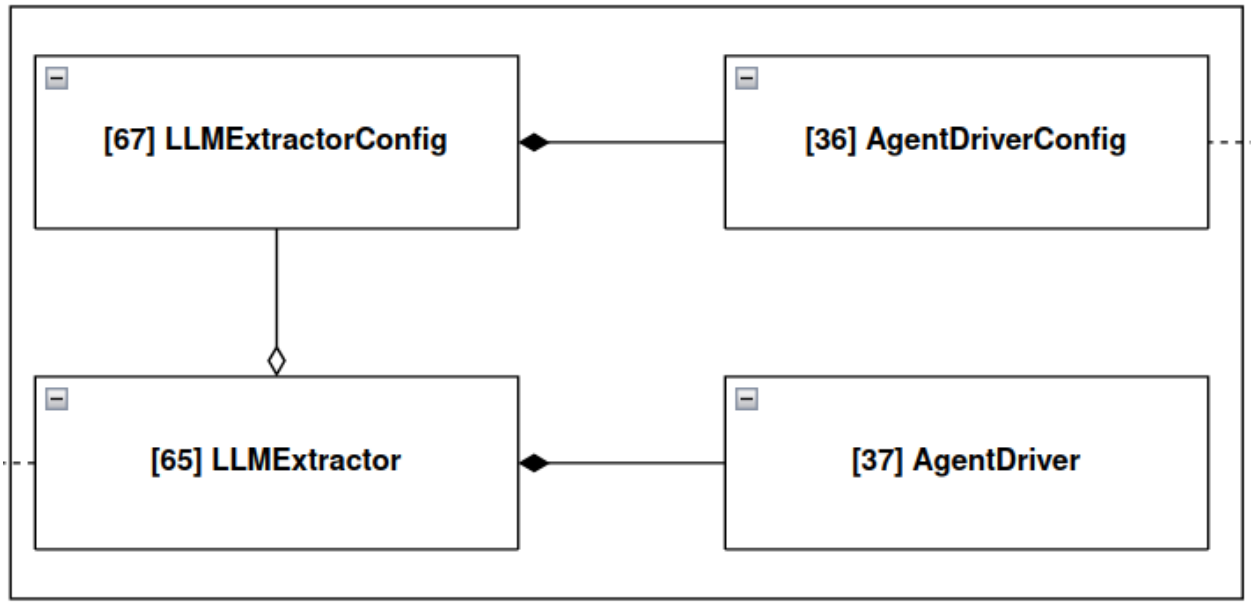
2.4 Memorize-pipeline



2.4.1 src.memorize_pipeline package

Subpackages

Memory Extractor



src.memorize_pipeline.extractor package

Submodules

src.memorize_pipeline.extractor.LLMExtractor module

class src.memorize_pipeline.extractor.LLMExtractor.**LLMExtractorConfig**

Базовые классы: `object`

Конфигурация Extractor-стадии.

Параметры

- **lang** (*str*) – Язык, который будет использоваться в подаваемом на вход тексте. На основании выбранного языка будут использоваться соответствующие промпты. Если „auto“, то язык определяется автоматически. Значение по умолчанию „auto“.
- **agent_config** (*AgentDriverConfig*) – Конфигурация LLM-агента, который будет использоваться в рамках данной стадии.
- **triplet_extract_system_prompt** (*dict*) – System-промпты с описание персоны, свойствам которой должен удовлетворять LLM-агент при генерации оветов по задаче извлечения триплетов.
- **triplet_extract_user_prompt** (*dict*) – User-промпты для LLM-агента с описанием задачи по извлечению триплетов из текста на естественном языке.
- **triplet_parse_func** (*dict*) – Функция разбора результатов генерации LLM-агента по задаче извлечения триплетов.
- **thesis_extract_system_prompt** (*dict*) – System-промпты с описание персоны, свойствам которой должен удовлетворять LLM-агент при генерации оветов по задаче извлечения тезисной информации.
- **thesis_extract_user_prompt** (*dict*) – User-промпты для LLM-агента с описанием задачи по извлечению тезисной информации из текста на естественном языке.

- **thesis_parse_func** (*dict*) – Функция разбора результатов генерации LLM-агента по задаче извлечения тезисной информации.
- **log** (Logger) – Отладочный класс для журналирования/мониторинга поведения инициализируемой комопненты. Значение по умолчанию Logger(MEM_EXTRACT_LOG_PATH).
- **verbose** (*bool*) – Если True, то информация о поведении класса будет сохраняться в stdout и файл-журналирования (log), иначе только в файл. Значение по умолчанию False.

lang: str = 'auto'

agent_config: AgentDriverConfig

triplet_extract_system_prompt: dict

triplet_extract_user_prompt: dict

triplet_parse_func: dict

thesis_extract_system_prompt: dict

thesis_extract_user_prompt: dict

thesis_parse_func: dict

log: *Logger*

verbose: bool = False

class src.memorize_pipeline.extractor.LLMExtractor.**LLMExtractor**

Базовые классы: object

Верхнеуровневый класс первой стадии Memorize-конвейера для извлечения информации (и её приведения в triplet-формат) из слабоструктурированных данных.

Параметры

config (LLMExtractorConfig) – Конфигурация Exctrator-стадии. Значение по умолчанию LLMExtractorConfig().

extract (*text: str*, *need_simple: bool = True*, *need_thesises: bool = True*, *need_episodic: bool = True*, *properties: Dict = {}*) → Tuple[List[*Triplet*], *ReturnInfo*]

Метод предназначен для извлечения информации (в виде триплетов) из слабоструктурированного текста на естественном языке.

Параметры

- **text** (*str*) – Слабоструктурированный текст.
- **need_simple** (*bool, optional*) – Если True, то из входного текста на первой стадии Mem-конвейера будет выполнено извлечение триплетов с типом связи „simple“, иначе False. Значение по умолчанию True.
- **need_thesises** (*bool, optional*) – Если True, то из входного текста на первой стадии Mem-конвейера будет выполнено извлечение триплетов с типом связи „hyper“, иначе False. Значение по умолчанию True.
- **need_episodic** (*bool, optional*) – Если True, то из входного текста на первой стадии Mem-конвейера будет выполнено извлечение триплетов с типом связи „episodic“, иначе False. Значение по умолчанию True.

- **properties** (*Dict, optional*) – Набор свойств, который должен быть сохранён в памяти вместе с извлечённой из текста информацией, Значение по умолчанию dict().

Результат
Кортеж из двух объектов: (1) список извлечённой из текста информации (в виде триплетов); (2) статус завершения операции с пояснительной информацией.

Тип результата
Tuple[List[*Triplet*], *ReturnInfo*]

extract_triplets (*text: str*, *lang: str*, *node_prop={}*, *rel_prop={}*) → Tuple[List[*Triplet*], *ReturnStatus*]

extract_thesises (*text: str*, *lang: str*, *node_prop: Dict = {}*, *rel_prop: Dict = {}*) → Tuple[List[*Triplet*], *ReturnStatus*]

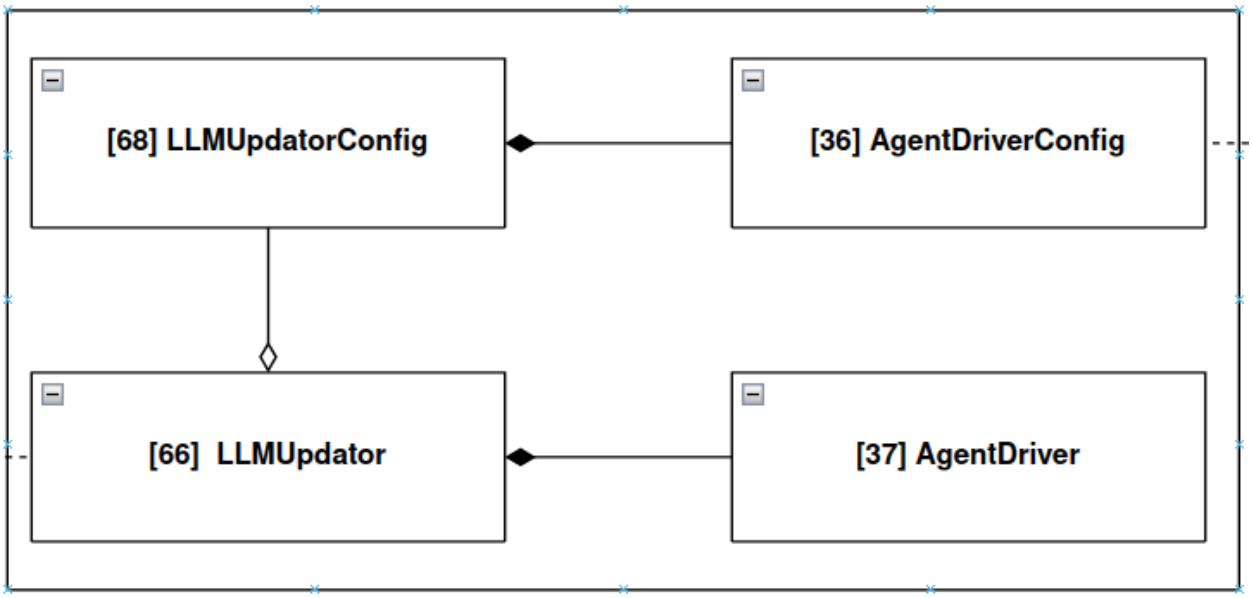
static get_entities_from_triplets (*triplets: List[*Triplet*]*) → List[*Node*]

parse_thesises (*raw_response: str*, *lang: str*, *node_prop: Dict*, *rel_prop: Dict*) → Tuple[List[*Triplet*], *ReturnStatus*]

parse_triplets (*raw_response: str*, *lang: str*, *node_prop: Dict*, *rel_prop: Dict*) → Tuple[List[*Triplet*], *ReturnStatus*]

static get_episodic_relationships (*text: str*, *entities: List[*Node*]*, *node_prop: Dict = {}*, *rel_prop: Dict = {}*) → List[*Triplet*]

Memory Updator



src.memorize_pipeline.updator package

Submodules

src.memorize_pipeline.updator.LLMUpdator module

class src.memorize_pipeline.updator.LLMUpdator.**LLMUpdatorConfig**

Базовые классы: object

Конфигурация Updator-стадии.

Параметры

- **lang** (*str*) – Язык, который будет использоваться в подаваемом на вход тексте. На основании выбранного языка будут использоваться соответствующие промпты. Если „auto“, то язык определяется автоматически. Значение по умолчанию „auto“.
- **agent_config** (*AgentDriverConfig*) – Конфигурация LLM-агента, который будет использоваться в рамках данной стадии.
- **replace_thesis_prompt** (*Dict*) – Значение по умолчанию REPLACE_THESIS_PROMPT.
- **replace_simple_prompt** (*Dict*) – Значение по умолчанию REPLACE_SIMPLE_PROMPT.
- **log** (Logger) – Отладочный класс для журналирования/мониторинга поведения инициализируемой комопненты. Значение по умолчанию Logger(MEM_UPDATE_LOG).
- **verbose** (*bool*) – Если True, то информация о поведении класса будет сохраняться в stdout и файл-журналирования (log), иначе только в файл. Значение по умолчанию False.

lang: str = 'auto'

agent_config: AgentDriverConfig

replace_thesis_prompt: Dict

replace_simple_prompt: Dict

log: ***Logger***

verbose: bool = False

class src.memorize_pipeline.updator.LLMUpdator.**LLMUpdator**

Базовые классы: object

Верхнеуровневый класс первой стадии Memorize-конвейера для актуализации знаний в памяти ассистента.

Параметры

config (LLMUpdatorConfig) – Конфигурация Updator-стадии. Значение по умолчанию LLMUpdatorConfig().

update(*new_triplets*, *replacing_window_width*, *replacing_window_depth*, *need_simple=True*, *need_thesises=True*)

static parse_replacements_simple(*raw_replacements*)

static parse_replacements_thesis(*raw_replacements*)

static get_entities_from_triplets(*triplets*)

static stringify(*triplet*)

static stringify_all(*triplets*)

Submodules

src.memorize_pipeline.MemPipeline module

class src.memorize_pipeline.MemPipeline.**MemPipelineConfig**

Базовые классы: object

Конфигурация Memorize-конвейера.

Параметры

- **extractor_config** (LLMExtractorConfig) – Конфигурация первой стадии Memorize-конвейера: извлечение информации из текстовых данных и приведение их в triplet-формат. Значение по умолчанию LLMExtractorConfig().
- **updator_config** (LLMUpdatorConfig) – Конфигурация второй стадии Mem-конвейера: актуализация знаний в памяти ассистента. Значение по умолчанию LLMUpdatorConfig().
- **log** (Logger) – Отладочный класс для журналирования/мониторинга поведения инициализируемой комопненты. Значение по умолчанию Logger(MEM_LOG_PATH).
- **verbose** (*bool*) – Если True, то информация о поведении класса будет сохраняться в stdout и файл-журналирования (log), иначе только в файл. Значение по умолчанию False.

extractor_config: *LLMExtractorConfig*

updator_config: *LLMUpdatorConfig*

log: *Logger*

verbose: bool = False

class src.memorize_pipeline.MemPipeline.**MemPipeline**

Базовые классы: object

Верхнеуровневый класс Memorize-конвейера, отвечающего за изменение знаний в памяти ассистента.

Параметры

- **kg_model** (KnowledgeGraphModel) – Модель памяти (графа знаний) ассистента.
- **config** (MemPipelineConfig) – Конфигурация Memorize-конвейера. Значение по умолчанию MemPipelineConfig().
- **bfs** (BFSRetriever) – Значение по умолчанию None.

remember(*text: str*, *replacing_window_width: int = 32*, *replacing_window_depth: int = 1*, *need_simple: bool = True*, *need_thesises: bool = True*, *need_episodic: bool = True*, *need_update: bool = False*, *properties: Dict = {}*) → Tuple[List[*Triplet*], *ReturnInfo*]

Метод предназначен для извлечения информации (в виде триплетов) из слабоструктурированного текста и обновление/актуализацию знаний в памяти (графе знаний) ассистена.

Параметры

- **text** (*str*) – Слабоструктурированный текст на естественном языке.
- **replacing_window_width** (*int, optional*) – Значение по умолчанию 32.
- **replacing_window_depth** (*int, optional*) – Значение по умолчанию 1.

- **need_simple** (*bool, optional*) – Если True, то из входного текста на первой стадии Memorize-конвейера будет выполнено извлечение триплетов с типом связи „simple“, иначе False. Значение по умолчанию True.
- **need_thesises** (*bool, optional*) – Если True, то из входного текста на первой стадии Memorize-конвейера будет выполнено извлечение триплетов с типом связи „hyper“, иначе False. Значение по умолчанию True.
- **need_episodic** (*bool, optional*) – Если True, то из входного текста на первой стадии Memorize-конвейера будет выполнено извлечение триплетов с типом связи „episodic“, иначе False. Значение по умолчанию True.
- **need_update** (*bool, optional*) – Значение по умолчанию False.
- **properties** (*Dict, optional*) – Набор свойств, который должен быть сохранён в памяти вместе с извлечённой из текста информацией, Значение по умолчанию dict().

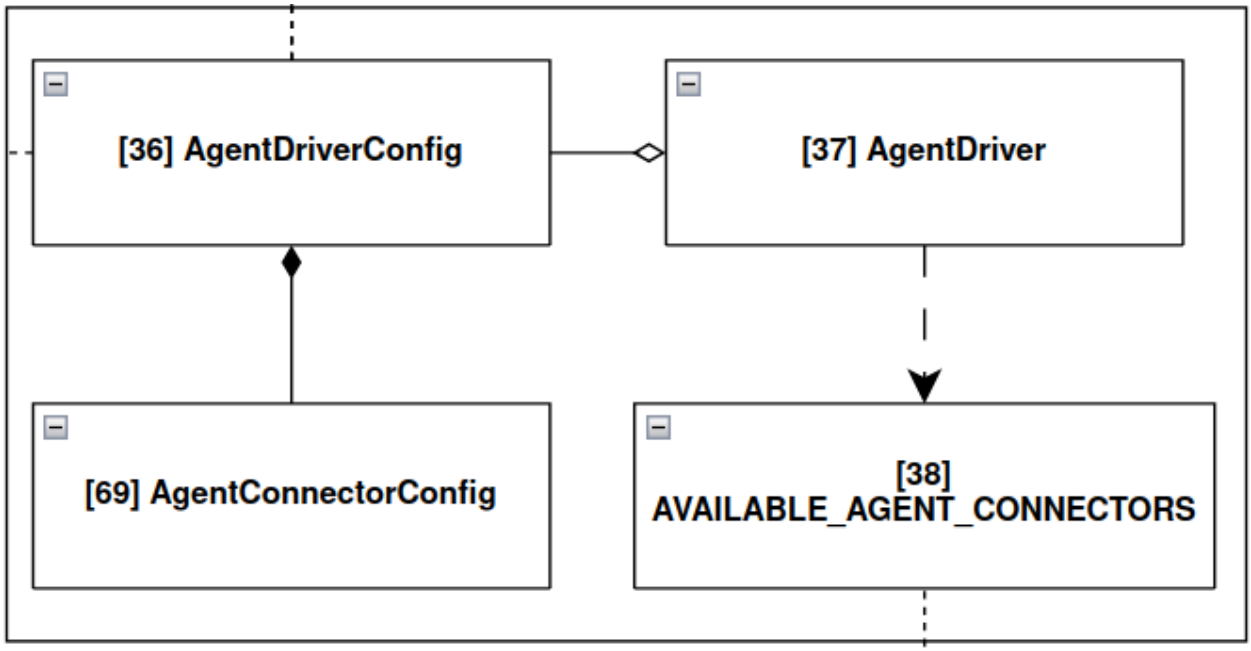
Результат

Кортеж из двух объектов: (1) список с извлечённой из текста информацией (в виде триплетов), который использовался для обновления/актуализации памяти ассистента; (2) статус завершения операции с пояснительной информацией.

Тип результата

Tuple[List[*Triplet*], *ReturnInfo*]

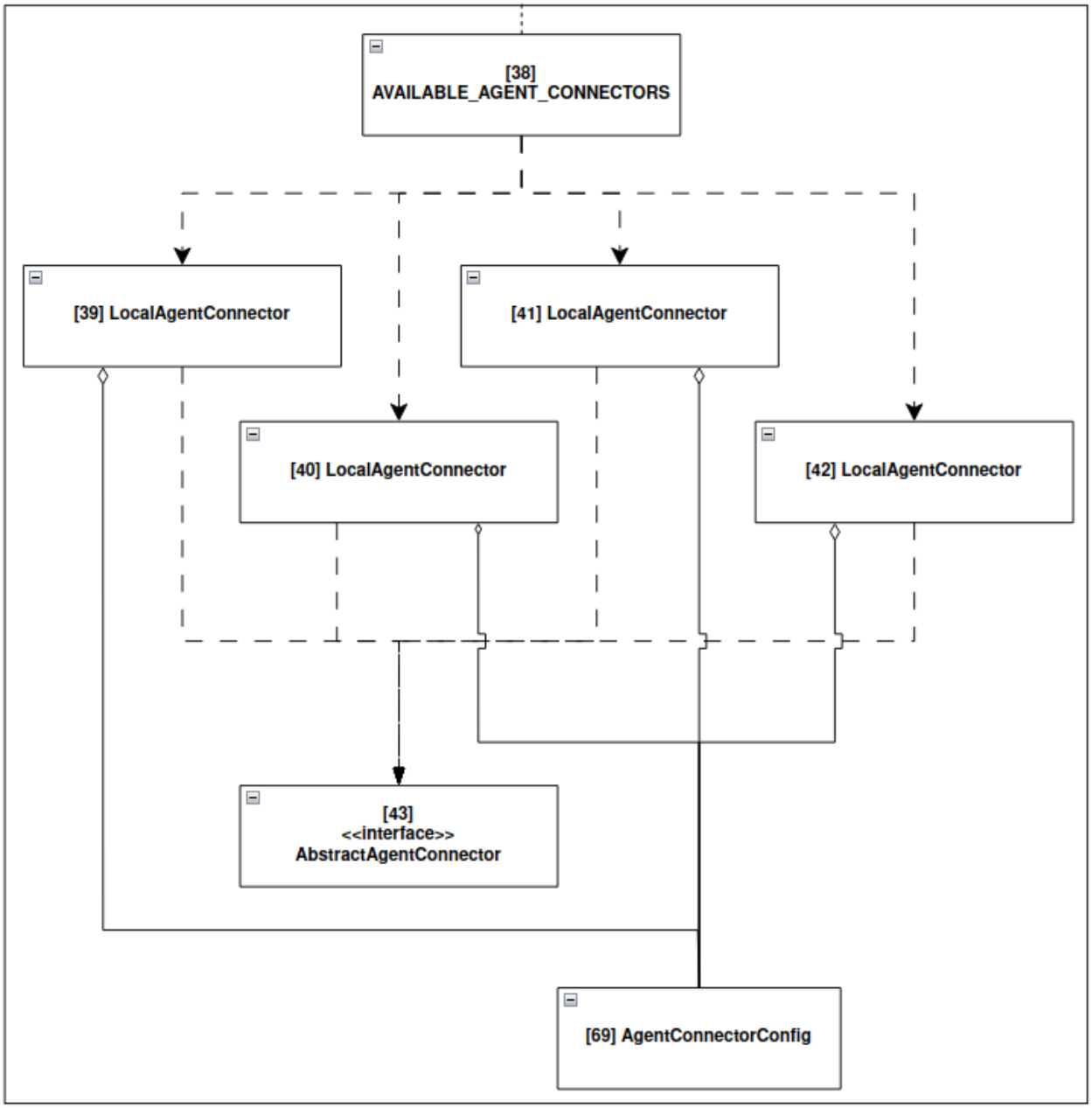
2.5 Agents



2.5.1 src.agents package

Subpackages

Available Agent Connectors



src.agents.connectors package

Submodules

src.agents.connectors.GigaChatConnector module

class src.agents.connectors.GigaChatConnector.**GigaChatConnector**(*config: AgentConnectorConfig = AgentConnectorConfig(gen_strategy={}, credentials={'token': 'OWUwOGUzOWEtMjJiNi00YmMxLTh 'scope': 'GIGACHAT_API_CORP', 'model': 'GigaChat-Pro', 'verify_ssl_certs': False}, ext_params={'timeout': 480})*)

Базовые классы: AbstractAgentConnector

check_connection()

generate(*system_prompt: str*, *user_prompt: str*, *assistant_prompt: str | None = None*) → str

src.agents.connectors.LlamaConnector module

class src.agents.connectors.LlamaConnector.**LlamaConnector**(*config: AgentConnectorConfig = AgentConnectorConfig(gen_strategy={'max_new_ 2048}, credentials={'host': 'http://localhost:45678', 'generate_merhod': 'generate', 'check_method': ''}, ext_params={})*)

Базовые классы: AbstractAgentConnector

check_connection() → bool

generate(*system_prompt: str*, *user_prompt: str*, *assistant_prompt: str | None = None*) → str

src.agents.connectors.LocalAgentConnector module

class src.agents.connectors.LocalAgentConnector.**LocalAgentConnector**(*config: AgentConnectorConfig = AgentConnectorConfig(gen_strateg 2048}, credentials={'model_name_or_pat '../models/Undi95/Meta-Llama-3-8B-Instruct-hf'}, ext_params={'num_workers': 4, 'torch_dtype': torch.bfloat16})*)

Базовые классы: AbstractAgentConnector

check_connection()

generate(*system_prompt: str*, *user_prompt: str*, *assistant_prompt: str | None = None*) → str

src.agents.connectors.OpenAIConnector module

class src.agents.connectors.OpenAIConnector.**OpenAIConnector**(*config: AgentConnectorConfig = AgentConnectorConfig(gen_strategy={}, credentials={'token': 'sk-861mINAavom2SSBqgrI82D4thMOfqT37knC 'model': 'gpt-4o-mini'}, ext_params={})*)

Базовые классы: AbstractAgentConnector

check_connection()

generate(*system_prompt: str*, *user_prompt: str*, *assistant_prompt: str | None = None*) → str

Submodules

src.agents.AgentDriver module

class src.agents.AgentDriver.**AgentDriverConfig**(*name: str = 'gigachat'*, *agent_config: src.agents.utils.AgentConnectorConfig = <factory>*)

Базовые классы: object

name: str = 'gigachat'

agent_config: AgentConnectorConfig

class src.agents.AgentDriver.**AgentDriver**

Базовые классы: object

static connect(*config: AgentDriverConfig = AgentDriverConfig(name='gigachat', agent_config=AgentConnectorConfig(gen_strategy={}, credentials={'token': 'OWUwOGUzOWEtMjJiNi00YmMxLThmMmItNzMwNjM2MTI2YmYxOjg2ODdiOTVhLTZkNDctNGFjOC1 'scope': 'GIGACHAT_API_CORP', 'model': 'GigaChat-Pro', 'verify_ssl_certs': False}, ext_params={'timeout': 480}))*) → AbstractAgentConnector

src.agents.utils module

class src.agents.utils.**AgentConnectorConfig**(*gen_strategy: Dict*, *credentials: Dict = <factory>*, *ext_params: Dict = <factory>*)

Базовые классы: object

gen_strategy: Dict

credentials: Dict

ext_params: Dict

class src.agents.utils.**AbstractAgentConnector**

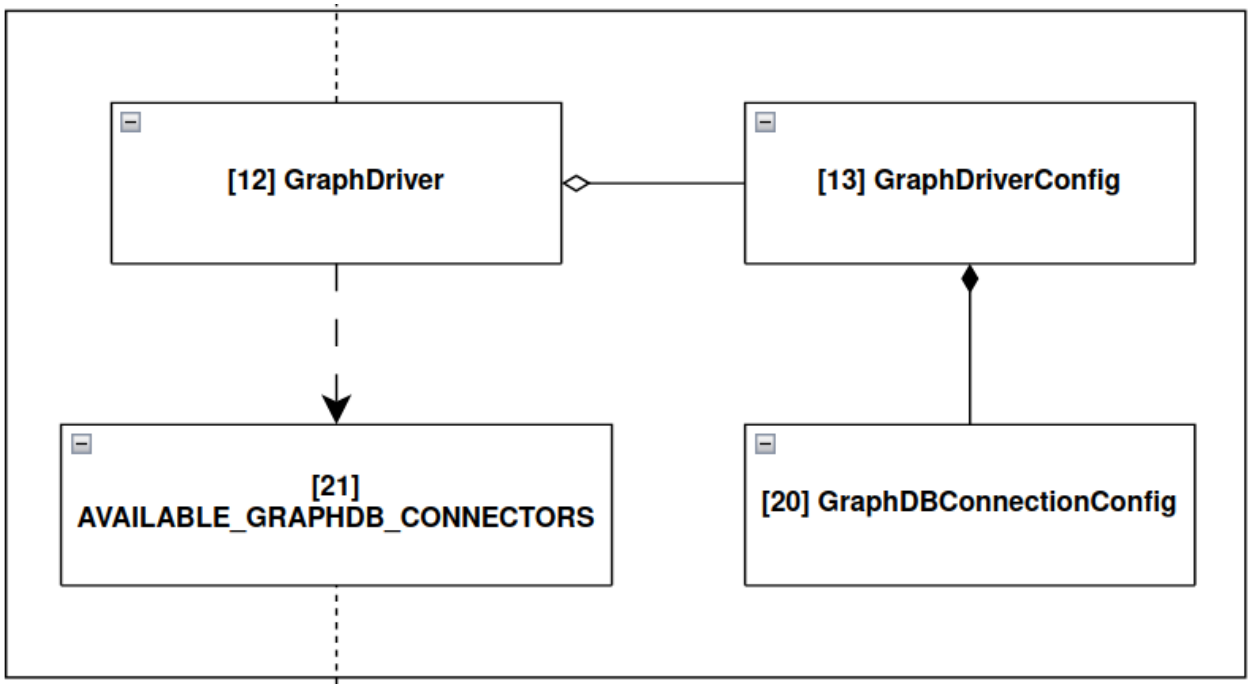
Базовые классы: object

abstract check_connection() → bool

abstract generate(*system_prompt: str*, *user_prompt: str*, *assistant_prompt: str | None = None*) → str

2.6 Database Drivers

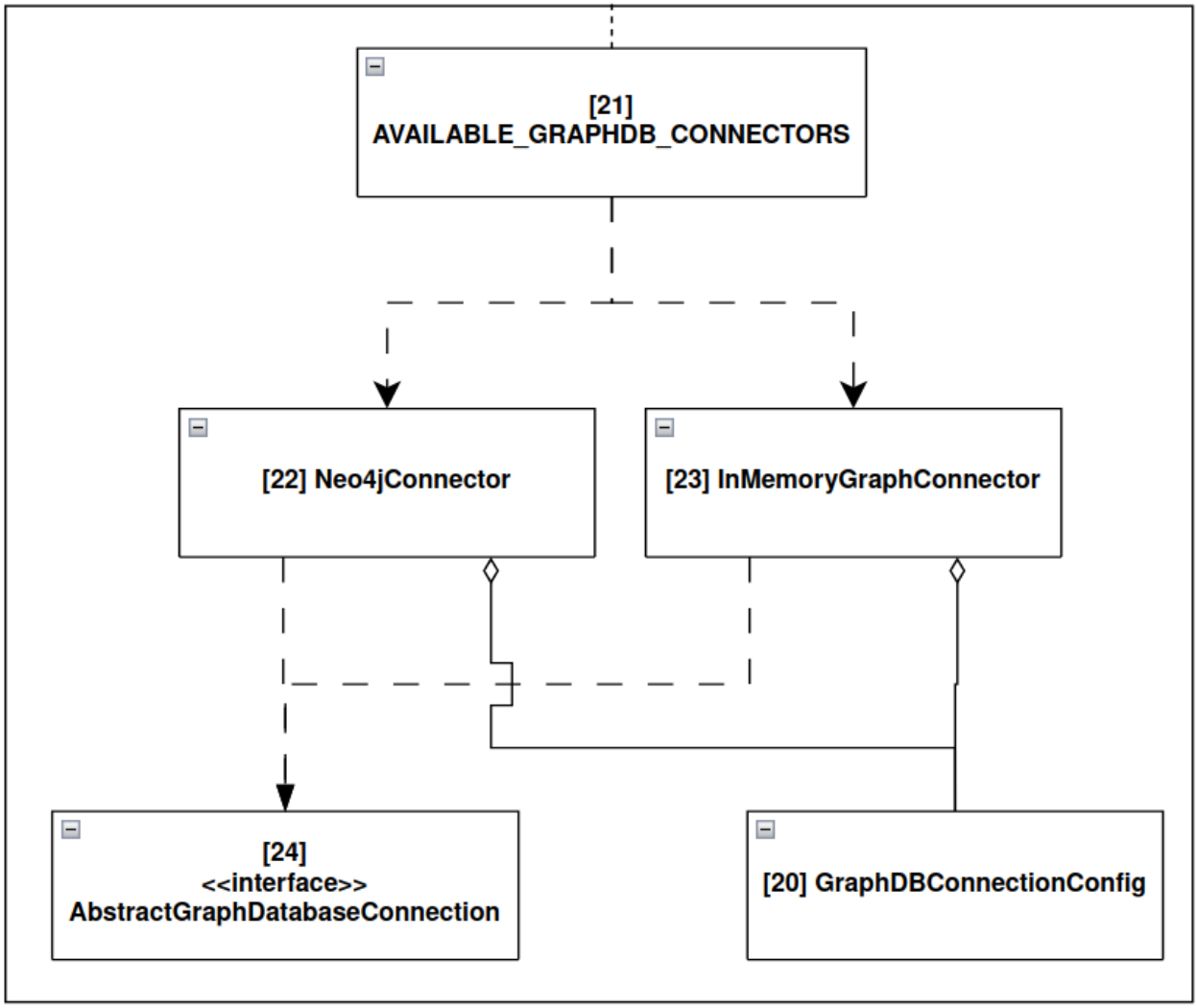
2.6.1 Graph Driver



src.db_drivers.graph_driver package

Subpackages

Available Graph connectors



src.db_drivers.graph_driver.connectors package

Submodules

src.db_drivers.graph_driver.connectors.InMemoryGraphConnector module

class src.db_drivers.graph_driver.connectors.InMemoryGraphConnector.**InMemoryGraphConnector**

Базовые классы: `AbstractGraphDatabaseConnection`

summary

open_connection() → None

Метод предназначен для подключения к бд.

Результат

Статус завершения операции с пояснительной информацией.

Тип результата

ReturnInfo

is_open() → bool

Метод предназначен для проверки статуса подключения к бд.

Результат
Если True, то соединение с бд есть, иначе False.

Тип результата
bool

close_connection() → None

Метод предназначен для разрыва соединения с бд.

Результат
Статус завершения операции с пояснительной информацией.

Тип результата
ReturnInfo

generate_id(*seed: str | None = None*)

create(*triplets: List[*Triplet*]*, *creation_info: Dict = {}*) → None

Метод предназначен для добавления новых объектов в бд. Уникальность добавлемых объектов определяется по полю id. Если объект с таким id уже существует, то затирания информации не произойдёт: в бд останется прежний объект.

Параметры
items (*List[object]*) – Объекты на добавление.

Результат
Статус завершения операции с пояснительной информацией.

Тип результата
ReturnInfo

read(*ids: List[str]*) → List[*Triplet*]

Метод предназначен для получения объектов из бд по их идентификаторам. Если такого идентификатора не существует, то он будет пропущен.

Параметры
ids (*List[str]*) – Идентификаторы объектов, которые нужно получить.

Результат
Список запрошенных объектов.

Тип результата
List[object]

update(*items: List[*Triplet*]*) → None

delete(*ids: List[str]*) → None

Метод предназначен для удаления объектов из бд по их идентификаторам. Если такого идентификатора не существует, то он будет пропущен.

Параметры
ids (*List[object]*) – Идентификаторы объектов, которые нужно удалить.

Результат
Статус завершения операции с пояснительной информацией.

Тип результата
ReturnInfo

get_adjecent_nodes(*base_node_id: str*, *accepted_n_types: List[*NodeType*]*) → List[str]

get_triplets(*node1_id: str*, *node2_id: str*) → List[*Triplet*]

get_triplets_by_name(*subj_names: List[str]*, *obj_names: List[str]*, *obj_type: str*) → List[*Triplet*]

count_items() → int

Метод предназначен для получения количества элементов в таблице бд, к которой было выполнено подключение.

Результат
Структура данных, в которой хранится информация о количестве объектов.

Тип результата
object

item_exist(*id: str*, *id_type='triplet'*) → bool

Метод предназначен для проверки на наличие объекта в бд по его идентификатору.

Параметры
id (*str*) – Иднетификатор объекта.

Результат
Если объект существует, то True, иначе False.

Тип результата
bool

clear() → None

Метод предназначен для удаления содержания таблицы в бд, к которой было выполнено подключение.

Результат
description

Тип результата
ReturnInfo

src.db_drivers.graph_driver.connectors.Neo4jConnector module

class src.db_drivers.graph_driver.connectors.Neo4jConnector.**Neo4jConnector**(*config: GraphDBConnectionCo*

Базовые классы: AbstractGraphDatabaseConnection

summary

open_connection()

Метод предназначен для подключения к бд.

Результат
Статус завершения операции с пояснительной информацией.

Тип результата
ReturnInfo

is_open() → None

Метод предназначен для проверки статуса подключения к бд.

Результат
Если True, то соединение с бд есть, иначе False.

Тип результата
bool

close_connection()

Метод предназначен для разрыва соединения с бд.

Результат
Статус завершения операции с пояснительной информацией.

Тип результата
ReturnInfo

create_node_query(*node:* Node) → str

summary

Параметры
node (Node) – _description_

Результат
description

Тип результата
str

create_rel_query(*triplet:* Triplet) → str

summary

Параметры
triplet (Triplet) – _description_

Результат
description

Тип результата
str

create(*triplets: List[*Triplet*]*, *creation_info: Dict = {}*) → None

Метод предназначен для добавления новых объектов в бд. Уникальность добавлемых объектов определяется по полю id. Если объект с таким id уже существует, то затирания информации не произойдёт: в бд останется прежний объект.

Параметры
items (*List[object]*) – Объекты на добавление.

Результат
Статус завершения операции с пояснительной информацией.

Тип результата
ReturnInfo

read(*ids: List[str]*) → List[*Triplet*]

Метод предназначен для получения объектов из бд по их идентификаторам. Если такого идентификатора не существует, то он будет пропущен.

Параметры
ids (*List[str]*) – Идентификаторы объектов, которые нужно получить.

Результат
Список запрошенных объектов.

Тип результата
List[object]

update (*items: List[Triplet]*) → None

delete (*ids: List[str]*) → None

Метод предназначен для удаления объектов из бд по их идентификаторам. Если такого идентификатора не существует, то он будет пропущен.

Параметры
ids (*List[object]*) – Идентификаторы объектов, которые нужно удалить.

Результат
Статус завершения операции с пояснительной информацией.

Тип результата
ReturnInfo

execute_query (*query: str*, *db_flag: bool = True*)

summary

Параметры

- **query** (*str*) – _description_
- **db_flag** (*bool, optional*) – _description_, defaults to True

Результат
description

Тип результата
type

get_adjecent_nodes (*base_node_id: str*, *accepted_n_types: List[NodeType]*) → List[str]

parse_query_output (*output*)

summary

Параметры
output (*_type_*) – _description_

Результат
description

Тип результата
type

get_triplets (*node1_id: str*, *node2_id: str*) → List[*Triplet*]

get_triplets_by_name (*subj_names: List[str]*, *obj_names: List[str]*, *obj_type: str*) → List[*Triplet*]

count_items () → int

Метод предназначен для получения количества элементов в таблице бд, к которой было выполнено подключение.

Результат
Структура данных, в которой хранится информация о количестве объектов.

Тип результата
object

item_exist (*id: str*, *id_type='triplet'*) → bool

Метод предназначен для проверки на наличие объекта в бд по его идентификатору.

Параметры
id (*str*) – Иднетификатор объекта.

Результат
Если объект существует, то True, иначе False.

Тип результата
bool

clear() → None

Метод предназначен для удаления содержания таблицы в бд, к которой было выполнено подключение.

Результат
description

Тип результата
ReturnInfo

Submodules

src.db_drivers.graph_driver.GraphDriver module

class src.db_drivers.graph_driver.GraphDriver.**GraphDriverConfig**(*db_vendor: str = 'neo4j'*, *db_config: src.db_drivers.graph_driver.utils.GraphI = <factory>*)

Базовые классы: object

db_vendor: str = 'neo4j'

db_config: GraphDBConnectionConfig

class src.db_drivers.graph_driver.GraphDriver.**GraphDriver**

Базовые классы: object

static connect(*config: GraphDriverConfig = GraphDriverConfig(db_vendor='neo4j', db_config=GraphDBConnectionConfig(db_info={'db': 'default_db', 'table': 'default_table'}, params={'user': 'neo4j', 'pwd': 'password'}, need_to_clear=False, uri='bolt://localhost:7687'))*) → AbstractGraphDatabaseConnection

src.db_drivers.graph_driver.utils module

class src.db_drivers.graph_driver.utils.**GraphDBConnectionConfig**(*db_info: Dict = <factory>*, *params: Dict = <factory>*, *need_to_clear: bool = False*, *uri: str = None*)

Базовые классы: *BaseDatabaseConfig*

uri: str = None

class src.db_drivers.graph_driver.utils.**AbstractGraphDatabaseConnection**

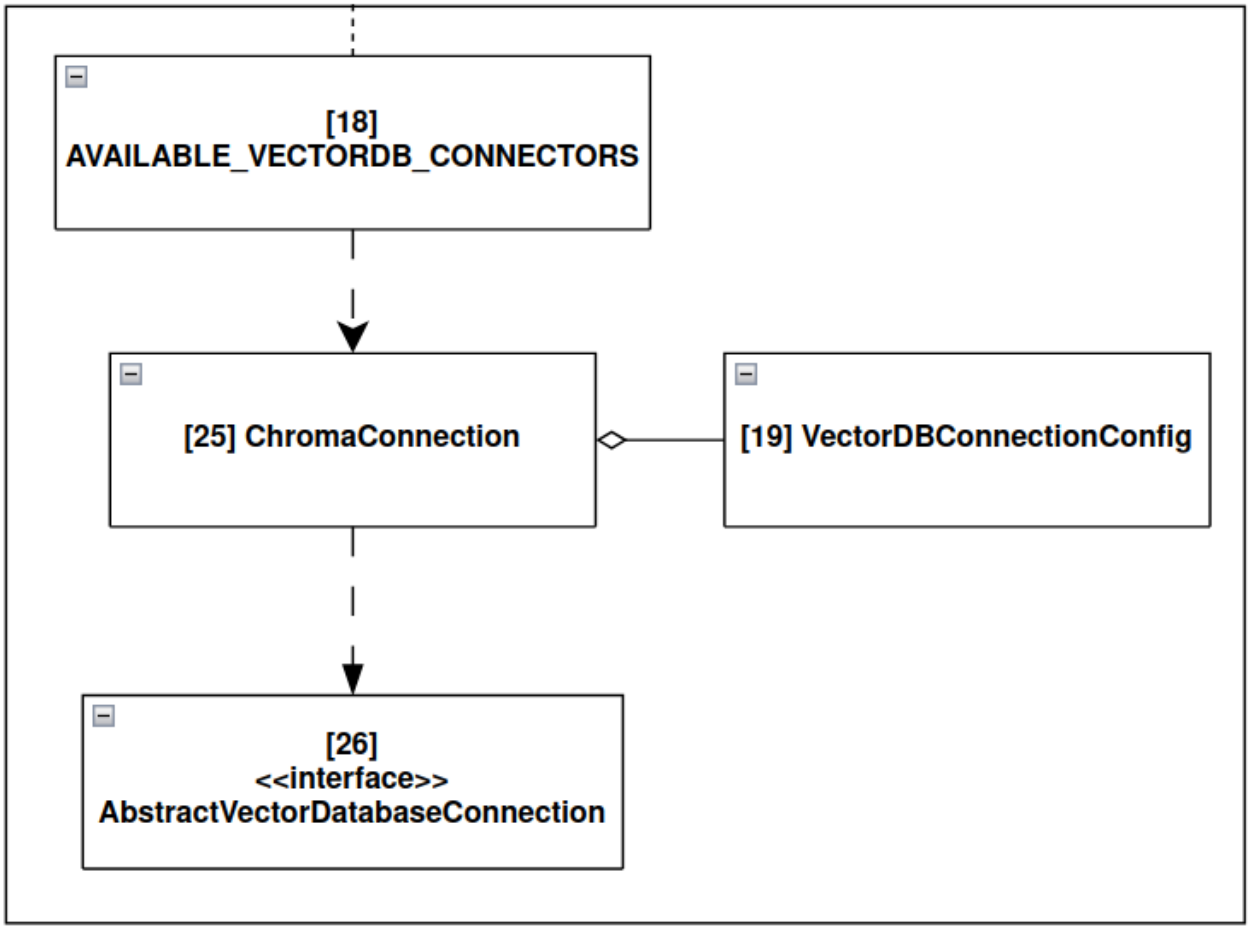
Базовые классы: *AbstractDatabaseConnection*

abstract get_adjecent_nodes(*base_node_id: str*, *parent_node_id: str*, *accepted_n_types: List[*NodeType*]*) → List[str]

abstract get_triplets_by_name(*subj_name: str*, *obj_name: str*, *obj_type*) → List[*Triplet*]

abstract get_triplets(*node1_id: str*, *node2_id: str*) → List[*Triplet*]

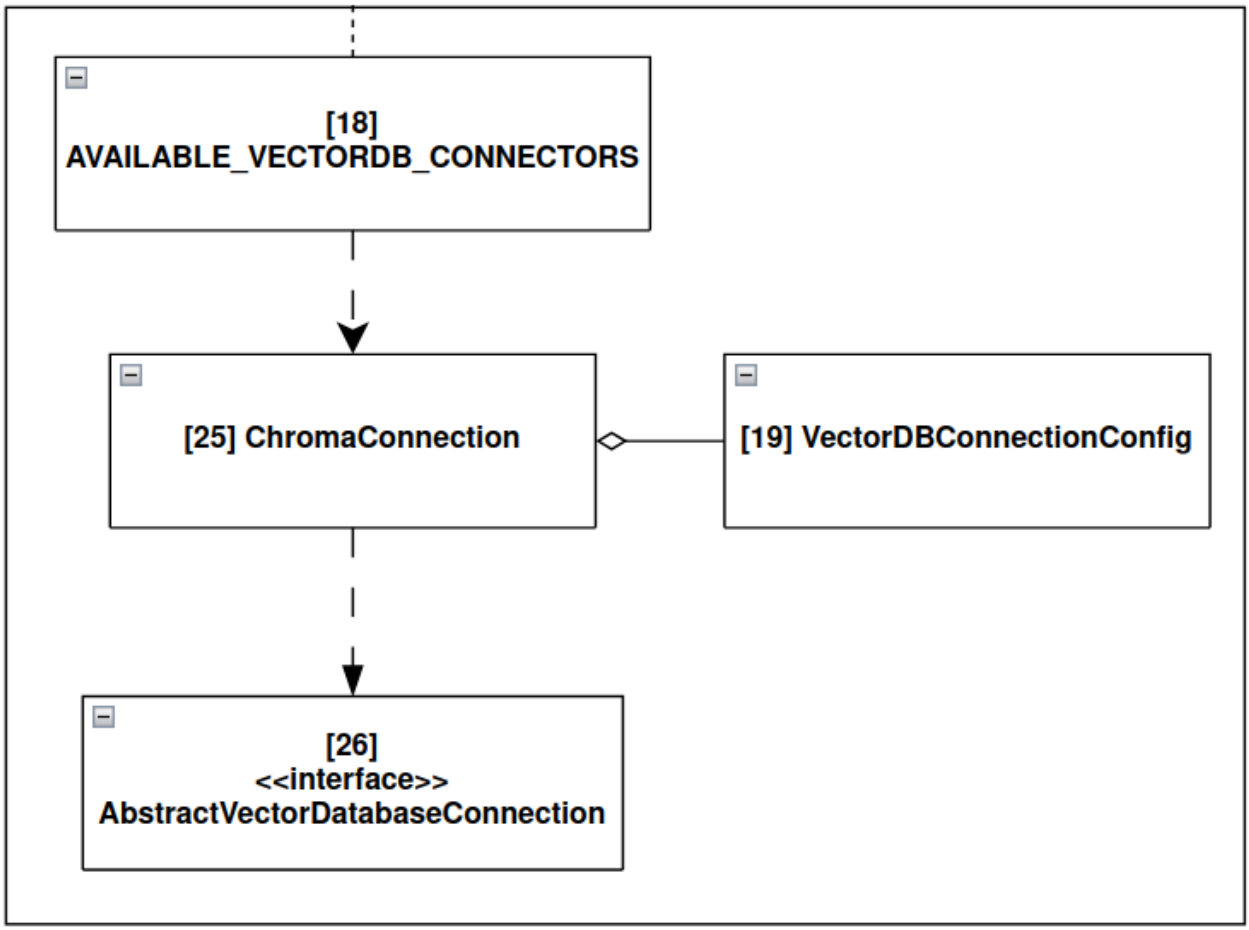
2.6.2 Vector Driver



src.db_drivers.vector_driver package

Subpackages

Available Vector Connectors



src.db_drivers.vector_driver.connectors package

Submodules

src.db_drivers.vector_driver.connectors.ChromaConnector module

class src.db_drivers.vector_driver.connectors.ChromaConnector.**ChromaConnection**(*config: VectorDBConnec*

Базовые классы: `AbstractVectorDatabaseConnection`

Класс предназначен для взаимодействия с векторной базой ChromaDB.

open_connection() → None

Метод предназначен для подключения к бд.

Результат

Статус завершения операции с пояснительной информацией.

Тип результата

ReturnInfo

is_open() → bool

Метод предназначен для проверки статуса подключения к бд.

Результат
Если True, то соединение с бд есть, иначе False.

Тип результата
bool

close_connection() → None

Метод предназначен для разрыва соединения с бд.

Результат
Статус завершения операции с пояснительной информацией.

Тип результата
ReturnInfo

create(*items: List[VectorDBInstance]*) → None

Метод предназначен для добавления новых объектов в бд. Уникальность добавлемых объектов определяется по полю id. Если объект с таким id уже существует, то затирания информации не произойдёт: в бд останется прежний объект.

Параметры
items (*List[object]*) – Объекты на добавление.

Результат
Статус завершения операции с пояснительной информацией.

Тип результата
ReturnInfo

read(*ids: List[str]*, *includes: List[str] = ['embeddings', 'documents']*, ***kwargs*) → List[VectorDBInstance]

Метод предназначен для получения объектов из бд по их идентификаторам. Если такого идентификатора не существует, то он будет пропущен.

Параметры
ids (*List[str]*) – Идентификаторы объектов, которые нужно получить.

Результат
Список запрошенных объектов.

Тип результата
List[object]

update() → None

delete(*ids: List[str]*, ***kwargs*) → None

Метод предназначен для удаления объектов из бд по их идентификаторам. Если такого идентификатора не существует, то он будет пропущен.

Параметры
ids (*List[object]*) – Идентификаторы объектов, которые нужно удалить.

Результат
Статус завершения операции с пояснительной информацией.

Тип результата
ReturnInfo

retrieve(*query_instances: List[VectorDBInstance]*, *n_results: int = 50*, *includes: List[str] = ['embeddings', 'documents', 'metadatas']*, ***kwargs*) → List[List[Tuple[float, VectorDBInstance]]]

count_items() → int

Метод предназначен для получения количества элементов в таблице бд, к которой было выполнено подключение.

Результат
Структура данных, в которой хранится информация о количестве объектов.

Тип результата
object

item_exist(*id: str*) → bool

Метод предназначен для проверки на наличие объекта в бд по его идентификатору.

Параметры
id (*str*) – Иднетификатор объекта.

Результат
Если объект существуют, то True, иначе False.

Тип результата
bool

clear() → None

Метод предназначен для удаления содержания таблицы в бд, к которой было выполнено подключение.

Результат
description

Тип результата
ReturnInfo

src.db_drivers.vector_driver.connectors.MilvusConnector module

class src.db_drivers.vector_driver.connectors.MilvusConnector.**MilvusConnection**

Базовые классы: AbstractVectorDatabaseConnection

Submodules

src.db_drivers.vector_driver.VectorDriver module

class src.db_drivers.vector_driver.VectorDriver.**VectorDriverConfig**(*db_vendor: str = 'chroma'*, *db_config: src.db_drivers.vector_driver.utils.Ve = <factory>*)

Базовые классы: object

db_vendor: str = 'chroma'

db_config: VectorDBConnectionConfig

class src.db_drivers.vector_driver.VectorDriver.**VectorDriver**

Базовые классы: object

static connect(*config: VectorDriverConfig = VectorDriverConfig(db_vendor='chroma', db_config=VectorDBConnectionConfig(db_info={'db': 'default_db', 'table': 'default_table'}, params={'hnsw:space': 'ip'}, need_to_clear=False, path='../data/graph_structures/default_vectorstore'))*) → AbstractVectorDatabaseConnection

src.db_drivers.vector_driver.embedders module

class src.db_drivers.vector_driver.embedders.**EmbedderModelConfig**(*model_name_or_path: str = '../models/intfloat/multilingual-e5-small', prompts: Dict = <factory>, device: str = 'cuda', normalize_embeddings: bool = True*)

Базовые классы: object

model_name_or_path: str = '../models/intfloat/multilingual-e5-small'

prompts: Dict

device: str = 'cuda'

normalize_embeddings: bool = True

class src.db_drivers.vector_driver.embedders.**EmbedderModel**(*config: EmbedderModelConfig | None = None*)

Базовые классы: object

encode_queries(*queries: List[str], **kwargs*) → List[List[float]]

encode_passages(*passages: List[str], **kwargs*) → List[List[float]]

src.db_drivers.vector_driver.utils module

class src.db_drivers.vector_driver.utils.**VectorDBConnectionConfig**(*db_info: Dict = <factory>, params: Dict = <factory>, need_to_clear: bool = False, path: str = None*)

Базовые классы: *BaseDatabaseConfig*

path: str = None

params: Dict

class src.db_drivers.vector_driver.utils.**VectorDBInstance**(*id: str = None, document: str = None, embedding: List[float] = None, metadata: Dict = <factory>*)

Базовые классы: object

id: str = None

document: str = None

embedding: List[float] = None

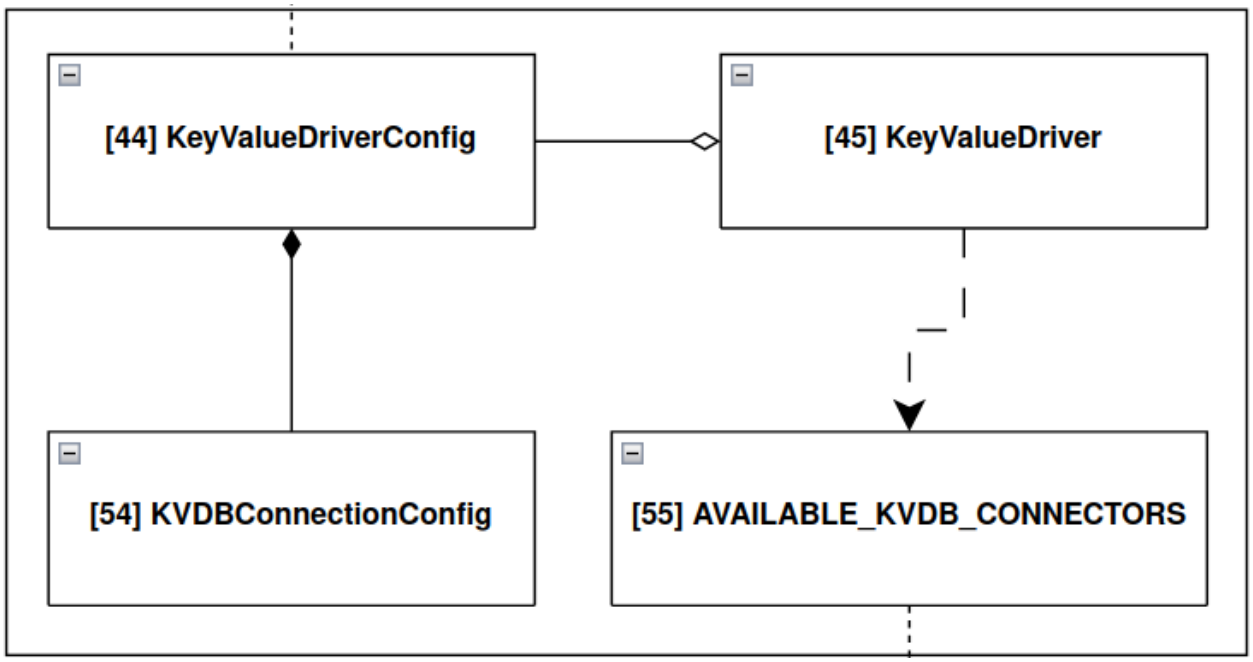
metadata: Dict

class src.db_drivers.vector_driver.utils.**AbstractVectorDatabaseConnection**

Базовые классы: *AbstractDatabaseConnection*

abstract retrieve(*queries: List[VectorDBInstance]*, *n_results: int*, *includes: List[str]*, ***kwargs*) → List[List[Tuple[float, VectorDBInstance]]]

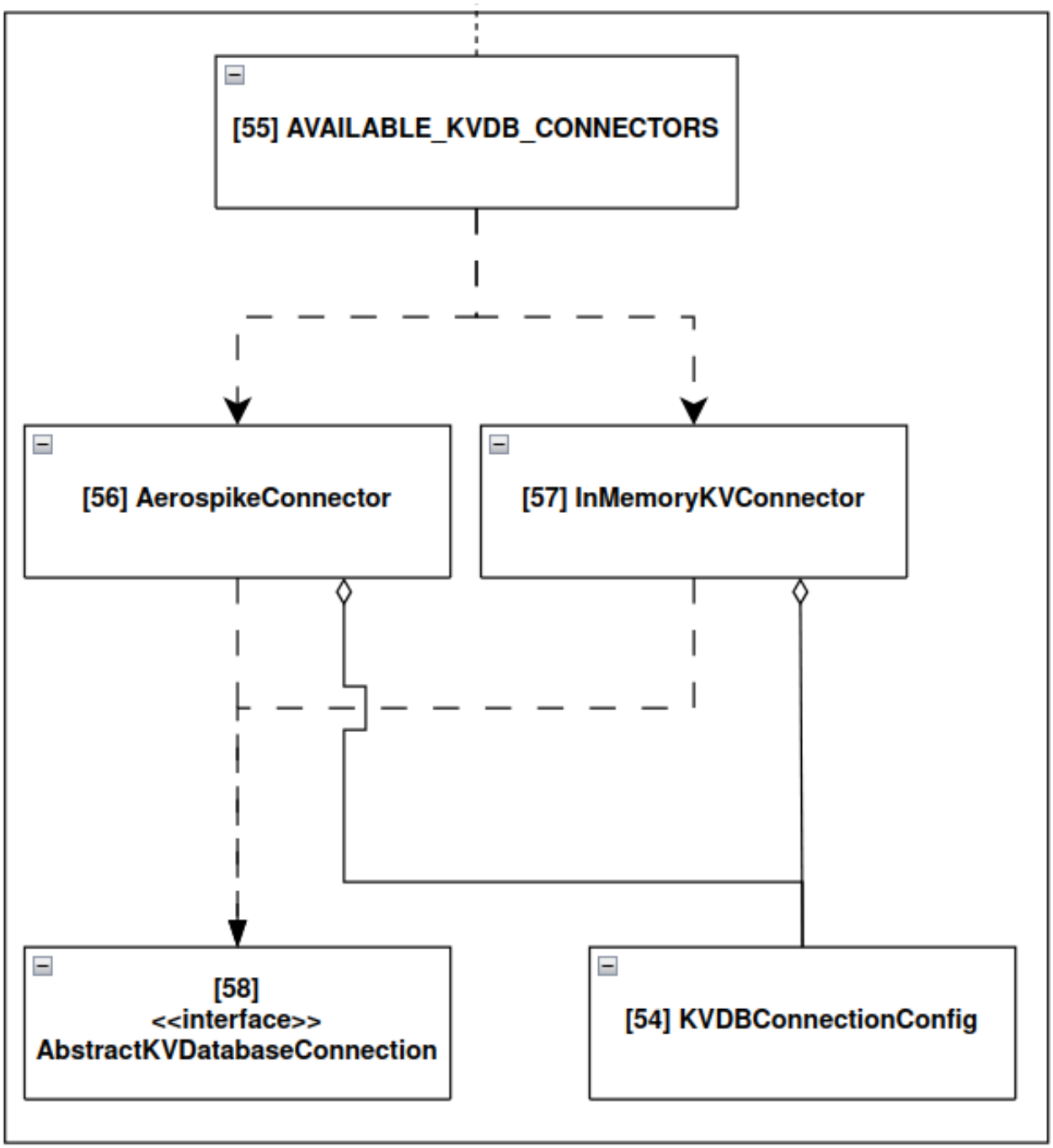
2.6.3 KeyValue Driver



src.db_drivers.kv_driver package

Subpackages

Available KeyValue Connectors



src.db_drivers.kv_driver.connectors package

Submodules

src.db_drivers.kv_driver.connectors.AerospikeConnector module

class src.db_drivers.kv_driver.connectors.AerospikeConnector.**AerospikeConnector**(*config: KVDBConnectio = KVDBConnectio 'default_db', 'table': 'default_table'}, params={}, need_to_clear= host='aerospike port=3000)*)

Базовые классы: AbstractKVDatabaseConnection

summary

open_connection() → None

Метод предназначен для подключения к бд.

Результат
Статус завершения операции с пояснительной информацией.

Тип результата
ReturnInfo

is_open() → bool

Метод предназначен для проверки статуса подключения к бд.

Результат
Если True, то соединение с бд есть, иначе False.

Тип результата
bool

close_connection() → None

Метод предназначен для разрыва соединения с бд.

Результат
Статус завершения операции с пояснительной информацией.

Тип результата
ReturnInfo

create(*items: List[KeyValueDBInstance]*) → None

Метод предназначен для добавления новых объектов в бд. Уникальность добавлемых объектов определяется по полю id. Если объект с таким id уже существует, то затирания информации не произойдёт: в бд останется прежний объект.

Параметры
items (*List[object]*) – Объекты на добавление.

Результат
Статус завершения операции с пояснительной информацией.

Тип результата
ReturnInfo

read(*ids: List[str]*) → List[KeyValueDBInstance]

Метод предназначен для получения объектов из бд по их идентификаторам. Если такого идентификатора не существует, то он будет пропущен.

Параметры
ids (*List[str]*) – Идентификаторы объектов, которые нужно получить.

Результат
Список запрошенных объектов.

Тип результата
List[object]

update(*items: List[KeyValueDBInstance]*) → None

delete(*ids: List[str]*, *durable_delete: bool = False*) → None

Метод предназначен для удаления объектов из бд по их идентификаторам. Если такого идентификатора не существует, то он будет пропущен.

Параметры
ids (*List[object]*) – Идентификаторы объектов, которые нужно удалить.

Результат
Статус завершения операции с пояснительной информацией.

Тип результата
ReturnInfo

clear() → None

Метод предназначен для удаления содержания таблицы в бд, к которой было выполнено подключение.

Результат
description

Тип результата
ReturnInfo

item_exist(*id: str*) → bool

Метод предназначен для проверки на наличие объекта в бд по его идентификатору.

Параметры
id (*str*) – Иднетификатор объекта.

Результат
Если объект существуюет, то True, иначе False.

Тип результата
bool

count_items() → int

Метод предназначен для получения количества элементов в таблице бд, к которой было выполнено подключение.

Результат
Структура данных, в которой хранится информация о количестве объектов.

Тип результата
object

src.db_drivers.kv_driver.connectors.InMemoryKVConnector module

class src.db_drivers.kv_driver.connectors.InMemoryKVConnector.**InMemoryKVConnector**(*config: KVDBConne = KVDBConne 'default_db', 'table': 'default_tabl params={'kv 'inmemory_s 'load_from_ False, 'load_dump_ '.', 'save_on_dis True, 'save_dump_ '.'}, need_to_clea host='localh port=None)*)

Базовые классы: AbstractKVDatabaseConnection

summary

open_connection() → None

Метод предназначен для подключения к бд.

Результат
Статус завершения операции с пояснительной информацией.

Тип результата
ReturnInfo

is_open() → bool

Метод предназначен для проверки статуса подключения к бд.

Результат
Если True, то соединение с бд есть, иначе False.

Тип результата
bool

close_connection() → None

Метод предназначен для разрыва соединения с бд.

Результат
Статус завершения операции с пояснительной информацией.

Тип результата
ReturnInfo

create(*items: List[KeyValueDBInstance]*) → None

Метод предназначен для добавления новых объектов в бд. Уникальность добавлемых объектов определяется по полю id. Если объект с таким id уже существует, то затирания информации не произойдёт: в бд останется прежний объект.

Параметры
items (*List[object]*) – Объекты на добавление.

Результат
Статус завершения операции с пояснительной информацией.

Тип результата
ReturnInfo

read (*ids: List[str]*) → List[KeyValueDBInstance]

Метод предназначен для получения объектов из бд по их идентификаторам. Если такого идентификатора не существует, то он будет пропущен.

Параметры
ids (*List[str]*) – Идентификаторы объектов, которые нужно получить.

Результат
Список запрошенных объектов.

Тип результата
List[object]

update (*items: List[KeyValueDBInstance]*) → None

delete (*ids: List[str]*)

Метод предназначен для удаления объектов из бд по их идентификаторам. Если такого идентификатора не существует, то он будет пропущен.

Параметры
ids (*List[object]*) – Идентификаторы объектов, которые нужно удалить.

Результат
Статус завершения операции с пояснительной информацией.

Тип результата
ReturnInfo

clear ()

Метод предназначен для удаления содержания таблицы в бд, к которой было выполнено подключение.

Результат
description

Тип результата
ReturnInfo

count_items () → int

Метод предназначен для получения количества элементов в таблице бд, к которой было выполнено подключение.

Результат
Структура данных, в которой хранится информация о количестве объектов.

Тип результата
object

item_exist (*id: str*)

Метод предназначен для проверки на наличие объекта в бд по его идентификатору.

Параметры
id (*str*) – Иднетификатор объекта.

Результат
Если объект существуюет, то True, иначе False.

Тип результата
bool

src.db_drivers.kv_driver.connectors.MixedKVConnector module

class src.db_drivers.kv_driver.connectors.MixedKVConnector.**MixedKVConnection**

Базовые классы: AbstractKVDatabaseConnection

src.db_drivers.kv_driver.connectors.MongoConnector module

class src.db_drivers.kv_driver.connectors.MongoConnector.**MongoConnector**(*config: KVDBConnectionConfig = KVDBConnectionConfig(db_ 'personalai', 'collection': 'astar_ip'}, params={'user': 'mongo_root', 'password': 'root_password'}, need_to_clear=False, host='localhost', port=27017)*)

Базовые классы: AbstractKVDatabaseConnection

open_connection()

Метод предназначен для подключения к бд.

Результат
Статус завершения операции с пояснительной информацией.

Тип результата
ReturnInfo

is_open() → bool

Метод предназначен для проверки статуса подключения к бд.

Результат
Если True, то соединение с бд есть, иначе False.

Тип результата
bool

close_connection()

Метод предназначен для разрыва соединения с бд.

Результат
Статус завершения операции с пояснительной информацией.

Тип результата
ReturnInfo

create(*items: List[KeyValueDBInstance]*) → None

Метод предназначен для добавления новых объектов в бд. Уникальность добавлемых объектов определяется по полю id. Если объект с таким id уже существует, то затирания информации не произойдёт: в бд останется прежний объект.

Параметры
items (*List[object]*) – Объекты на добавление.

Результат
Статус завершения операции с пояснительной информацией.

Тип результата
ReturnInfo

delete (*ids: List[str]*) → None

Метод предназначен для удаления объектов из бд по их идентификаторам. Если такого идентификатора не существует, то он будет пропущен.

Параметры
ids (*List[object]*) – Идентификаторы объектов, которые нужно удалить.

Результат
Статус завершения операции с пояснительной информацией.

Тип результата
ReturnInfo

read (*ids: List[str]*) → List[Dict]

Метод предназначен для получения объектов из бд по их идентификаторам. Если такого идентификатора не существует, то он будет пропущен.

Параметры
ids (*List[str]*) – Идентификаторы объектов, которые нужно получить.

Результат
Список запрошенных объектов.

Тип результата
List[object]

item_exist (*id: str*) → bool

Метод предназначен для проверки на наличие объекта в бд по его идентификатору.

Параметры
id (*str*) – Иднетификатор объекта.

Результат
Если объект существуюет, то True, иначе False.

Тип результата
bool

count_items () → int

Метод предназначен для получения количества элементов в таблице бд, к которой было выполнено подключение.

Результат
Структура данных, в которой хранится информация о количестве объектов.

Тип результата
object

clear ()

Метод предназначен для удаления содержания таблицы в бд, к которой было выполнено подключение.

Результат
description

Тип результата
ReturnInfo

src.db_drivers.kv_driver.connectors.RedisConnector module

class src.db_drivers.kv_driver.connectors.RedisConnector.**RedisConnection**

Базовые классы: AbstractKVDatabaseConnection

Submodules

src.db_drivers.kv_driver.KeyValueDriver module

class src.db_drivers.kv_driver.KeyValueDriver.**KeyValueDriverConfig**(*db_vendor: str = 'aerospike', db_config: src.db_drivers.kv_driver.utils.KVDB = <factory>*)

Базовые классы: object

db_vendor: str = 'aerospike'

db_config: KVDBConnectionConfig

class src.db_drivers.kv_driver.KeyValueDriver.**KeyValueDriver**

Базовые классы: object

static connect(*config: KeyValueDriverConfig = KeyValueDriverConfig(db_vendor='aerospike', db_config=KVDBConnectionConfig(db_info={'db': 'default_db', 'table': 'default_table'}, params={}, need_to_clear=False, host='aerospikelservice', port=3000))*) → AbstractKVDatabaseConnection

src.db_drivers.kv_driver.utils module

class src.db_drivers.kv_driver.utils.**KVDBConnectionConfig**(*db_info: Dict = <factory>, params: Dict = <factory>, need_to_clear: bool = False, host: str = None, port: str = None*)

Базовые классы: *BaseDatabaseConfig*

host: str = None

port: str = None

class src.db_drivers.kv_driver.utils.**KeyValueDBInstance**(*id: str, metadata: Dict*)

Базовые классы: object

id: str

metadata: Dict

class src.db_drivers.kv_driver.utils.**AbstractKVDatabaseConnection**

Базовые классы: *AbstractDatabaseConnection*

2.6.4 Submodules

2.6.5 src.db_drivers.utils module

class src.db_drivers.utils.**BaseDatabaseConfig**(*db_info: ~typing.Dict = <factory>*, *params: ~typing.Dict = <factory>*, *need_to_clear: bool = False*)

Базовые классы: object

Базовая конфигурация для прдключения к базе данных

Параметры

- **db_info** (*Dict*) – Название базы данных и таблицы, которой ножно прдключитьмя и выпрлнять в дальнейшем операции соответственно. Значение по умолчанию {„db“: „default_db“, „table“: „default_table“}.
- **params** (*bool*) – Набор дополнительных параметров, которые необходимы для подключения и настройки бд. Значения по умолчанию dict().
- **params** – Если True, то после успешного прдключения к базе данных содержимое указанной таблицы будет очищено. Значения по умолчанию False.

db_info: Dict

params: Dict

need_to_clear: bool = False

class src.db_drivers.utils.**AbstractDatabaseConnection**

Базовые классы: ABC

Интерфейс, который должен поддерживать класс взаимодействия с определённой базой данных

abstract open_connection() → *ReturnInfo*

Метод предназначен для подключения к бд.

Результат
Статус завершения операции с пояснительной информацией.

Тип результата
ReturnInfo

abstract is_open() → bool

Метод предназначен для проверки статуса подключения к бд.

Результат
Если True, то соединение с бд есть, иначе False.

Тип результата
bool

abstract close_connection() → *ReturnInfo*

Метод предназначен для разрыва соединения с бд.

Результат
Статус завершения операции с пояснительной информацией.

Тип результата
ReturnInfo

abstract create(*items: List[object]*) → *ReturnInfo*

Метод предназначен для добавления новых объектов в бд. Уникальность добавлемых объектов определяется по полю id. Если объект с таким id уже существует, то затирания информации не произойдёт: в бд останется прежний объект.

Параметры
items (*List[object]*) – Объекты на добавление.

Результат
Статус завершения операции с пояснительной информацией.

Тип результата
ReturnInfo

abstract read(*ids: List[str]*) → List[object]

Метод предназначен для получения объектов из бд по их идентификаторам. Если такого идентификатора не существует, то он будет пропущен.

Параметры
ids (*List[str]*) – Идентификаторы объектов, которые нужно получить.

Результат
Список запрошенных объектов.

Тип результата
List[object]

abstract update(*items: List[object]*) → *ReturnInfo*

abstract delete(*ids: List[str]*) → *ReturnInfo*

Метод предназначен для удаления объектов из бд по их идентификаторам. Если такого идентификатора не существует, то он будет пропущен.

Параметры
ids (*List[object]*) – Идентификаторы объектов, которые нужно удалить.

Результат
Статус завершения операции с пояснительной информацией.

Тип результата
ReturnInfo

abstract count_items() → object

Метод предназначен для получения количества элементов в таблице бд, к которой было выполнено подключение.

Результат
Структура данных, в которой хранится информация о количестве объектов.

Тип результата
object

abstract item_exist(*id: str*) → bool

Метод предназначен для проверки на наличие объекта в бд по его идентификатору.

Параметры
id (*str*) – Иднетификатор объекта.

Результат
Если объект существуюет, то True, иначе False.

Тип результата
bool

abstract clear() → *ReturnInfo*

Метод предназначен для удаления содержания таблицы в бд, к которой было выполнено подключение.

Результат
description

Тип результата
ReturnInfo

2.7 src.parsers package

2.7.1 Submodules

2.7.2 src.parsers.memory_extraction module

src.parsers.memory_extraction.**mem_custom_triplet_parse_func**(*raw_response: str*) → Tuple[List[Tuple[str, str, str]], *ReturnStatus*]

Функция предназначена для разбора результата генерации ответа LLM-агента, в рамаках задачи по извлечению триплетов из текста на естественном языке.

Параметры
raw_response (*str*) – Исходный ответ LLM-агента.

Результат
Разобранный список триплетов из ответа LLM-агента.

Тип результата
Tuple[List[Tuple[str, str, str]], *ReturnStatus*]

src.parsers.memory_extraction.**mem_custom_thesis_parse_func**(*raw_response: str*) → Tuple[List[Tuple[str, str]], *ReturnStatus*]

Функция предназначена для разбора результата генерации ответа LLM-агента, в рамаках задачи по извлечению тезисной информации из текста на естественном языке.

Параметры
raw_response (*str*) – Исходный ответ LLM-агента.

Результат
Разобранный список „тезисных“ триплетов из ответа LLM-агента.

Тип результата
Tuple[List[Tuple[str, str]], *ReturnStatus*]

2.7.3 src.parsers.query_parser module

src.parsers.query_parser.**qa_custom_entities_parse_func**(*raw_response: str*) → Tuple[List[str], *ReturnStatus*]

Функция предназначена для разбора результата генерации ответа LLM-агента, в рамаках задачи по извлечению ключевых сущностей из текста на естественном языке.

Параметры
raw_response (*str*) – Исходный ответ LLM-агента.

Результат
Разобранный список ключевых сущностей из ответа LLM-агента.

Тип результата
Tuple[List[str], *ReturnStatus*]

2.7.4 src.parsers.question_answering module

src.parsers.question_answering.**qa_custom_answer_parse_func_en**(*raw_response: str*) → Tuple[str, *ReturnStatus*]

Функция предназначена для разбора результата генерации ответа LLM-агента, в рамаках условной QA-задачи на английском языке.

Параметры
raw_response (*str*) – Исходный ответ LLM-агента.

Результат
Разобранный ответ на user-вопрос от LLM-агента.

Тип результата
Tuple[str, *ReturnStatus*]

src.parsers.question_answering.**qa_custom_answer_parse_func_ru**(*raw_response: str*) → Tuple[str, *ReturnStatus*]

Функция предназначена для разбора результата генерации ответа LLM-агента, в рамаках условной QA-задачи на русском языке.

Параметры
raw_response (*str*) – Исходный ответ LLM-агента.

Результат
Разобранный ответ на user-вопрос от LLM-агента.

Тип результата
Tuple[str, *ReturnStatus*]

2.8 src.utils package

2.8.1 Submodules

2.8.2 src.utils.data_structs module

class src.utils.data_structs.**NodeType**(*value*)

Базовые классы: Enum

Доступные типы вершин.

object = 'object'
Вершина хранит атомарную сущность.

hyper = 'hyper'
Вершина хранит тезисную информацию.

episodic = 'episodic'
Вершина хранит эпизодическую информацию.

class src.utils.data_structs.**RelationType**(*value*)

Базовые классы: Enum

Доступные типы связей/триплетов.

simple = 'simple'

Связывает только вершины с типом „object“.

hyper = 'hyper'

Связывает пары вершин („object“,“hyper“).

episodic = 'episodic'

Связывает пары вершин („object“, „episodic“) и („object“, „hyper“).

class src.utils.data_structs.**Node**(*name: str*, *type: ~src.utils.data_structs.NodeType*, *prop: dict = <factory>*, *stringified: str | None = None*, *id: str | None = None*)

Базовые классы: object

Структура данных вершины.

name: str

Главная смысловая информация.

type: *NodeType*

Тип вершины.

prop: dict

Дополнительные свойства вершины.

stringified: str = None

Строковое представление вершины.

id: str = None

Идентификатор вершины, полученный на основе её строкового представления.

class src.utils.data_structs.**Relation**(*name: str*, *type: ~src.utils.data_structs.RelationType*, *prop: dict = <factory>*, *id: str | None = None*)

Базовые классы: object

Струкура данных связи.

name: str

Главная смысловая информация.

type: *RelationType*

prop: dict

Дополнительные свойства связи.

id: str = None

Идентификатор связи, полученный на основе строкового представления триплета, в котором она находится. Отличается от значения в поле id объекта класса Triplet.

class src.utils.data_structs.**Triplet**(*start_node: Node*, *relation: Relation*, *end_node: Node*, *stringified: str | None = None*, *id: str | None = None*)

Базовые классы: object

Структура данных триплета.

start_node: *Node*

relation: *Relation*

end_node: *Node*

stringified: str = None

Строковое представление триплета.

id: str = None

Идентификатор триплета, полученный на основе его строкового представления. Отличается от значения в поле id объекта класса Relation.

class src.utils.data_structs.**BaseCreator**

Базовые классы: object

static add_str_props(*obj:* Relation | Node, *obj_str: str*) → str

Метод предназначен для добавления свойств, хранящихся в структуре связи/вершины, к их базовым стрококвым представлениям.

Параметры

- **obj** (*Union[Relation, Node]*) – Структура объекта, строковое представление которого обогащается свойствами.
- **obj_str** (*str*) – Текущее стрококове представление объекта.

Результат
Обогащённое строковое представление.

Тип результата
str

class src.utils.data_structs.**NodeCreator**

Базовые классы: *BaseCreator*

static create(*add_stringified_node: bool = True*, ***kwargs*) → *Node*

Метод предназначен для создания структуры данных вершины с указанным содержанием.

Параметры

- **add_stringified_node** (*bool, optional*) – Если True, то в структуру данных вершины будет сохранено её строковое представление, иначе False, Значение по усолчанию True.
- **kwargs** – Значения полей структуры данных Node. Если они не будут указаны, то будут использованы значения по-умолчанию.

Результат
Созданная структура данных вершины.

Тип результата
Node

static stringify(*node:* Node) → str

Метод предназначен для приведения структуры данных вершины в её строковое представление.

Параметры
triplet (Node) – Структура данных вершины.

Результат
Строковое представление вершины.

Тип результата
str

src.utils.data_structs.**create_id_for_node_pair**(*node1_id: str*, *node2_id: str*) → str

Метод предназначен для условной генерации идентификатора к паре вершин. Вершины представлены в виде их собственных идентификаторов. При указании такой же пары вершин, но в другом порядке, полученный идентифкатор не изменится: инвариант относительно перестановок.

Параметры

- **node1_id** (*str*) – Идентификатор первой вершины.
- **node2_id** (*str*) – Идентификатор второй вершины.

Результат
Идентификатор пары вершин.

Тип результата
str

src.utils.data_structs.**create_id**(*seed: str*) → str

class src.utils.data_structs.**TripletCreator**

Базовые классы: *BaseCreator*

static create(*start_node:* Node, *relation:* Relation, *end_node:* Node, *add_stringified_triplet: bool = True*, *t_id: str | None = None*) → *Triplet*

Метод предназначен для создания структуры данных триплета с указанным содержанием. Триплет является ориентированным: у связи между вершинами (парой объект/субъект) есть направление.

Параметры

- **start_node** (Node) – Структура данных старотовой вершины триплета
- **relation** (Relation) – Структура данных связи триплета.
- **end_node** (Node) – Структура данных конеченой вершины триплета.
- **add_stringified_triplet** (*bool, optional*) – Если True, то в структуру данных триплета будет сохранено его строковое представление, иначе False, Значение по умолчанию True.
- **t_id** (*str, optional*) – Идентификатор триплета, который будет назначен вручную, Если идентификатор не указан, то он будет автоматически сгенерирован. Значение по умолчанию None.

Результат
Созданная структура данных триплета.

Тип результата
Triplet

static stringify(*triplet:* Triplet) → str

Метод предназначен для приведения структуры данных триплета в его строковое представление. Строковое представление зависит от типа триплета: (1) simple - используется информация из обоих вершин и связи; (2) hyper/episodic - используется информация только из конечной вершины.

Параметры
triplet (Triplet) – Структура данных триплета.

Исключение
KeyError – В триплете указа связь с типом, который не поддерживается.

Результат
Стрококвое представление триплета.

Тип результата
str

class src.utils.data_structs.**QueryInfo**(*query: str*, *entities: List[str] | None = None*, *linked_nodes: List[object] | None = None*, *linked_nodes_by_entities: List[object] | None = None*)

Базовые классы: object

Класс предназначен для хранения промежуточных результатов по user-вопросу, который обрабатывается в рамках QA-конвейера.

Параметры

- **query** (*str*) – Исходный user-вопрос.
- **entities** (*List[str]*) – Набор сущностей, который был извлечён из user-вопроса. Значение по умолчанию None.
- **linked_nodes** (*List[object]*) – Набор объектов (вершин) из памяти (графа знаний) ассистента, который был сопоставлен сущностям из user-вопроса. Значение по умолчанию None.
- **linked_nodes_by_entities** (*List[object]*) – Значение по умолчанию None.

query: str

entities: List[str] = None

linked_nodes: List[object] = None

linked_nodes_by_entities: List[object] = None

2.8.3 src.utils.errors module

class src.utils.errors.**ReturnStatus**(*value*)

Базовые классы: Enum

An enumeration.

success = 0

warning = 1

error = 2

bad_format = 3
QA_BAD_QA_PROMPT_MSG, QA_BAD_ENTITIES_EXTRACTION_PROMPT_MSG, MEM_BAD_THESIS_EXTRACTION_PROMPT_MSG, MEM_BAD_TRIPLET_EXTRACTION_PROMPT_MSG

zero_triplets = 4
MEM_ZERO_EXTRACTED_TRIPLETS_MSG

zero_entities = 5
QA_ZERO_ENTITIES_MSG

zero_linked_nodes = 6
QA_ZERO_LINKED_NODES_MSG

zero_retrieved_triplets = 7
QA_ZERO_RETRIEVED_TRIPLETS_MSG

empty_answer = 8
QA_EMPTY_ANSWER_MSG

not_supported_lang = 9
NOT_SUPPORTED_LANG_MSG

empty_input_text = 10
NOT_SUPPORTED_LANG_MSG

unknown_lang = 11
NOT_SUPPORTED_LANG_MSG

class src.utils.errors.**ReturnInfo**(*occurred_warning: ~typing.List[~src.utils.errors.ReturnStatus] = <factory>*, *status: ~src.utils.errors.ReturnStatus = ReturnStatus.success*, *message: str = ''*)

Базовые классы: object

Класс предназначен для хранения пояснительной информации к полученному результату в рамках некоторой операции.

Параметры

- **occurred_warning** (*List[ReturnStatus]*) – Предупреждения, которые возникли в процессе выполнения операции.
- **status** (ReturnStatus) – Статус завершения операции.
- **message** (*str*) – Пояснительное сообщение к статусу возврата.

occurred_warning: List[*ReturnStatus*]

status: *ReturnStatus* = 0

message: str = ''

2.8.4 src.utils.language_detector module

src.utils.language_detector.**detect_lang**(*text: str*) → Tuple[str, *ReturnStatus*]

Функция предназначена для определения доминирующего языка, который используется во входном тексте.

Параметры
text (*str*) – Текст, для которого требуется определить язык.

Результат
Кортеж из двух объектов: (1) язык в краткой нотации; (2) статус завершения операции с пояснительной информацией.

Тип результата
Tuple[str, *ReturnStatus*]

2.8.5 src.utils.logger module

src.utils.logger.**init_logger**(*args*, *stdout_only=False*)

class src.utils.logger.**Logger**(*path*)

Базовые классы: object

to_json(*obj*, *filename='history.json'*)

ГЛАВА 3

Indices and tables

- genindex
- modindex
- search

Содержание модулей Python

S

Алфавитный указатель

D

E

M

N

O

P

Q

R

S

T

U

V

W

Z