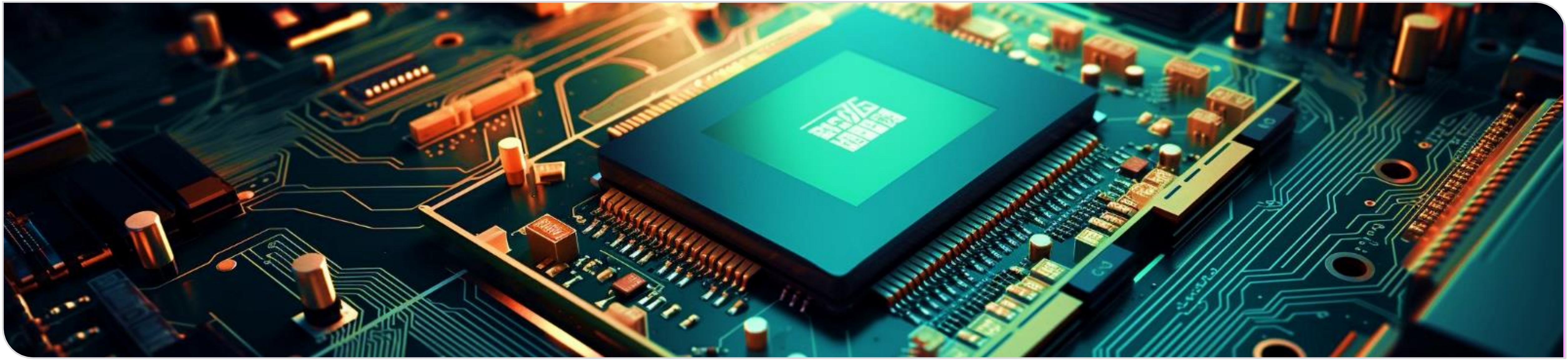
Master Thesis – Overview:

Leveraging Embeddings and Knowledge Graphs for Enhanced Scholarly Information Retrieval: A Comparative Analysis of Retrieval Approaches Using Large Language Models

by Marco Schneider

supervised by Angelika Kaplan



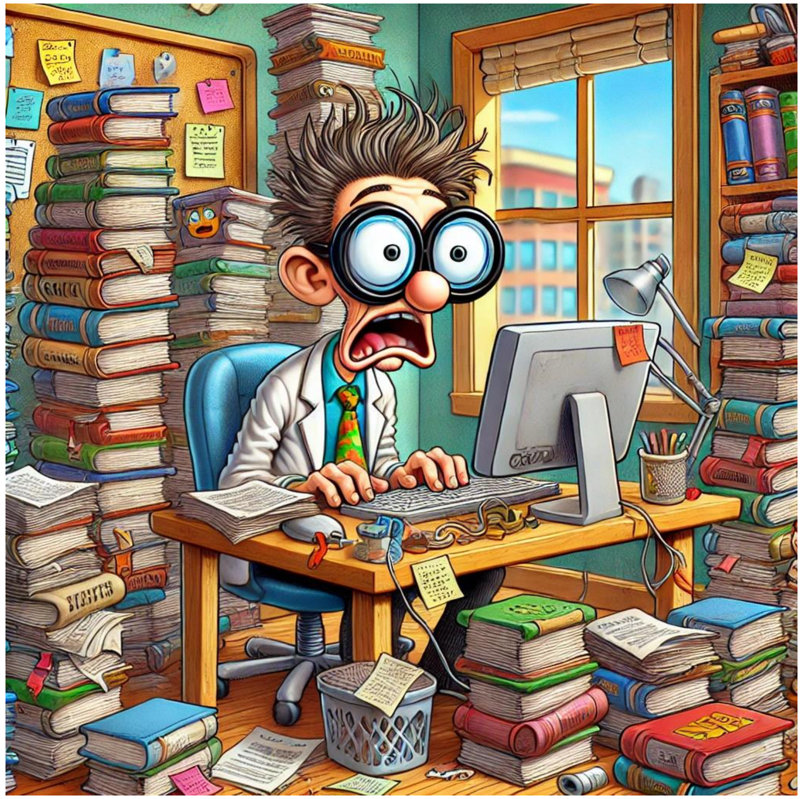
Overview for the Architecture Review

- Thank you for participating in the Code Review
- The following pages introduce you to the topic of the master thesis and explain what we intend to create
- We also give you an overview over the project's files and a small usage guide to try out things
- **Note:** In preparation of the code review, I am still in the process of refactoring parts of the repository.

Content

Literature Research is hard

Who doesn't know this situation?



OpenAI. (2024). Cartoon-style image of a frustrated academic researcher. Created with DALL-E. Retrieved on July 4, 2024

In modern academic settings, most research results are published in scholarly **digital articles**, presenting difficulties for human and automated processing [Jaradeh19].

Academic search engines return a **set of ranked documents** and users have the tedious task of finding relevant information from it [Thambiand22].

Future generation academic search engines should **focus on understanding meaning** rather than simply matching keywords [Hippel23].

Literature Research could be easy

Large Language Models (LLMs) demonstrate **remarkable abilities** in natural language tasks [Yangetal24]

The application of LLMs to academic search has the potential to **reduce barriers** to accessing information and **speed up** research tasks

Just research by chatting!

OpenAI. (2024). Cartoon-style image of a happy academic researcher. Created with DALL-E. Retrieved on July 4, 2024

LLMs alone are not sufficient

OpenAI. (2024). Cartoon-style image of a hallucinating ChatGPT. Created with DALL-E. Retrieved on July 4, 2024

LLM is trained and its knowledge is **frozen** in time

Especially in knowledge specific tasks LLMs tend to **hallucinate** where they contradict existing sources or lack supporting evidence [Yangetal24]

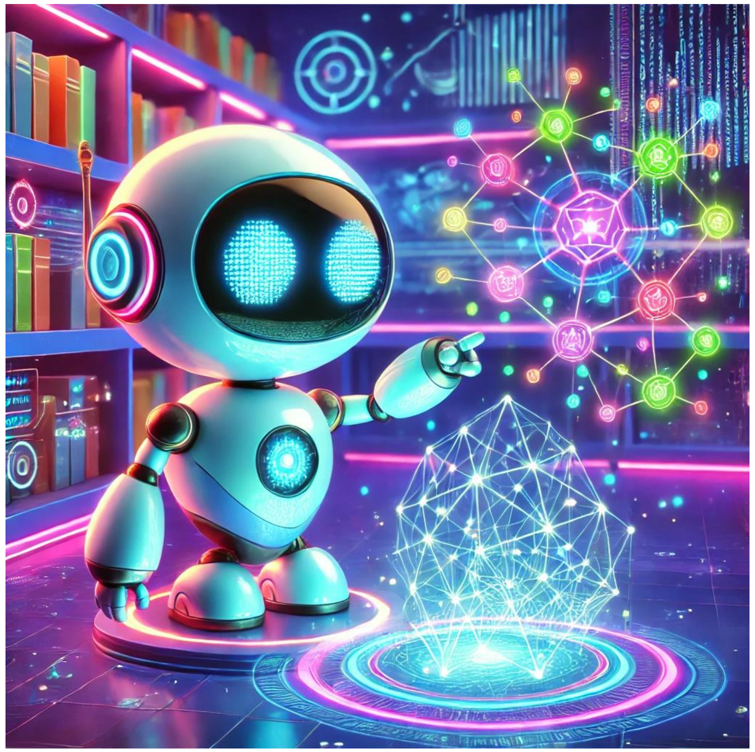
Retrieval Augmented Generation (RAG) allows to **integrate external knowledge** to retrieve current knowledge and reduce hallucinations [Lewis20]

Knowledge Graphs (KG) for Structured Information

The ORKG already offers a way to relate scientific papers in a structured manner in a graph format. [Jaradeh19]

Applying LLM-based Retrieval on such KGs recently sparked major interest in the research community.

It is currently unknown how well a LLM-based Retrieval on the ORKG performs and what drawbacks these might have.



OpenAI. (2024). Cartoon-style image of a bot reading from a graph. Created with DALL-E. Retrieved on November 20, 2024

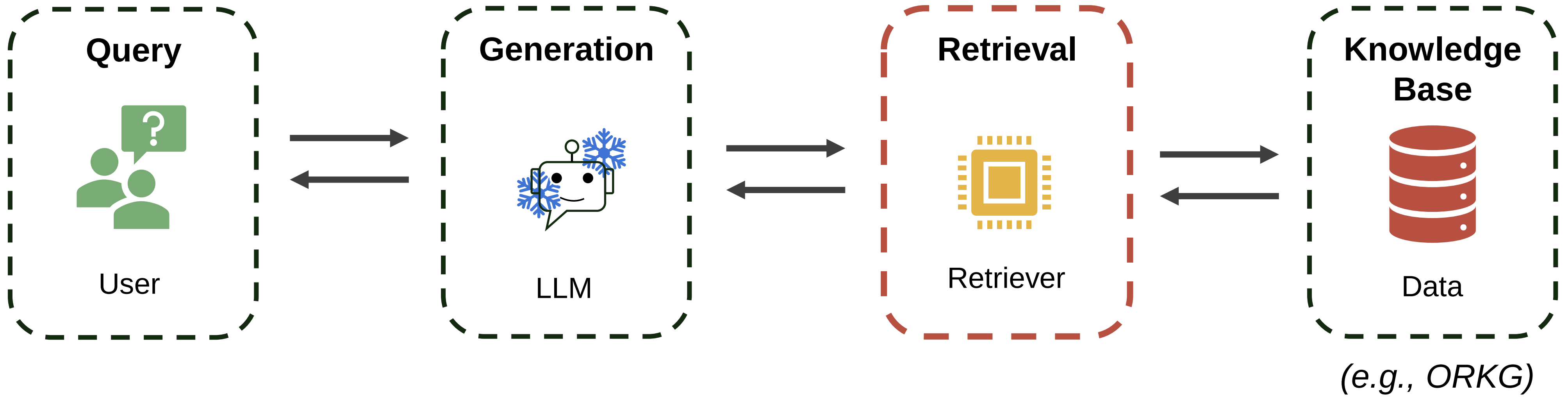
Content

Fundamentals: Retrieval Augmented Generation (RAG)

What is RAG?

- RAG aims to **retrieve relevant data from an external knowledge base** to give the LLM recent and factual data for answer generation

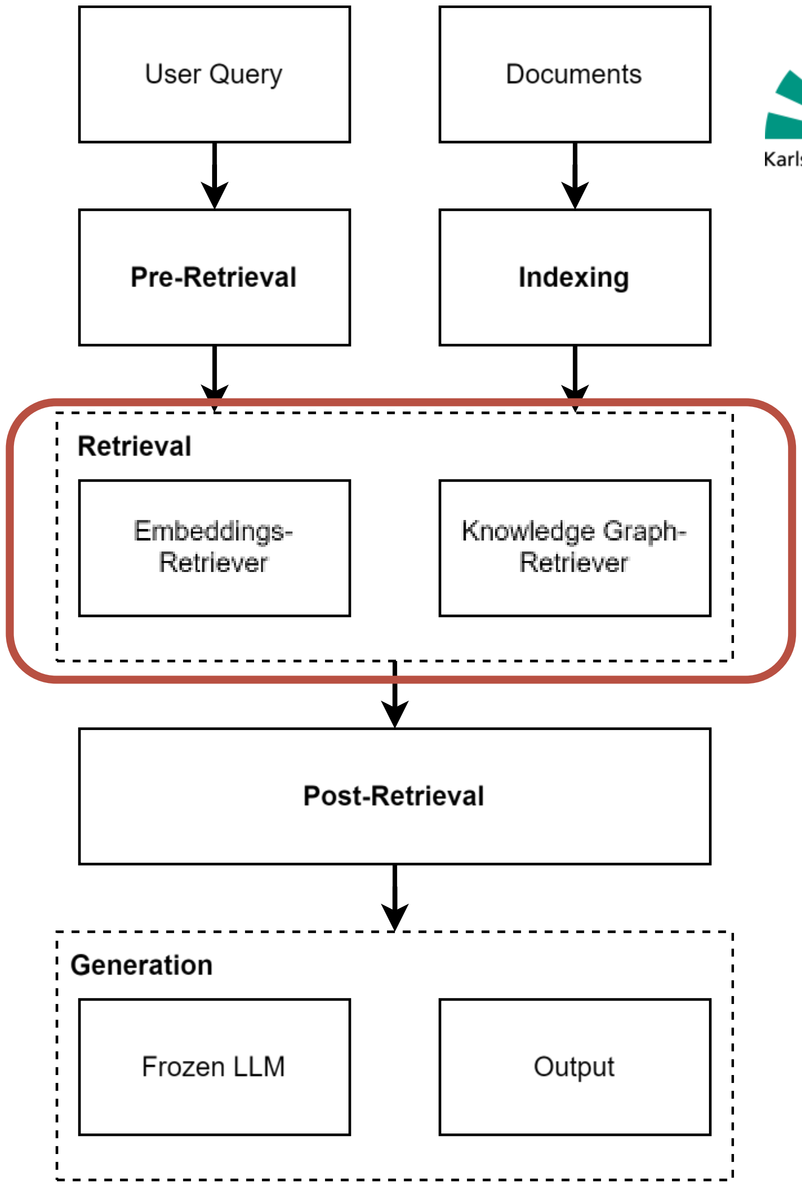
Overview RAG Process



Fundamentals: Deeper look

- The RAG approach basically has 4 main phases

1. It starts with the **indexing** where data is saved in an appropriate format in a knowledge base (KB)
2. **Pre-Retrieval** are processing steps applied to a user query (question) to prepare the question for the retriever
3. **Retrieval** here a retriever object receives the query and retrieves the most relevant documents from the KB. This can be either embedding-based or knowledge graph-based depending on the KB used
4. **Generation** here a language model (like GPT) is tasked to generate an answer based on the documents retrieved

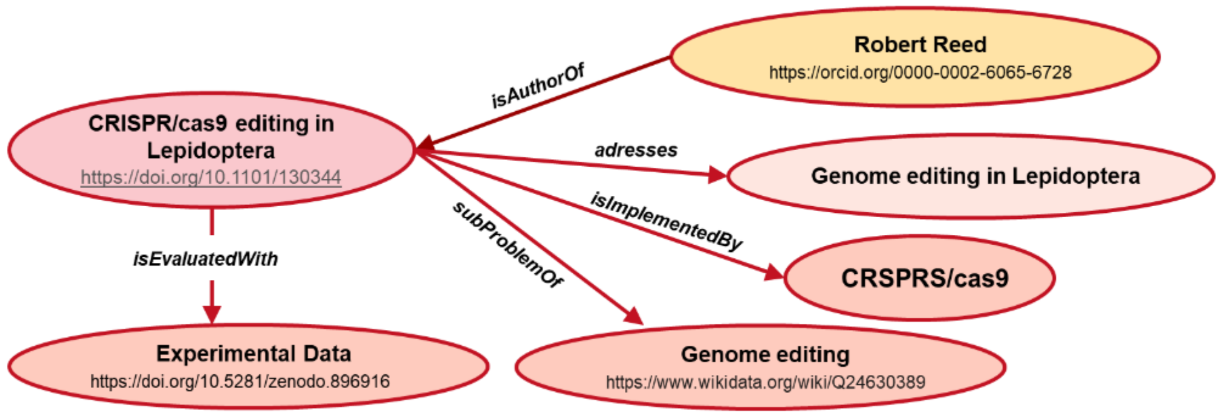


Fundamentals: Open Research Knowledge Graph (ORKG)

- Leverages **knowledge graphs** to represent scholarly literature semantically.
- Papers are added in a community effort by manually adding them based on **templates**

Templates

- Are **predefined structures** that guide users in adding a new paper to the ORKG graph
- Help **standardize** the representation of various types of research contributions
- **Domain specific templates** are needed to ensure that the unique requirements and characteristics of different fields of study are captured.


Robert Reed
https://orcid.org/0000-0002-6065-6728
isAuthorOf
CRISPR/cas9 editing in Lepidoptera
https://doi.org/10.1101/130344
adresses
Genome editing in Lepidoptera
isImplementedBy
CRSPRS/cas9
subProblemOf
isEvaluatedWith
Experimental Data
https://doi.org/10.5281/zenodo.896916
Genome editing
https://www.wikidata.org/wiki/Q24630389

Example graph representation [Jaradeh19]

Content

Overview:
What we need to code

Main Goal 1

Investigate how Large Language Models (LLMs) can be used to guide the retrieval on Knowledge Graphs in a Retrieval Augmented Generation (RAG) system to improve quality and reliability in a question-answering setting for software architecture research.

Main Goal 2

Develop and implement a improved Knowledge Graph Language Model Question Answering (KGLM-QA) Retriever. And compare it against existing KGLM-QA retrievers.

- The main thing we had to implement was a way to **compare** different Retrievers which each other to see which one has the best performance on numerous categories like completeness, scalability, runtime, …
- Secondly, it should be “straightforward” to later include new retrievers into the system to allow a comparison between other retrievers. Therefore, we need to provide unified interfaces that can be used by retrievers.
- Furthermore, we have **two types** of retrievers: 1) Knowledge Graph based and 2) Embedding-based retrievers. The second type are retrievers where no graph is used but the papers are filled into a vector store with a similarity based retrieval (which is the current State-of-Art)

The intended use of our Implemented System

- The system's intended use is split into three Roles:
  1. **User Role:** The system is a prototype for a user interaction. It implements a QA-Pipeline where users can ask natural language questions on a Paper Knowledge Base.
  2. **Developer Role:** The system provides interfaces and configurations that makes it easy for developers to extend the provided Knowledge Bases and Retrievers of the system.
  3. **Master Thesis:** The system allows to experiment on implemented Retrievers and Knowledge Bases to infer the best performance for conclusions in the thesis.

User Role

1. Ask questions on a scholarly dataset
2. Find relevant literature faster

Developer Role

1. Tool for the implementation of a RAG process
2. Benchmarking of retrievers

Master Thesis

Tool for a consistent and empirical evaluation of embedding and KG-based retrieval approaches

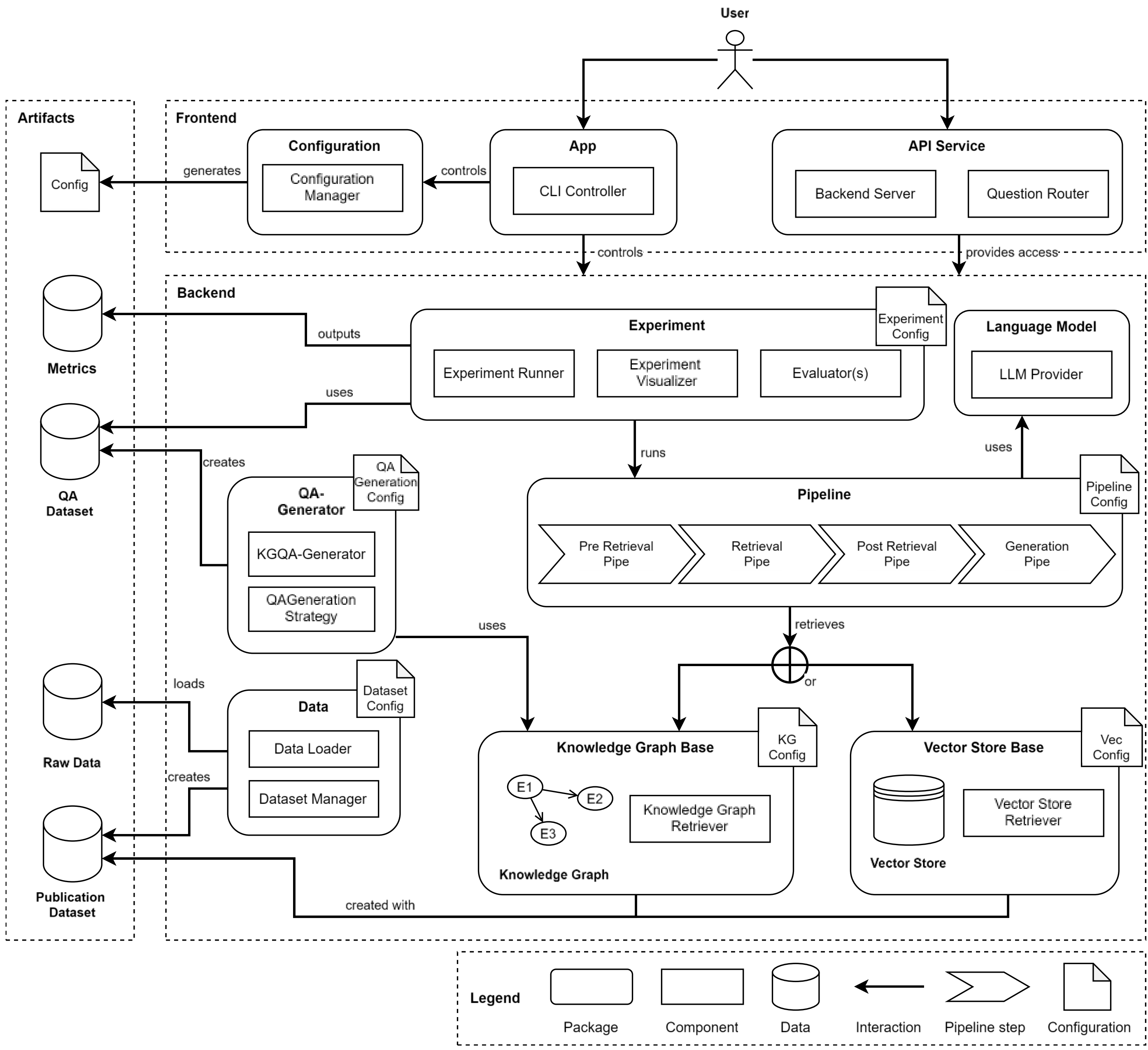
Content

Preparation

- In the following slides we will be presenting an **overview** of the components implemented in our system
- We also provide several **links** to the repository which you can use to get comfortable with the code
- One additional information before we begin. There is one directory pattern that is often used throughout the repository which we will explain here. A directory often consist of the following subfolders to give it structure:
  - **Interfaces**: Includes the base/interface implementations of a class
  - **Implementations:** Includes the implementations of one of the interfaces from the folder above
  - **Factory:** If applicable, contains factories that create classes from the implementations folder

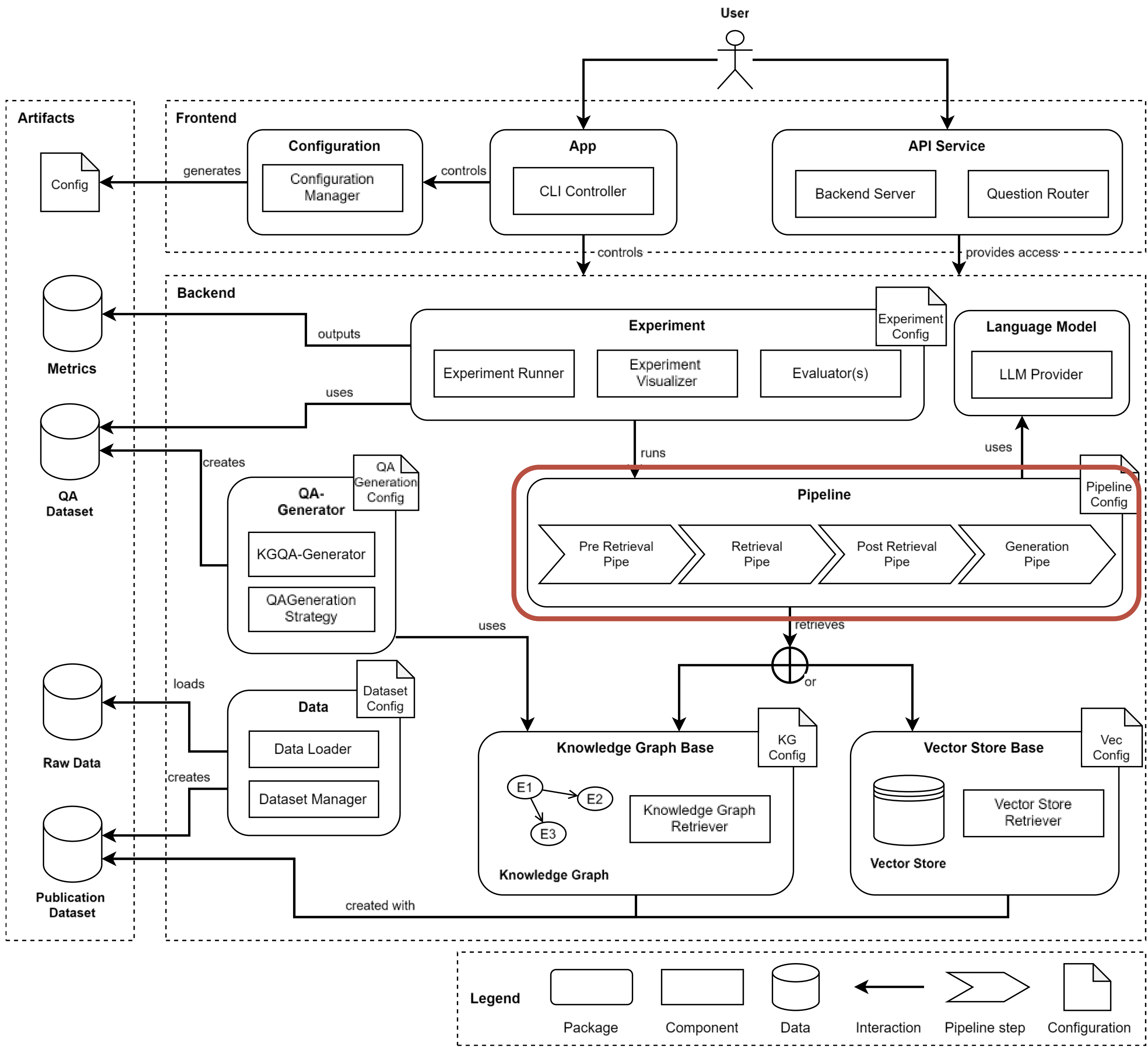
Overview of the SQA- System

- From now on we call the system that we implemented the Scholarly Question Answering (SQA) – System..
- The illustration on the right shows an overview of each component of the implemented SQA-System.



Overview of the SQA- System

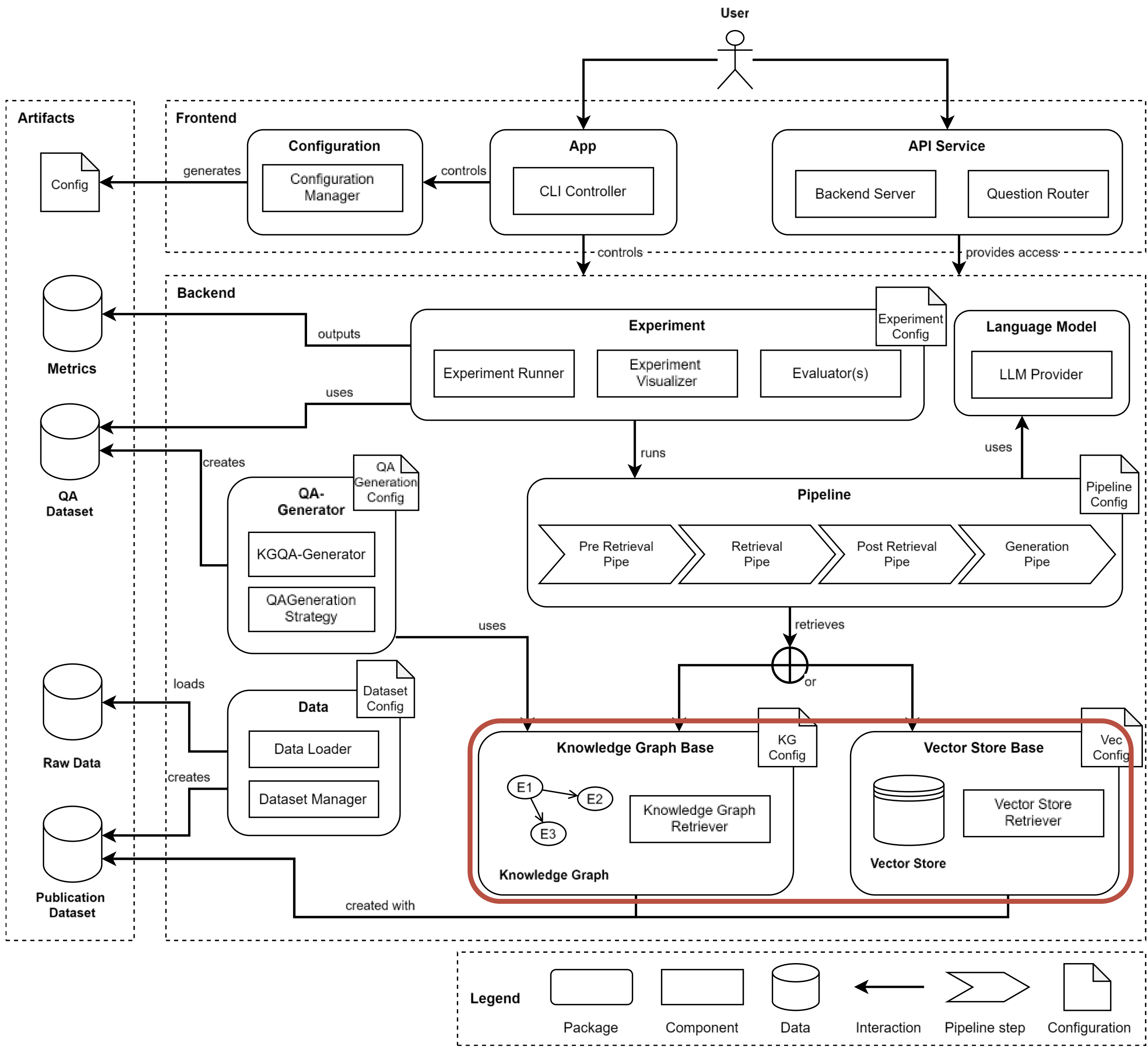
- The systems main component is the Pipeline
- It is implemented using the LangChain library to chain multiple pipes
- The pipeline is a chain of pipes that are used to process the data. The pipeline is implemented using the RetrievalPipeline class.
- The pipeline is generating a initial PipeIOData model that is then passed through the chain of pipes.
- A pipeline is created by the PipelineFactory based on the PipelineConfig.
- There are four types of pipes implemented
  - PreRetrievalPipe
  - RetrievalPipe
  - PostRetrievalPipe
  - GenerationPipe



Overview of the SQA- System

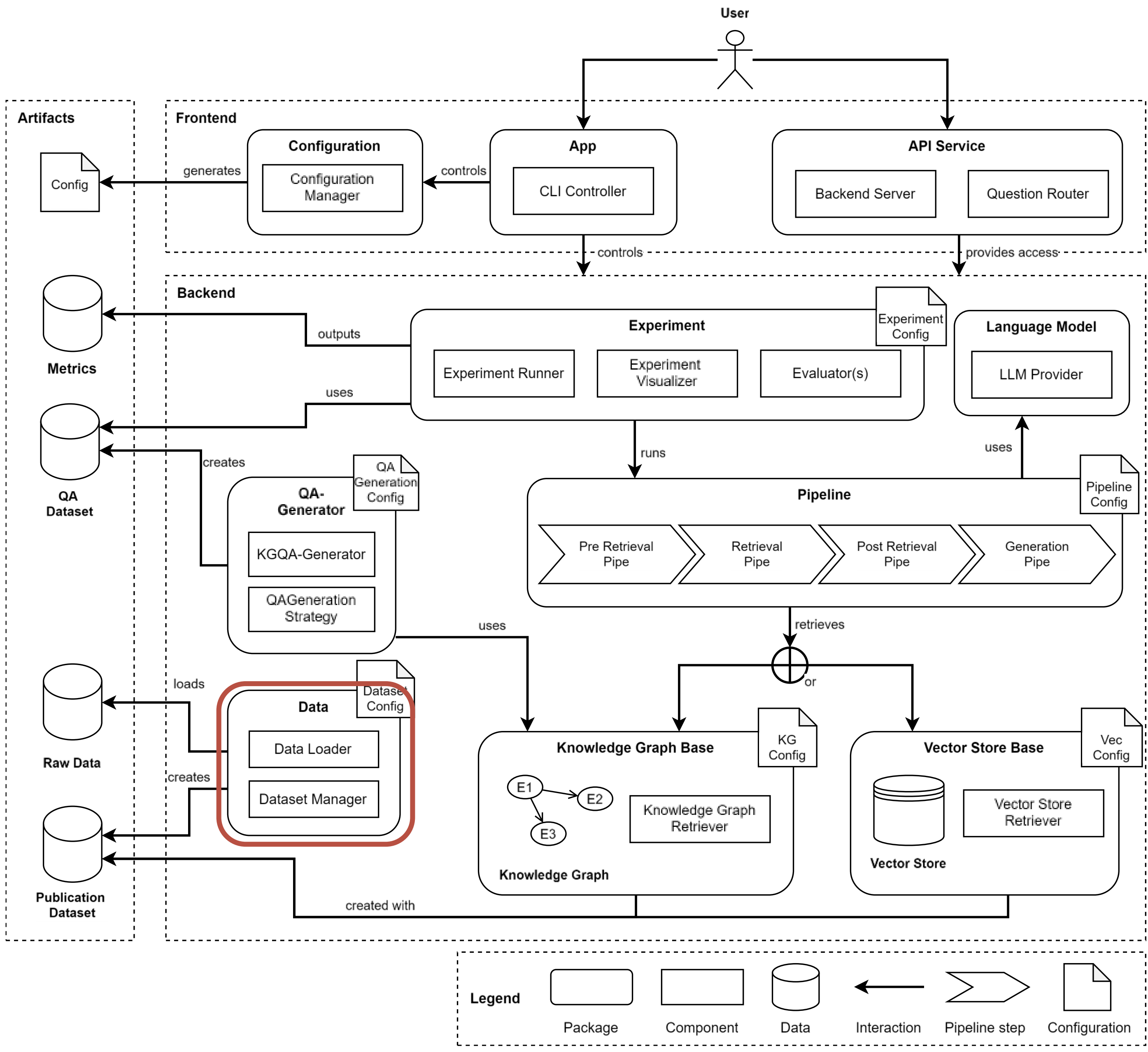
- One pipe used is the RetrievalPipe which uses either a KnowledgeGraphRetriever or a VectorStoreAdapter to retrieve contexts from a knowledge base.
- The KG retrievers are the focus point of the master thesis.
- KG retrievers were implemented from existing retrievers from the literature by adapting their code to our interfaces.
- We also implemented our own retriever which we will talk about later

- Each KnowledgeGraphRetriever retrieves facts from a KnowledgeGraph using a LLM based retrieval algorithm. On the other hand, the VectorStoreAdapter applies similarity metrics on its stored data vectors to find context relevant to a given question.



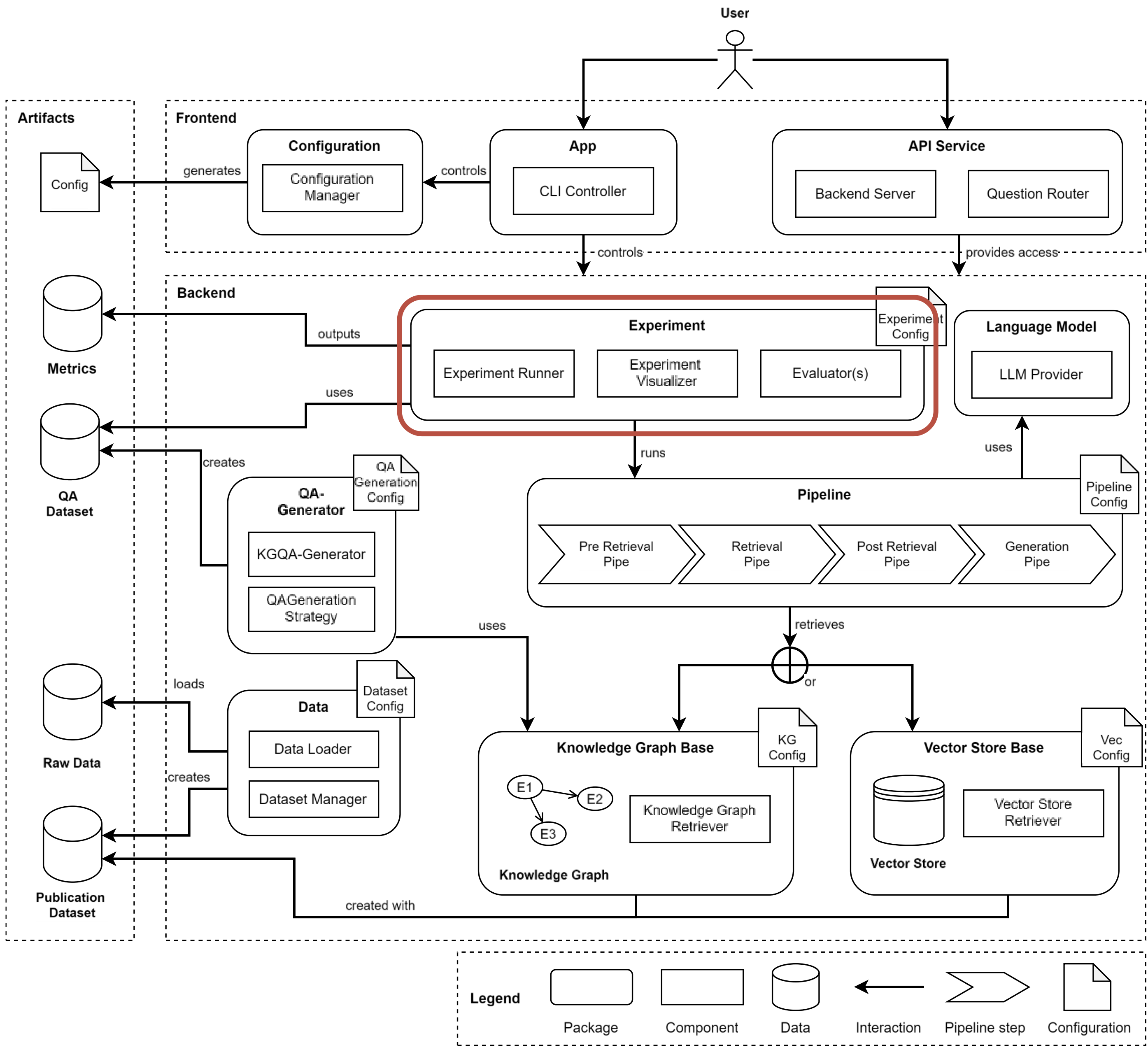
Overview of the SQA- System

- The data used to create both knowledge bases (Knowledge Graph and Vector Store) comes from the data component
- Its DatasetManager loads and processes raw data and provides it to the knowledge bases in form of a PublicationDataset which contains the metadata and full text of scientific papers.
- To get the exact structured form that we need for the Knowledge Graph construction from the scientific papers, we developed a full text extraction approach that utilizes a LLM. The code for this approach can be found in the SummaryExtractor class.



Overview of the SQA- System

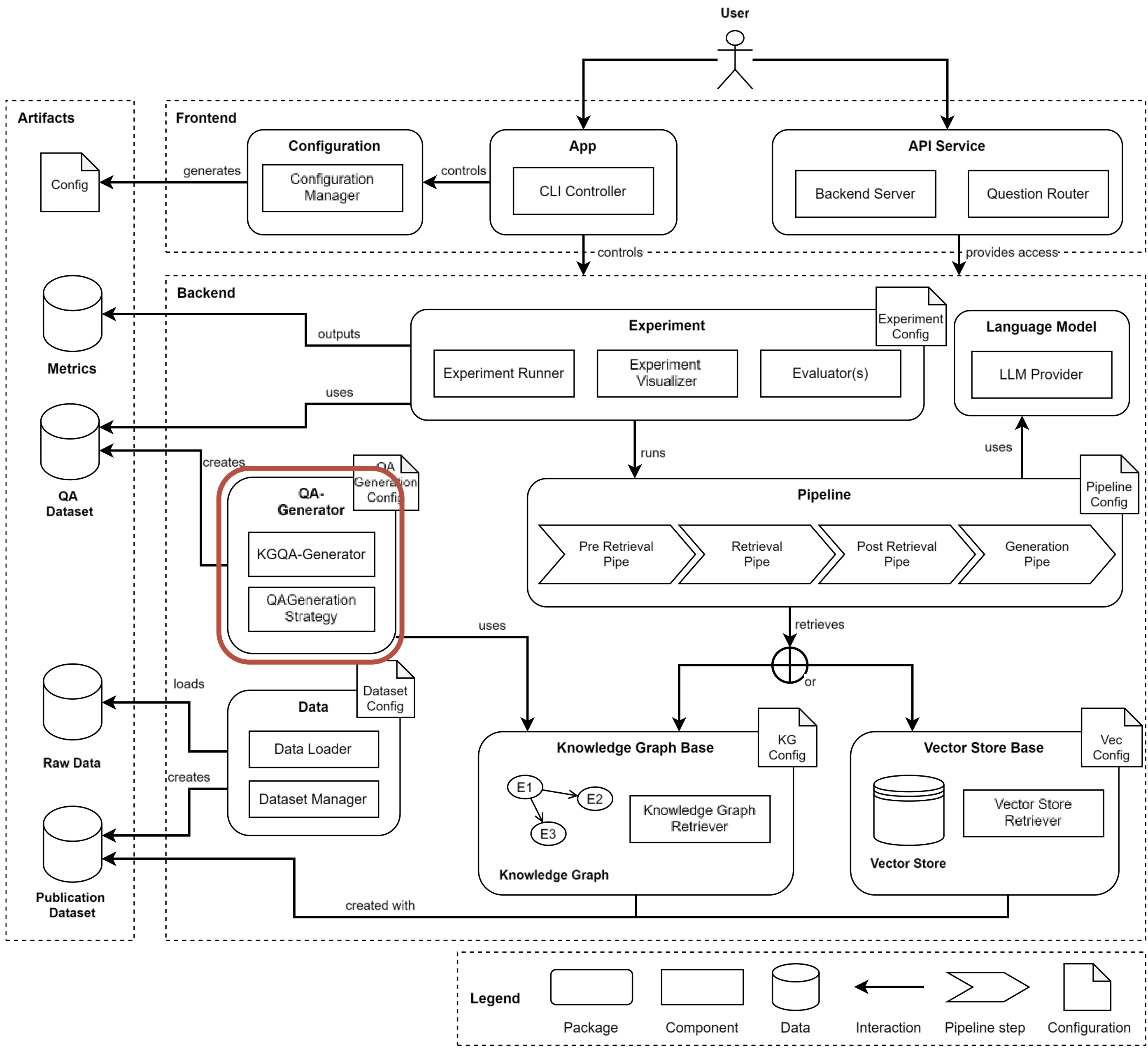
- To evaluate the performance of the retrievers, the ExperimentRunner class is responsible
- It generates Pipelines based on the given configuration which also allows for hyperparameter tuning and applies questions from a QADataset to it.
- The retrieved answers are then evaluated by Evalutors which run metrics on the retrieved context



Overview of the SQA- System

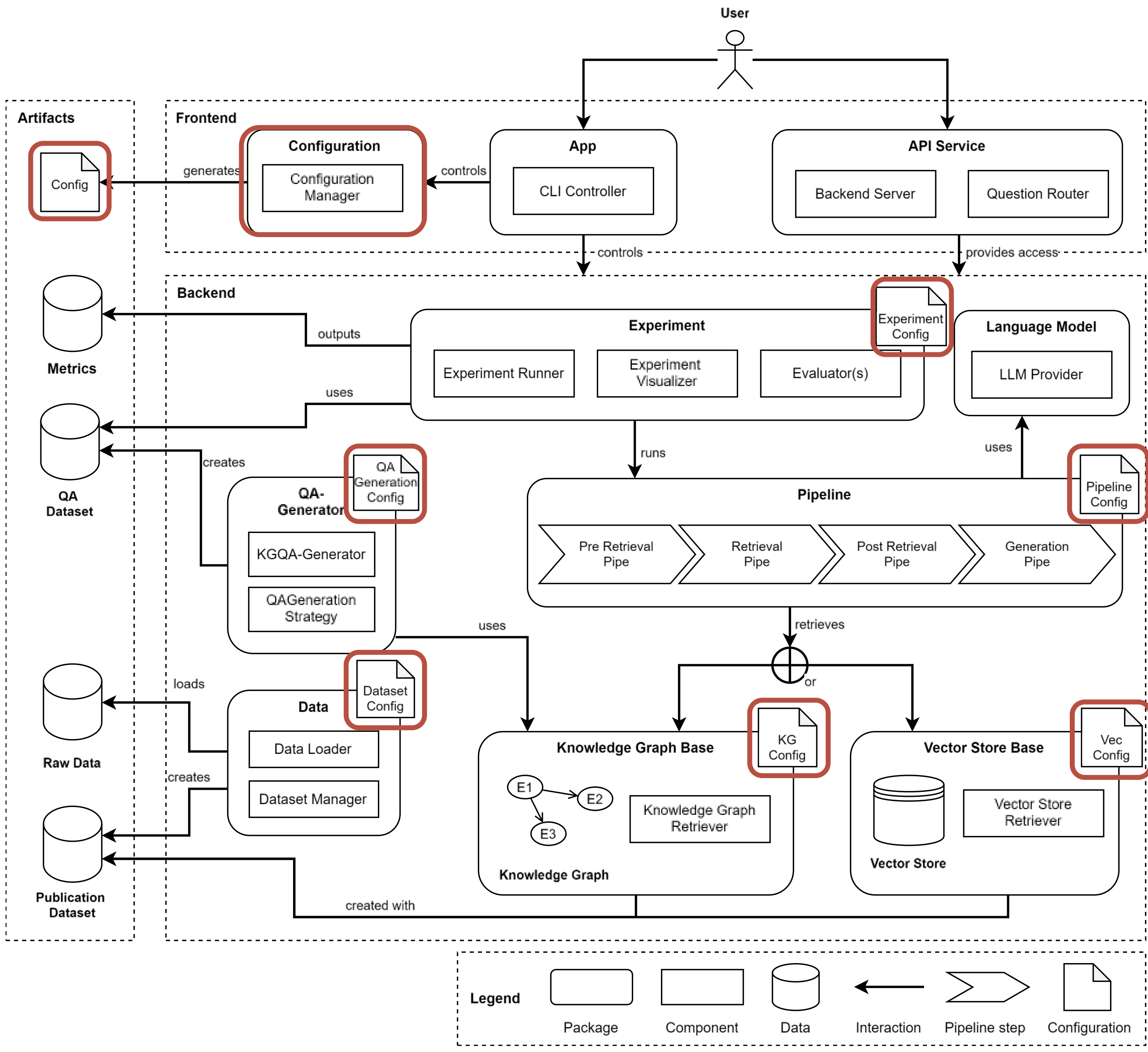
- The QADataset(s) used for the experiments, is created using a QAGenerator which utilizes a LLM to semi-automatically create QAPairs.

- The creation of the pairs depends on the QAGenerationStrategy used. We have two types of strategies.
  - A Single-Fact strategy that generates questions that only need a single triple from the graph to give answer to the question.
  - A Multi-Fact strategy that generates questions where multiple triples are needed.



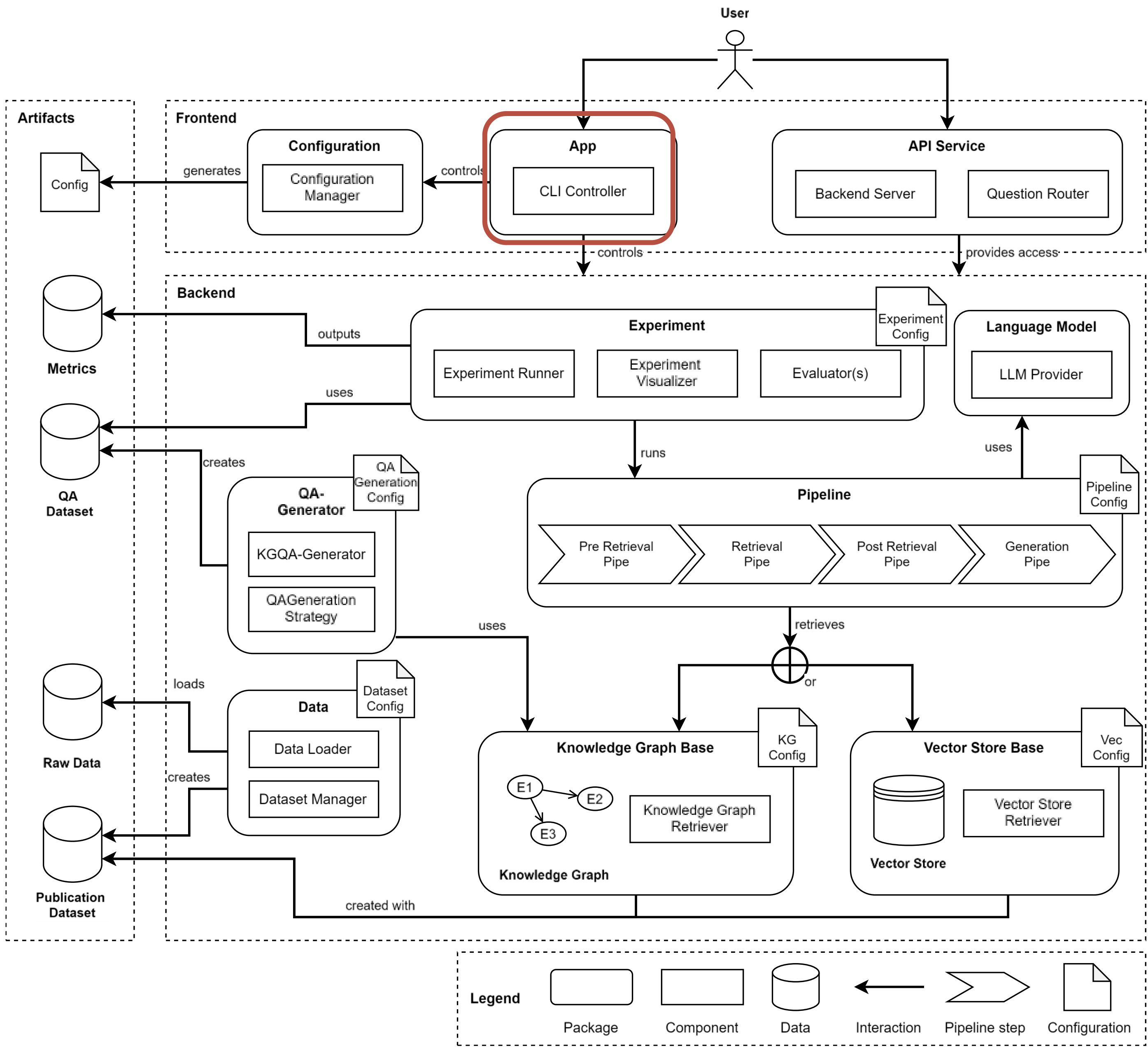
Overview of the SQA- System

- A major concept applied in the SQA-System are its configurations.
- The system is built from ground up to allow each component to be highly configurable
- Each component has one configuration file that is serialized in the data/configs folder to allow to reproduce experiments.
  - Take a look at the DefaultExperiments config json file. It recursively includes configurations from all over the system to accurately define how each component behaves.
- For each config there exists a corresponding ConfigurationManager that manages the serialization and distribution. All configs are stored in the configs folder.



Overview of the SQA- System

- The SQA-System also provides a CLI. It has the following features
  1. Create and manipulate configuration files for each component
  2. Run Experiments
  3. Run a Pipeline in interaction mode
  4. Run the QAGeneration



More Resources

- This should have given you a introduction to the system

- More resources to navigate the repository can be found in the readme file. There you can find a list of the repositories folders with detailed descriptions.

Evaluation

- In the following we will explain how the evaluation of the experiments is conducted.
- **What we want:** We want to compare the performance of the different KGQA Retrievers to assess a) how good the retrievers perform on our use case and b) how well our new retriever compares to the existing ones.
- **QA Dataset:** To achieve this comparison, we need a dataset of Question-Answer pairs. Each question is then queried and the retrievers are tasked to find an answer based on the data in the Knowledge Graph. To find out which retriever performs the best, we assess the answers that are returned

Evaluation using Evaluators

- Evaluating the answers is not as straightforward because the results are in natural language meaning that if the
  - **Expected answer** is “The paper XY states that Prompt engineering is the practice of crafting and refining prompts to achieve specific outcomes when interacting with AI models.”
  - **Generated answer** is “According to the paper XY, prompt engineering refers to the process of carefully designing and fine-tuning input instructions to guide AI systems toward producing targeted and specific results during interactions.”

- From the answers above we can see that while they contain the same information, the sentence structure is very much different. This makes it difficult to evaluate the performance because the evaluation metric has to consider the context of the sentences

- Because of this reason we are using “LLMs as a Judge” which is a technique that prompts an LLM to check whether the contexts of two answers are similar

Evaluation using Evaluators

- A popular open source Framework that provides multiple “LLM as a Judge” metrics is RAGAS.

- We have currently already implemented some of the provided metrics and intend to expand on this set. Current Metrics include:
  - **Faithfulness:** checks whether the answer was derived from the LLMs internal knowledge (or hallucination) or if the answer is actually based on the contexts
  - **Context Precision with reference:** Whether the retrieved contexts are useful to arrive at the golden answer
  - **Context Precision without reference:** Whether the retrieved contexts are useful to arrive at the generated answer
  - **LLM Context Recall:** Breaks down the golden_answer into “claims”. Each claim is then analyzed to determine whether the contexts include the claims. If all claims are in the context, the context recall should be maximal.

Evaluation using Evaluators

- Furthermore we also track the following metrics
  - **Carbon Emissions:** We track the emissions of the pipeline using CodeCarbon.io in Co2. This also tracks the hardware that has been used as well as related data such as power usage and consumed energy.
  - **Runtime:** We track the amount of time it takes for each retrieval
  - **Total Tokens:** We track the amount of tokens and costs for each retrieval

Evaluation using Evaluators

- Our list of metrics is not yet complete. For the following metrics, we are still in the process of evaluating if and how they can applied for our evaluation:

From RAGAS:

- Noise Sensitivity
- Response Relevancy
- Faithfulness
- Multimodal Faithfulness
- Multimodal Relevance
- Factual Correctness
- Semantic Similarity
- Non LLM String Similarity
- BLEU Score
- ROUGE Score
- String Presence
- Exact Match

From Yu et al. [Yu2024]

- Latency
- Diversity
- Noise Robustness
- Negative Rejection
- Counterfactual Robustness
- Readability
- Toxicity
- Perplexity

[Yu2024] [2405.07437] Evaluation of Retrieval-Augmented Generation: A Survey

Where to get the QA Datasets?

- We have a dataset with classifications from previous works at the institute [Kon2022] The dataset consists of Software Architecture Papers from ICSA and ECSA.

- Using this dataset and the full text of the papers, we are creating two types of QA-Datasets. One for a local self created Knowledge Graph and another for the ORKG graph.

- The creation is done semi-automatically using the QA-Generator as mentioned on this slide.

[Kon2022] Evaluation Methods and Replicability of Software Architecture Research Objects | IEEE Conference Publication | IEEE Xplore

Content

First of.. Why is the Retrieval so difficult?

- Before we start with the Algorithm, we would like to point out some of the difficulties that make the retrieval hard. You can skip this, but we recommend to read this to understand the issue.

- **What we want:** The user should be able to input a question in natural language. The system then looks through the Knowledge Graph and returns an answer considering all relevant facts from the graph.
- **How the retrieval works without LLMs:**
  - Classically, the search would be keyword based. Here a exact keyword matching is done on the graph to find relevant facts = (Subject, Predicate, Object) = (Node, Edge, Node).
  - However, doing so has limited success with a natural language question. While we could apply Entity Recognition to find and match words from the question sentence to the graph, this proved to be difficult because of ambiguity issues between the word in the question and in the graph. It’s also hard to consider context in the question using this approach.
- **Using LLMs** allows to consider the context of the question which has the possibility to increase the meaningfulness and relevance of the retrieved information towards the question

First of.. Why is the Retrieval so difficult?

- **But using the LLM for Retrieval is not trivial:** Obviously it is not feasible to just give the whole graph as context to the LLM as some graph contain millions of entries.. This means that we can only provide a selected amount of information from the graph to the LLM. Therefore, given a Question:
    1. How to select what we give the LLM as context?
    2. Where in the Graph should we even start looking for the context of interest?
    3. How do we determine which triples in the graph are relevant?
    4. LLM calls are costly, so how do we traverse through the graph as efficient as possible?
    5. When should we stop searching to avoid looking though the whole graph?
    6. Most graphs are directed, when do we need to traverse “against the direction”?
    7. What if we need to traverse multiple paths to find the answer?
    8. How do we merge information if multiple triples from the graph are needed?
    9. Information is simplified when stored in a graph because whole sentences or paragraphs are now stored as a (Subject, Predicate, Object) format inheritably losing context. Therefore, how do we process the triple information?

We are just at the Beginning

- The literature is currently not able to fully answer each question. Even more, the research of LLM guided KGQA retrievers is continuously evolving and just at the beginning.

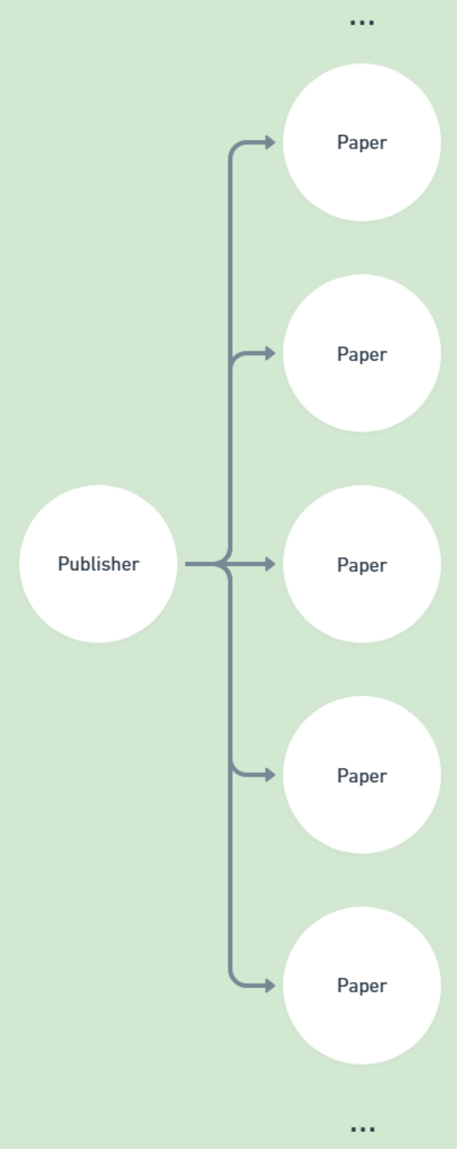
- **One crucial simplification:** Because it is currently unsolved how to effectively find an entry point to the graph for a given question. All existing Knowledge Graph Question Answering Datasets for these retrievers at least include the following information:

1. **Topic Entity:** The entry point to the graph is already provided with the question!
2. **Question:** The question
3. **Golden Answer:** The expected answer based on the contents of a graph

The Width is Difficult

- We want to highlight the following issue: **What if we need to traverse multiple paths to find the answer?** Which is classified as a **Multi-Fact Retrieval**, meaning that it requires multiple triples from the graph to give an answer to a question.
- Current Retrievers struggle with these types of questions. **But why?** Consider the following question: “What papers published by IEEE talk about the concept of prompt engineering on LLMs?”
- To answer this question, every paper with an edge to the publisher needs to be processed to find out if it contains relevant content for the question. As illustrated on the right graph this question quickly becomes very difficult as the **WIDTH** of the publisher can be very large (possible more than thousands of edges)
- We highlight this issue, because during development of our own retriever we concentrated on solving this problem

Sample Graph



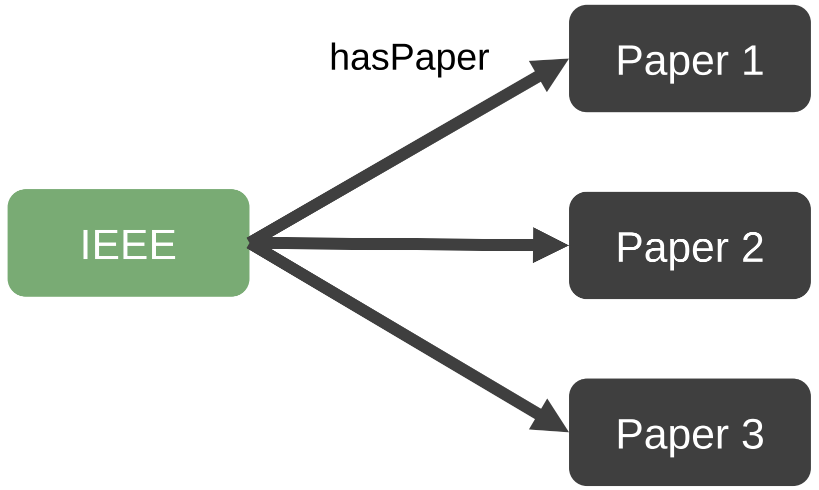
Our new Retrieval Approach

- **Given:** [Question, Topic-Entity] = [“What papers published by IEEE talk about the concept of prompt engineering on LLMs?”, “IEEE”]

We start here
IEEE

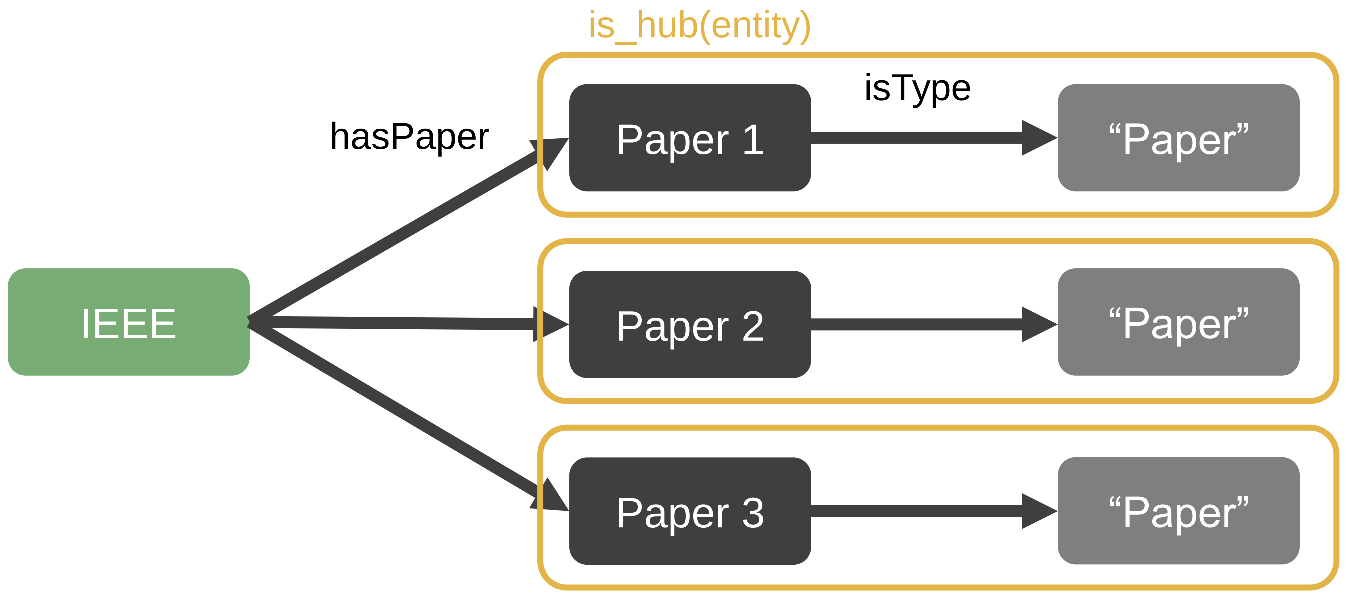
Our new Retrieval Approach

- **Traverse:** We traverse one edge (1 Hop) in every direction from the starting entity and gather the adjacent nodes.



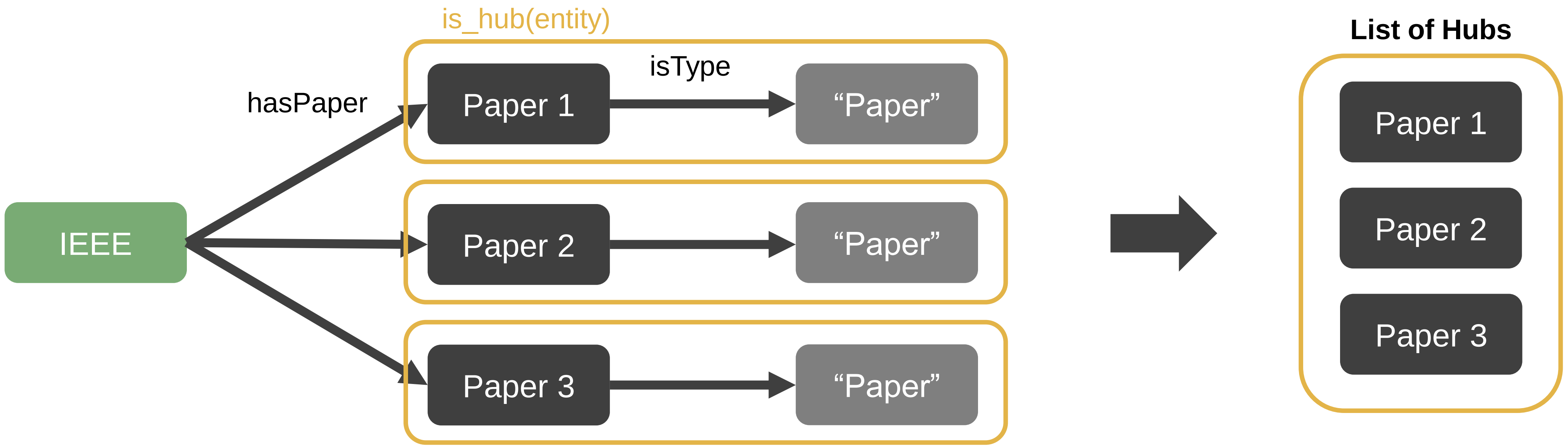
Our new Retrieval Approach

- **Hub Check:** For every node that we found, we check if it is a **Hub**.
  - The definition of a **Hub** might differ between graphs and could be defined for example by the type of the entity or the amount of edges.



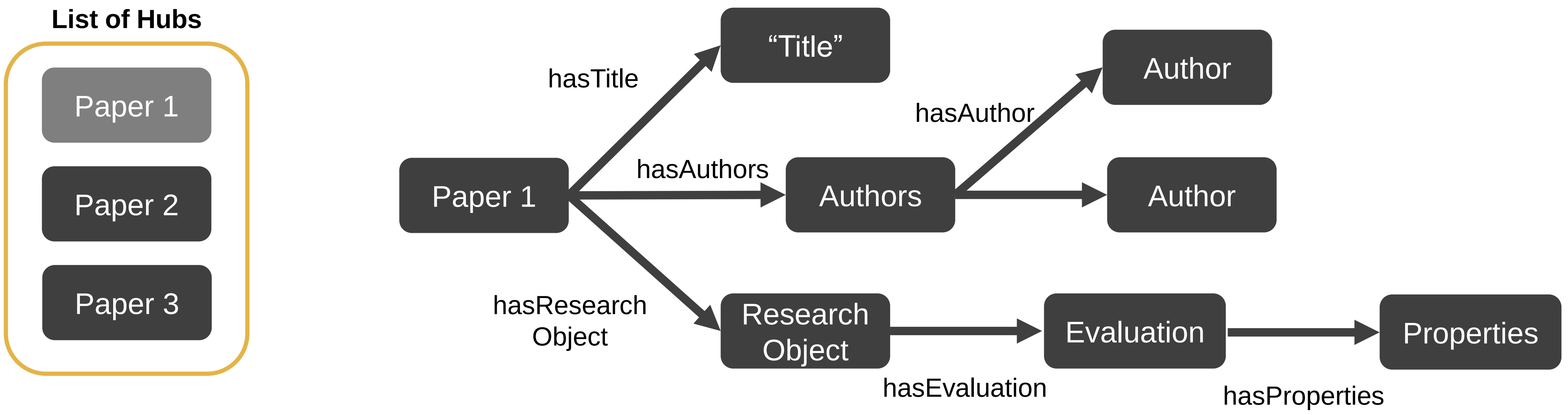
Our new Retrieval Approach

- **Collect all Hubs:** We collect all Hubs of the current level.
  - A level can be defined either by the number of hops from the topic entity (In that case #Level = #Hops)
  - Or by the number of Hubs between the current path and the topic entity (useful if the graph has intermediate nodes between Hubs)



Our new Retrieval Approach

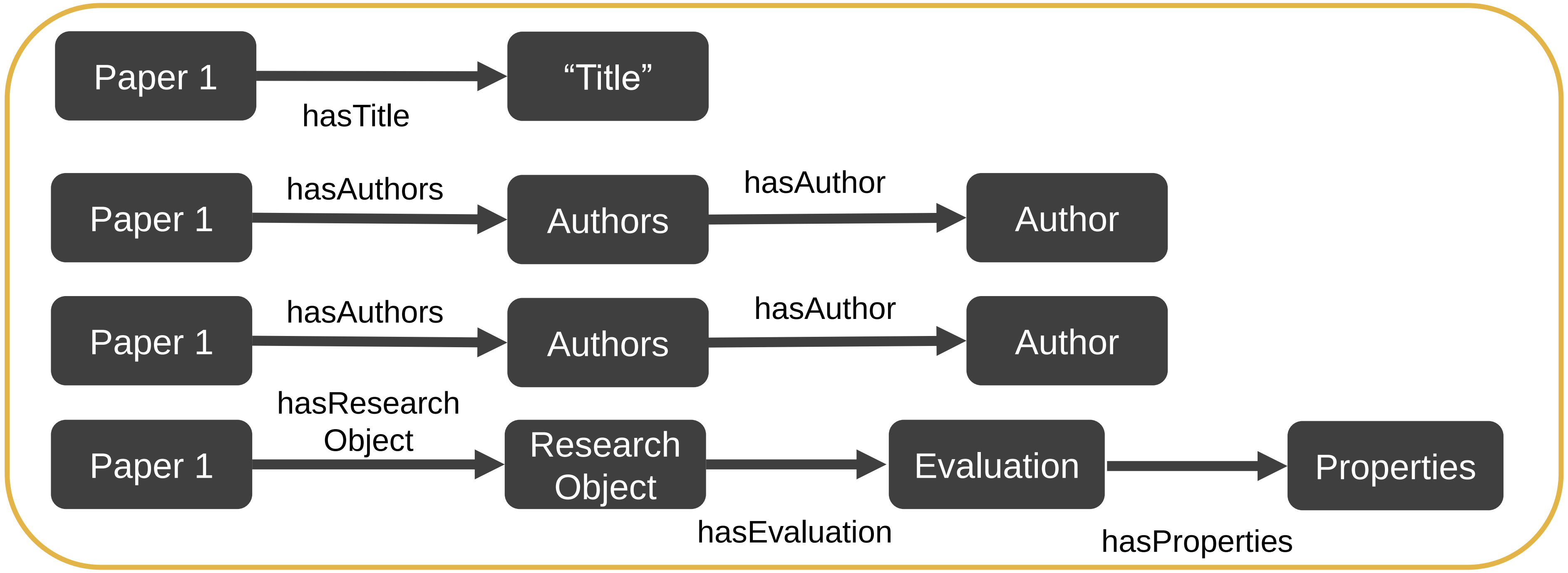
- **Process all Hubs:** Now we process all Hubs (can be done in parallel). This has multiple steps:
  1. Find all **HubPaths**
  2. Convert each collected path to text
  3. Encode each **PathText** to embedding
  4. Convert the Path from the Topic to the Hub-Root to text



Our new Retrieval Approach

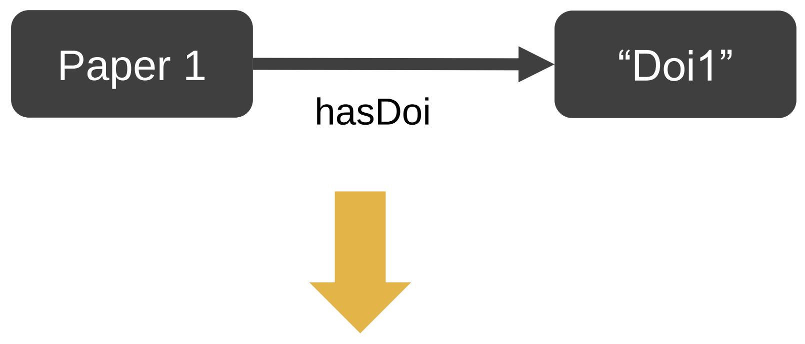
- **Find all HubPaths:** We find all possible paths of a hub
  - A HubPath starts from the root entity of the Hub and ends either when the Graph ends or when another Hub is found

Hub Paths

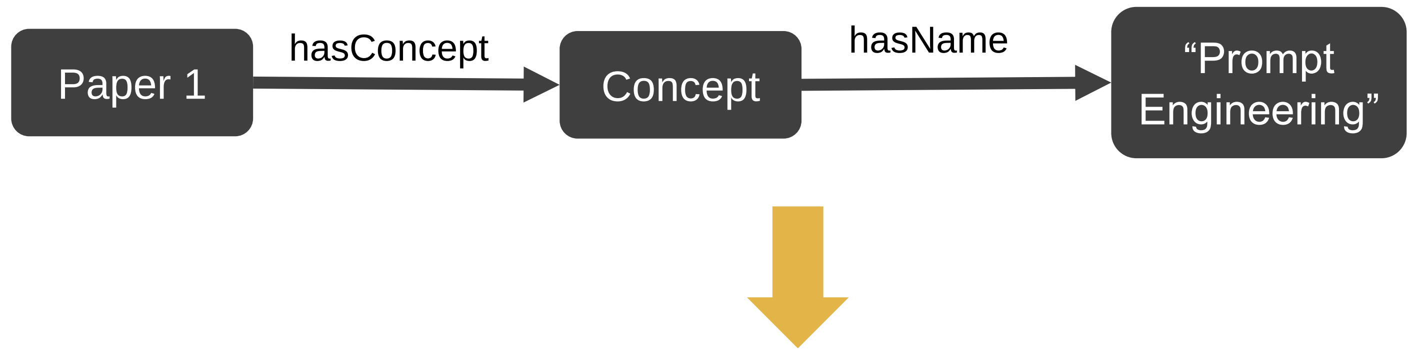


Our new Retrieval Approach

- **Convert each HubPath to Text:** We convert each path to a text that fully describes the path using a LLM.



The paper has the doi “Doi1”



The paper with the title “Paper1” has a concept with the name Prompt Engineering

Our new Retrieval Approach

- **Convert HubText to Embedding:** We convert each text to an embedding vector which encodes the text into a vector space with text similar to each other being closer to each other in the given space. This allows to apply distance calculations to find out how close one text is to another.

The paper has the doi "Doi1"

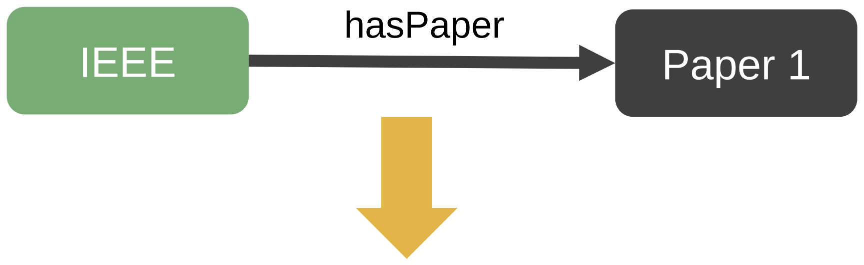
The paper with the title "Paper1" has a concept with the name Prompt Engineering

(0, 1, 5, 2, 1, 5, -5, 1, -2, …)

(6, 2, 5, -5, 2, 9, 5, -1, -7, …)

Our new Retrieval Approach

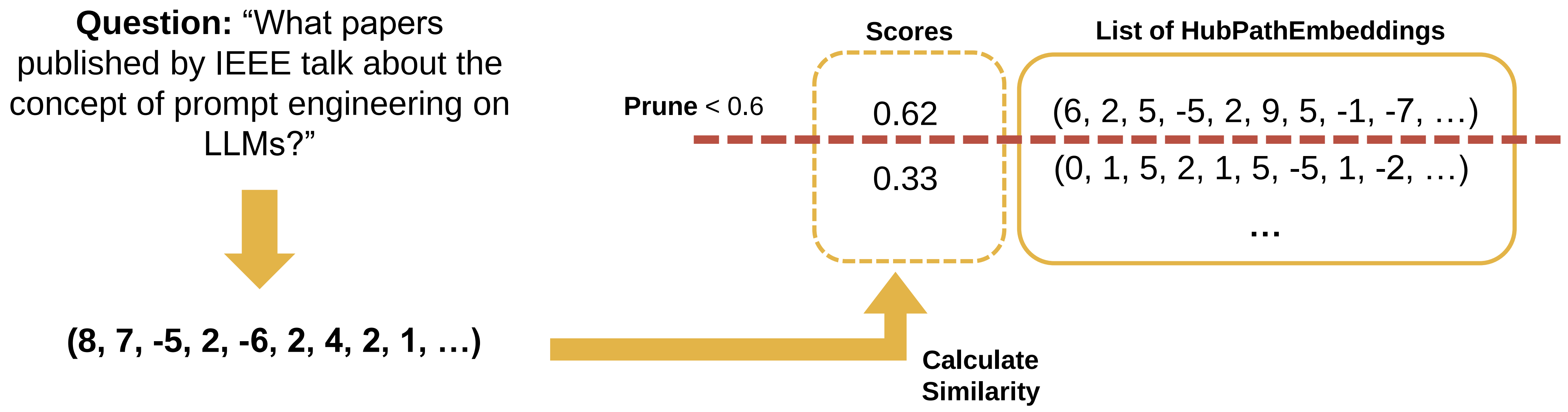
- **Convert the Topic to Root Entity Path to text:** The whole path starting from the Topic entity to the root entity (starting entity) of the Hub is also converted to text.
  - This later allows the retriever to memorize the context between the topic to the knowledge inside the Hubs.



The paper has been published by IEEE

Our new Retrieval Approach

- **Embed the question:** We also encode the question into a vector representation
- **Calculate the similarity:** Based on the question embedding we calculate for each HubPathEmbedding the similarity towards the question.
- **Prune the paths:** Based on a threshold we only collect the most relevant paths for each Hub



Our new Retrieval Approach

- **Collect Top Paths:** For each Hub we collect the top path embeddings. If a Hub has no path above the threshold, we consider the Hub as not relevant.
- **Generate Partial Answers:** For each remaining Hub we use the HubPathTexts as context for an LLM. The LLM is prompted to generate a Partial Answer given the Hub

Question: "What papers published by IEEE talk about the concept of prompt engineering on LLMs?"

TopicPath: The paper has been published by IEEE

+

Top HubPathTexts

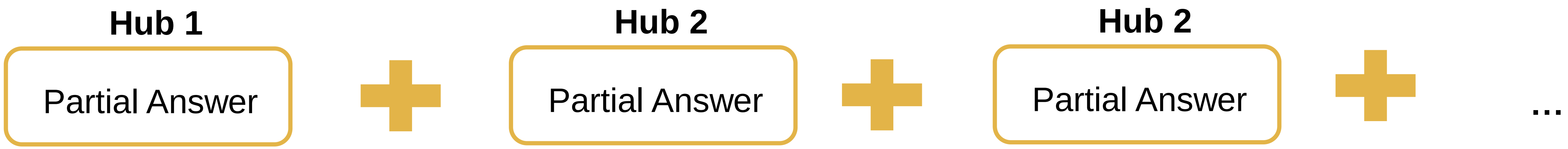
1: "The paper with the title "Paper1" has a concept with the name Prompt Engineering"

N: [Possible more relevant texts]

→ **Partial Answer:** The paper with the title "Paper1" is published by IEEE and includes the concept Prompt Engineering

Our new Retrieval Approach

- **Collect Partial Answers:** Collect each partial answer for each of the remaining relevant Hubs.
- **Generate final Answer:** Generate a final answer using an LLM by consolidating all partial answers.



Final Answer: The following papers are published by IEEE and include prompt engineering as concepts:

Our new Retrieval Approach

- That was the concept of our new retrieval approach
- Our initial tests show **promising results** on multi fact retrieval
- However, processing each hub is **costly**.
  - Finding all possible Hub Paths is calculation intensive
  - Querying each path to get the embedding takes time
- For the first versions it took 5+ mins to get the answer from 30 paper entries
- Through **parallelization** and **offline indexing** techniques we managed to cut the time significantly. It now takes around 10-20 seconds to get the answer for 30 paper entries
- The time increases however with more papers added to the graph. We have to do more testing to evaluate the scalability.

Our new Retrieval Approach

- The code for the retriever is available here
- The currently implemented version is very much in a **prototype state**. Meaning that the logic has been implemented but the code is not yet “in-shape”.
- Feedback should therefore focus on the **logic** of the current algorithm and possible points of **improvement**.

Content

User Guide for the Code Review

- At the current point in time, I haven’t prepared a full user/developer guide yet
- Therefore, in the following slides I will give you some instructions that you can follow to try out the functionality of the current implementation
- **First of:** Start by cloning the repository and installing the requirements as explained in the readme file. When done, go to the next page.
- **Second:** Create a Weights&Biases Account and login. When going through the user interaction guide you will be prompted to login using a API key from your account.

User Guides

1. Question Answering on our self created Graph
2. Question Answering on the ORKG
3. Question Answering with other Retrievers
4. Experimentation

User Guide for the Code Review

Question Answering on our self created Graph

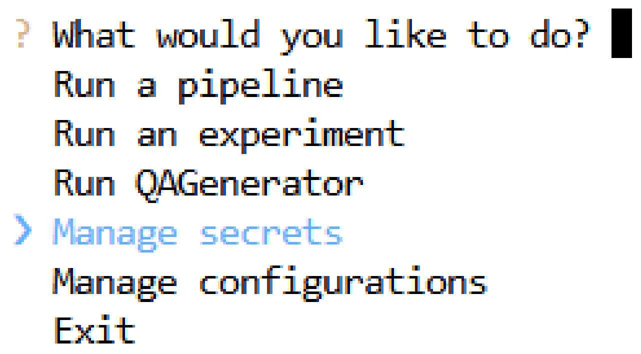
User Guide for the Code Review: Question Answering on our self created Graph

- We have created a Knowledge Graph that can be easily loaded on a local machine
- For the creation of this graph a LLM has processed full text information from papers. Furthermore, a classification schema was used
- Because the creation A) Takes long and B) Needs many LLM tokens, we cached the data necessary for the graph creation and provide it in the repository
- In the following guide you will use this graph to ask questions on the data.

1. **Start the CLI-Controller:** Open a terminal inside of the root directory
2. **Start the CLI-Controller:** Activate the virtual environment
   1. Windows: *venv\Scripts\activate*
   2. Linux: *source venv/bin/activate*
3. **Start the CLI-Controller:** Go to the app directory *cd sqa-system/src/app*

User Guide for the Code Review: Question Answering on our self created Graph

1. **Start the CLI-Controller:** Go to the app directory *cd sqa-system/src/app*
2. **Start the CLI-Controller:** Run the controller with *python cli_controller.py*

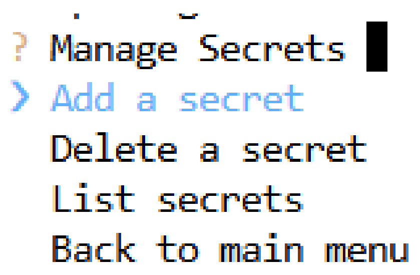


```
? What would you like to do?
  Run a pipeline
  Run an experiment
  Run QAGenerator
❯ Manage secrets
  Manage configurations
  Exit
```

3. **Navigate:** Select “Manage secrets” to add your openai api key
    1. **Note:** If you have done this before, maybe in another run or guide, you do not have to add your api key again.

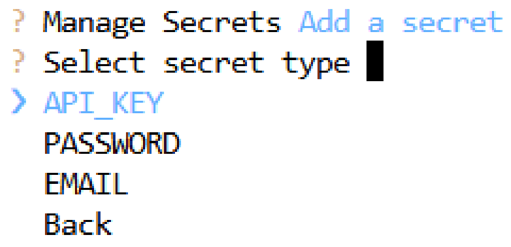
User Guide for the Code Review: Question Answering on our self created Graph

1. **Add your OpenAI API key:** Select “Add a secret”



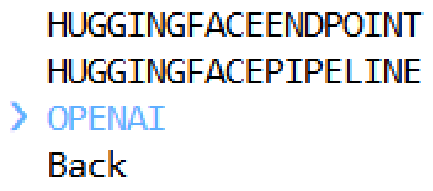
```
? Manage Secrets
❯ Add a secret
  Delete a secret
  List secrets
  Back to main menu
```

2. **Add your OpenAI API key:** Select “API_KEY”



```
? Manage Secrets Add a secret
? Select secret type
❯ API_KEY
  PASSWORD
  EMAIL
  Back
```

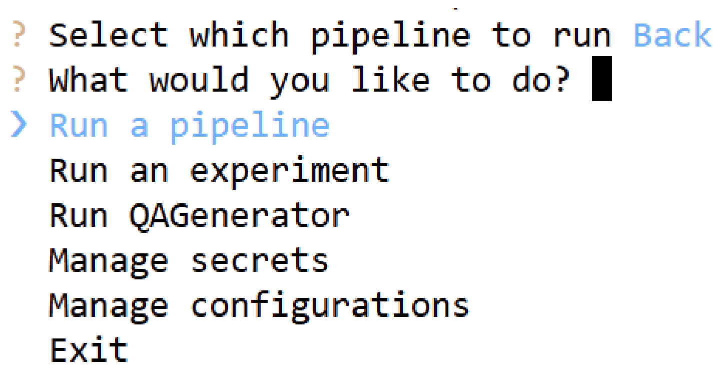
3. **Add your OpenAI API key:** Select “OPENAI”
4. **Add your OpenAI API key:** Add your key and confirm



```
  HUGGINGFACEENDPOINT
  HUGGINGFACEPIPELINE
❯ OPENAI
  Back
```

User Guide for the Code Review: Question Answering on our self created Graph

1. **Navigate:** Navigate back to the main menu



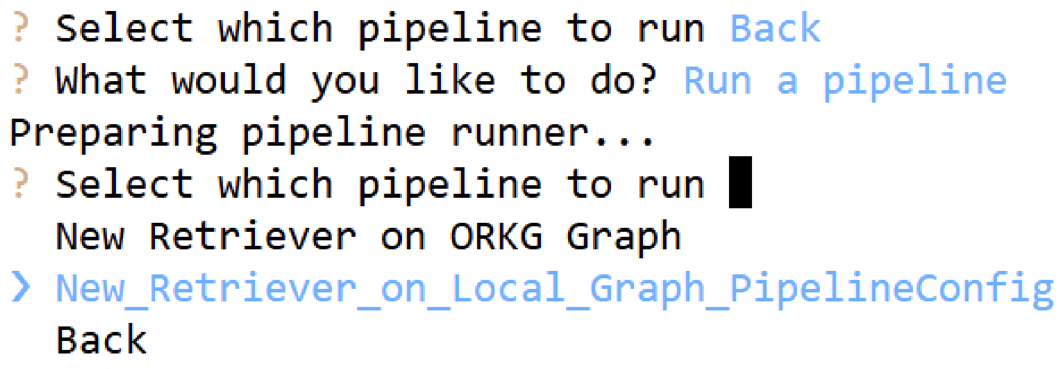
```
? Select which pipeline to run Back
? What would you like to do?
❯ Run a pipeline
  Run an experiment
  Run QAGenerator
  Manage secrets
  Manage configurations
  Exit
```

2. **Navigate:** Select “Run a pipeline”

User Guide for the Code Review: Question Answering on our self created Graph

- We already prepared two pipeline configurations for you that you can run.

1. **Select Pipeline:** Select the following configuration "New_Retriever_on_Local_Graph_PipelineConfig"



```
? Select which pipeline to run Back
? What would you like to do? Run a pipeline
Preparing pipeline runner...
? Select which pipeline to run █
  New Retriever on ORKG Graph
❯ New_Retriever_on_Local_Graph_PipelineConfig
  Back
```

2. **Wait for the Graph to be created:** While waiting, open the log file to see the progress of a) the creation and b) the retrieval process you will be executing afterwards. In your file explorer of choice, go to *sqa-system/logs* and open *log_file.log*

User Guide for the Code Review: Question Answering on our self created Graph

1. **Look at the graph:** Find the graph that has just been created and open it. The graph can be found in the folder *sqa-system/data/knowledge_base/knowledge_graphs* it is probably named: local_rdf_graph_434b34f21befb4bac2500ec8d2f9e2c6.ttl
2. **Find the papers:** Press STRG+F to open the search bar. Enter: *http://ressource.org/publication/*
3. **Analyze the Paper Structure:** Review the structure of the papers. You can use this to create a question. But we will also give you some example questions.

User Guide for the Code Review: Question Answering on our self created Graph

1. **Ask Questions:** In the CLI the interaction should have started by now. It asks you to enter a question. We provide you some examples based on the contents of the graph **(Leave all topic entity values empty):**
   1. **Question**: Which papers that are published by IEEE have Architecture Pattern as a research object? **Topic Entity id:** http://ressource.org/publisher_5 **Question:** Explain to me the concept of Semi-Active Replication and Active Replication. **Topic Entity id:** http://ressource.org/publisher_5 .
   2. **Question:** What is the conclusion of the paper 'Testing the Implementation of Concurrent AUTOS AR Drivers Against Architecture Decisions'? **Topic Entity id:** http://ressource.org/publication/10.1109/ICSA.2019.00026 .
   3. **Question:** Explain to me the concept of Semi-Active Replication and Active Replication. **Topic Entity id:** http://ressource.org/publisher_5 .
   4. **Question:** What are the threats to validity in the paper 'An Exploratory Study of Naturalistic Decision-Making in Complex Software Architecture Environments'? **Topic Entity id:** http://ressource.org/publisher_103 .
2. **Validate the Answer:** You can validate the answer by opening the graph (see slide before) and searching for the answer entity given by the LLM with STRG + F

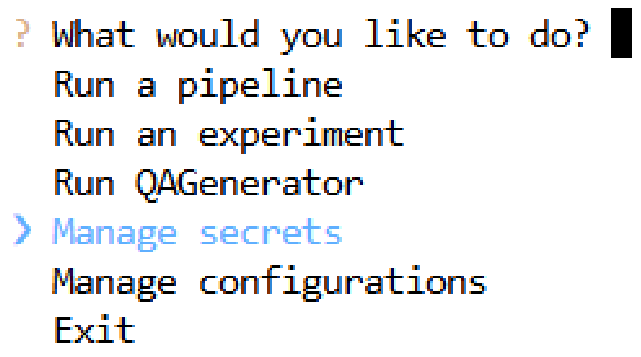
User Guide for the Code Review
Question Answering on the ORKG

User Guide for the Code Review: Question Answering on the ORKG

- The following guide will explain to you how you can use the SQA-System and our new retriever to ask questions on the ORKG
- First of, this is the link to the "Software Architecture and Design" Research Field in the Graph: Software Architecture and Design – ORKG
- As you can see it currently contains 25 papers.
- Because we need to set a entry point (topic entity) for the graph, **for this user guide only use the "Software Architecture and Design" Research Field or a specific paper as a entry point.** The reason for this is, that our new retriever does indexing on the first retrieval. This indexing needs to process the subgraph starting from the topic entity to the max depth defined in the retrievers parameters. Therefore, because the "Software Architecture and Design" Research Field doesn't contain a large subgraph, it is not costly to do testing on this. (We also cached this specific subgraph's data which you automatically downloaded by installing the repo). Therefore doing inference has very very little costs (less than 0.01cent). But when selecting another Research Field for example "Software Engineering", the subgraph contains thousands of entries. Which a) takes long to index and b) can be costly.

User Guide for the Code Review: Question Answering on the ORKG

1. **Start the CLI-Controller:** Go to the app directory *cd sqa-system/src/app*
2. **Start the CLI-Controller:** Run the controller with *python cli_controller.py*

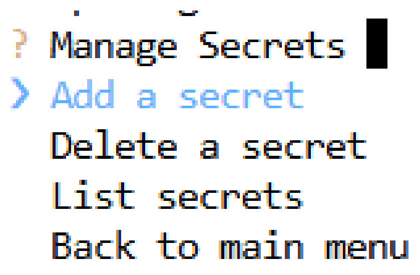


```
? What would you like to do?
  Run a pipeline
  Run an experiment
  Run QAGenerator
❯ Manage secrets
  Manage configurations
  Exit
```

3. **Navigate:** Select “Manage secrets” to add your openai api key
    1. **Note:** If you have done this before, maybe in another run or guide, you do not have to add your api key again.

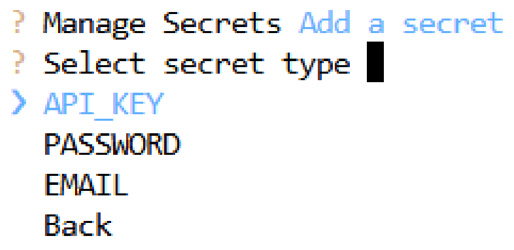
User Guide for the Code Review: Question Answering on the ORKG

1. **Add your OpenAI API key:** Select “Add a secret”



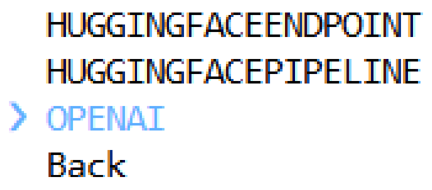
```
? Manage Secrets
❯ Add a secret
  Delete a secret
  List secrets
  Back to main menu
```

2. **Add your OpenAI API key:** Select “API_KEY”



```
? Manage Secrets Add a secret
? Select secret type
❯ API_KEY
  PASSWORD
  EMAIL
  Back
```

3. **Add your OpenAI API key:** Select “OPENAI”
4. **Add your OpenAI API key:** Add your key and confirm

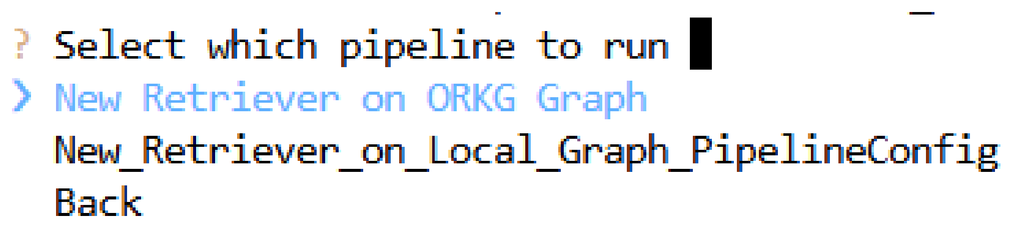


```
  HUGGINGFACEENDPOINT
  HUGGINGFACEPIPELINE
❯ OPENAI
  Back
```

User Guide for the Code Review: Question Answering on the ORKG

- We already prepared two pipeline configurations for you that you can run.

1. **Select Pipeline:** Select the following configuration
   “New Retriever on ORKG Graph”



```
? Select which pipeline to run
❯ New Retriever on ORKG Graph
  New_Retriever_on_Local_Graph_PipelineConfig
  Back
```

2. **(Open the log file to see process):** Open the log file to see the retrieval process. In your file explorer of choice, go to *sqa-system/logs* and open *log_file.log*

User Guide for the Code Review: Question Answering on the ORKG

1. **Look at the graph:** Open the Research Field Software Architecture and Design – ORKG and analyze the papers in it.
2. **Analyze the Paper Structure:** Review the structure of the papers. You can use this to create a question. But we will also give you some example questions.
3. **Ask Questions:** In the CLI the interaction should have started by now. It asks you to enter a question. We provide you some examples based on the contents of the graph:
    1. **Question**: Which paper aims to provide a co-design tool in the context of VR development? **Topic Entity id:** R659055 **Topic entity value:** Can be left empty.
    2. **Question**: What papers have virtual reality as their research problem? **Topic Entity id:** R659055 **Topic entity value:** Can be left empty.

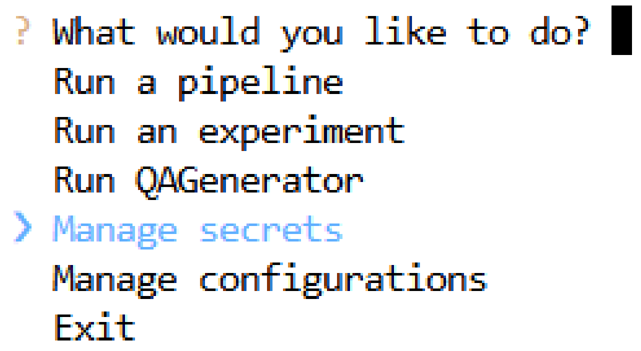
User Guide for the Code Review
Question Answering with other Retrievers

User Guide for the Code Review: Question Answering with other Retrievers

- The previous guides used our “Novel Retriever” to retrieve data from either the local graph or the ORKG.
- The following guide will explain to you how you can use other KGQA retrievers. We have prepared two Retrievers from the literature that you can try out 1) Think-on-Graph (TOG) and 2) StructGPT.

User Guide for the Code Review: Question Answering with other Retrievers

1. **Start the CLI-Controller:** Go to the app directory *cd sqa-system/src/app*
2. **Start the CLI-Controller:** Run the controller with *python cli_controller.py*

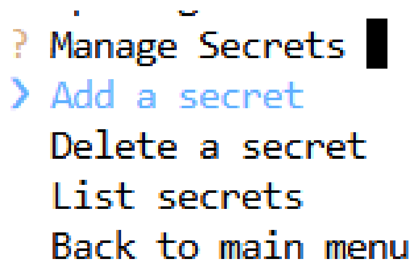


```
? What would you like to do?
  Run a pipeline
  Run an experiment
  Run QAGenerator
❯ Manage secrets
  Manage configurations
  Exit
```

3. **Navigate:** Select “Manage secrets” to add your openai api key
    1. **Note:** If you have done this before, maybe in another run or guide, you do not have to add your api key again.

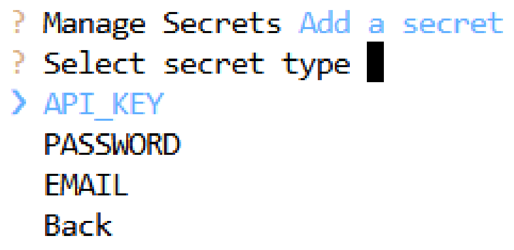
User Guide for the Code Review: Question Answering with other Retrievers

1. **Add your OpenAI API key:** Select “Add a secret”



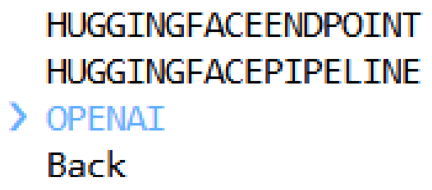
```
? Manage Secrets
❯ Add a secret
  Delete a secret
  List secrets
  Back to main menu
```

2. **Add your OpenAI API key:** Select “API_KEY”



```
? Manage Secrets Add a secret
? Select secret type
❯ API_KEY
  PASSWORD
  EMAIL
  Back
```

3. **Add your OpenAI API key:** Select “OPENAI”
4. **Add your OpenAI API key:** Add your key and confirm

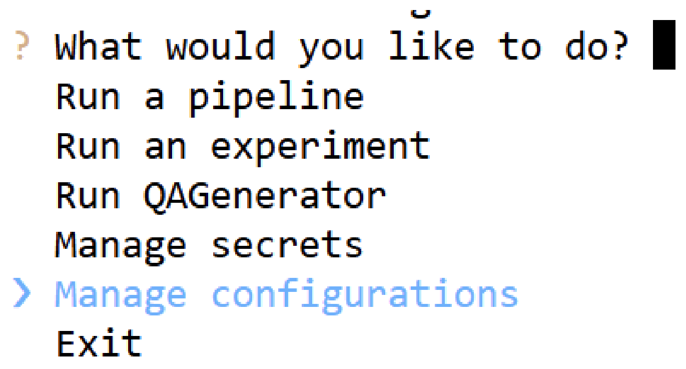


```
  HUGGINGFACEENDPOINT
  HUGGINGFACEPIPELINE
❯ OPENAI
  Back
```

User Guide for the Code Review: Question Answering with other Retrievers

- In this guide, instead of using a predefined configuration that I have prepared for you, I guide you through how you can set up such a configuration yourself.

1. **Create Configuration:** In the main menu select “Manage configurations”

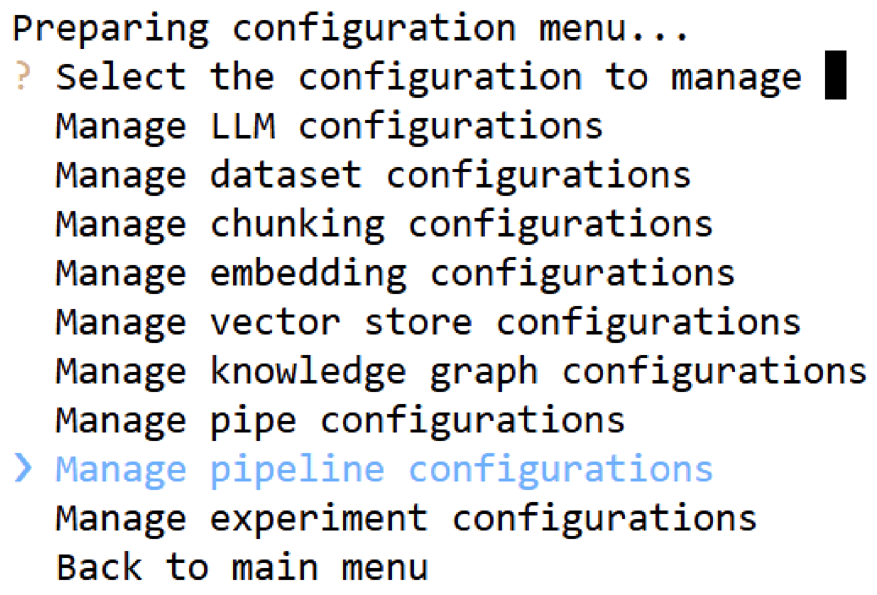


```
? What would you like to do?
  Run a pipeline
  Run an experiment
  Run QAGenerator
  Manage secrets
❯ Manage configurations
  Exit
```

2. **(Open the log file to see process):** Open the log file to see the retrieval process. In your file explorer of choice, go to *sqa-system/logs* and open *log_file.log*

User Guide for the Code Review: Question Answering with other Retrievers

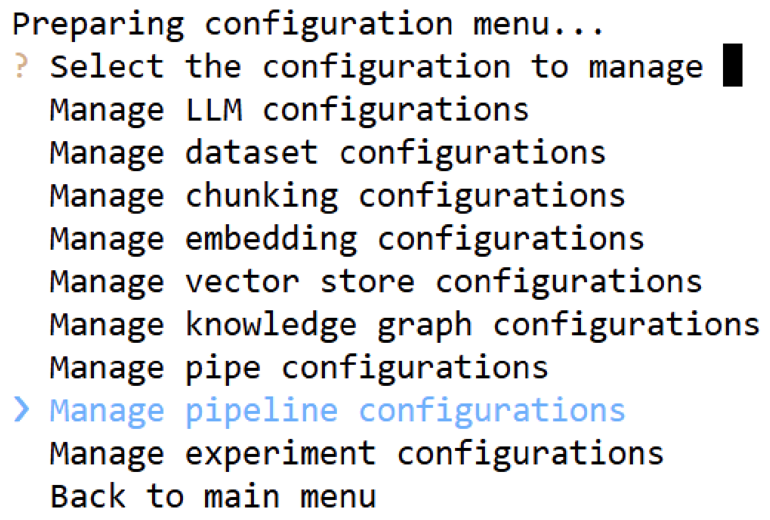
1. **Create Configuration:** We would like to add a new pipeline, therefore we select "Manage pipeline configurations"



```
Preparing configuration menu...
? Select the configuration to manage
  Manage LLM configurations
  Manage dataset configurations
  Manage chunking configurations
  Manage embedding configurations
  Manage vector store configurations
  Manage knowledge graph configurations
  Manage pipe configurations
❯ Manage pipeline configurations
  Manage experiment configurations
  Back to main menu
```

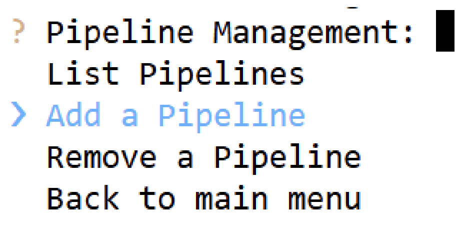
User Guide for the Code Review: Question Answering with other Retrievers

1. **Create Configuration:** We would like to add a new pipeline, therefore we select “Manage pipeline configurations”



```
Preparing configuration menu...
? Select the configuration to manage
  Manage LLM configurations
  Manage dataset configurations
  Manage chunking configurations
  Manage embedding configurations
  Manage vector store configurations
  Manage knowledge graph configurations
  Manage pipe configurations
❯ Manage pipeline configurations
  Manage experiment configurations
  Back to main menu
```

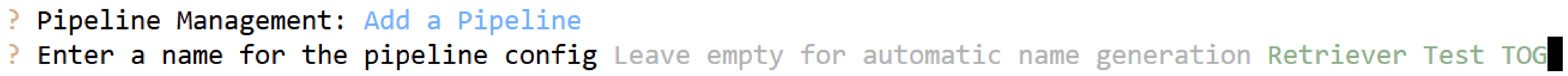
2. **Create Configuration:** Then select “Add a Pipeline”



```
? Pipeline Management:
  List Pipelines
❯ Add a Pipeline
  Remove a Pipeline
  Back to main menu
```

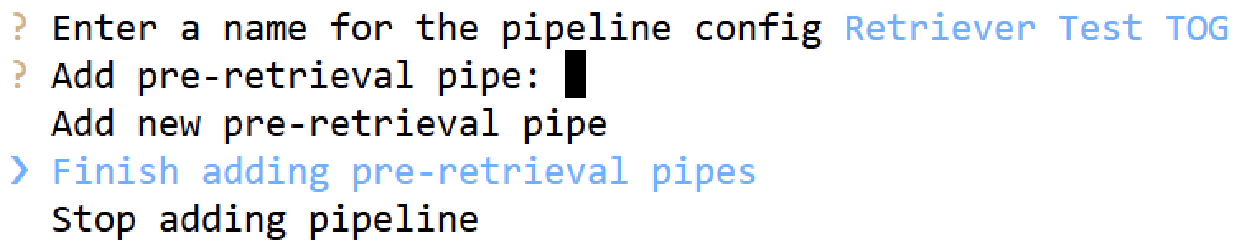
User Guide for the Code Review: Question Answering with other Retrievers

1. **Create Configuration:** This now guides you through the process of creating a new pipeline.
    1. Give the pipeline the name “Retriever Test TOG”



```
? Pipeline Management: Add a Pipeline
? Enter a name for the pipeline config Leave empty for automatic name generation Retriever Test TOG
```

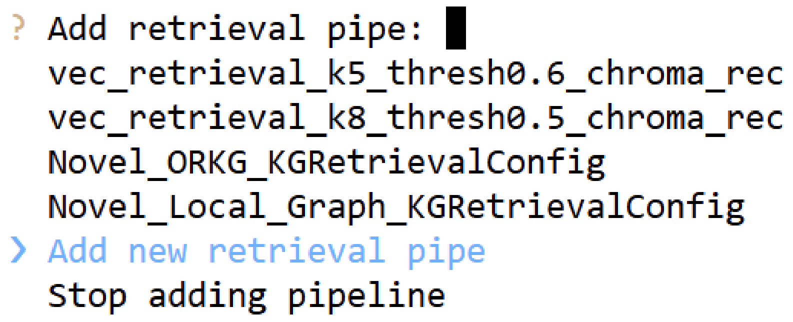
    2. Select “Finish adding pre-retrieval pipes”



```
? Enter a name for the pipeline config Retriever Test TOG
? Add pre-retrieval pipe:
  Add new pre-retrieval pipe
❯ Finish adding pre-retrieval pipes
  Stop adding pipeline
```

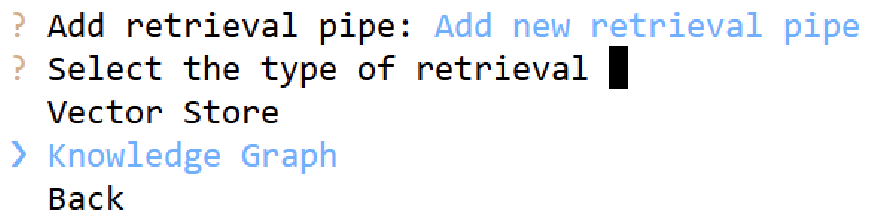
User Guide for the Code Review: Question Answering with other Retrievers

1. **Create Configuration:** This now guides you through the process of creating a new pipeline.
   1. Now we want to create a new retrieval pipe by selecting "Add new retrieval pipe"



```
? Add retrieval pipe: █
  vec_retrieval_k5_thresh0.6_chroma_rec
  vec_retrieval_k8_thresh0.5_chroma_rec
  Novel_ORKG_KGRetrievalConfig
  Novel_Local_Graph_KGRetrievalConfig
❯ Add new retrieval pipe
  Stop adding pipeline
```

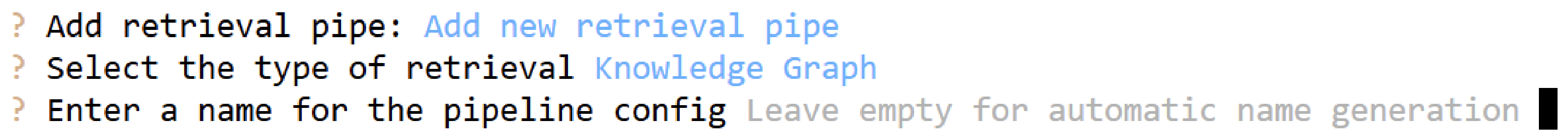
   2. Select "Knowledge Graph"



```
? Add retrieval pipe: Add new retrieval pipe
? Select the type of retrieval █
  Vector Store
❯ Knowledge Graph
  Back
```

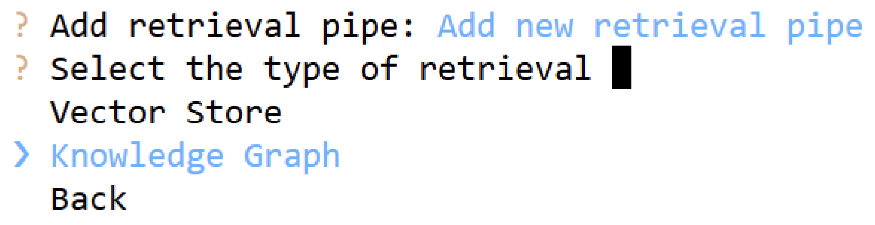
User Guide for the Code Review: Question Answering with other Retrievers

1. **Create Configuration:** This now guides you through the process of creating a new pipeline.
   1. Now we want to create a new retrieval pipe by selecting "Add new retrieval pipe"



```
? Add retrieval pipe: Add new retrieval pipe
? Select the type of retrieval Knowledge Graph
? Enter a name for the pipeline config Leave empty for automatic name generation
```

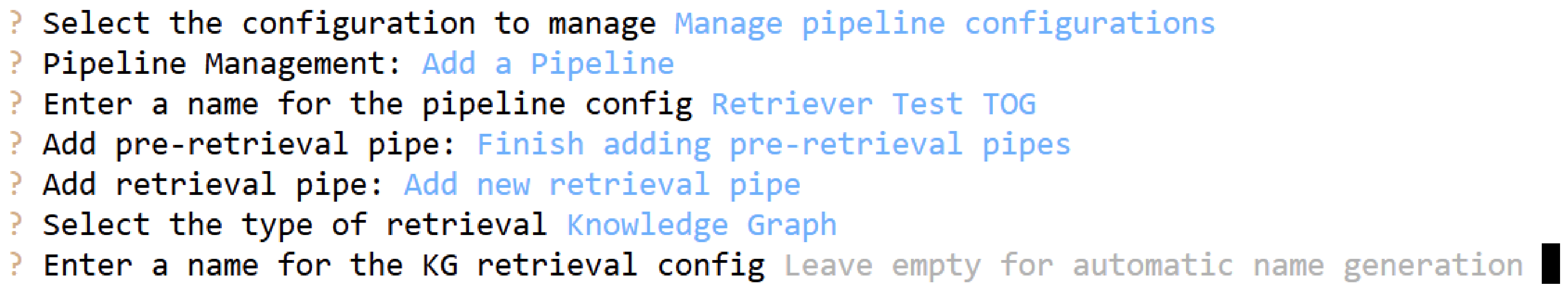
   2. Select "Knowledge Graph"



```
? Add retrieval pipe: Add new retrieval pipe
? Select the type of retrieval
  Vector Store
❯ Knowledge Graph
  Back
```

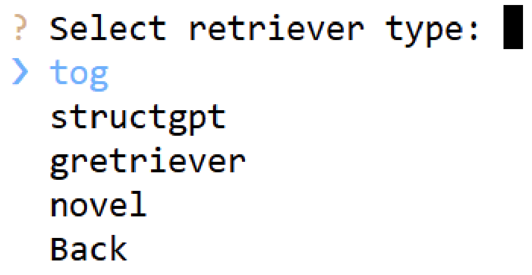
User Guide for the Code Review: Question Answering with other Retrievers

1. **Create Configuration:** This now guides you through the process of creating a new pipeline.
   1. Leave the name empty, just press “Enter”



```
? Select the configuration to manage Manage pipeline configurations
? Pipeline Management: Add a Pipeline
? Enter a name for the pipeline config Retriever Test TOG
? Add pre-retrieval pipe: Finish adding pre-retrieval pipes
? Add retrieval pipe: Add new retrieval pipe
? Select the type of retrieval Knowledge Graph
? Enter a name for the KG retrieval config Leave empty for automatic name generation
```

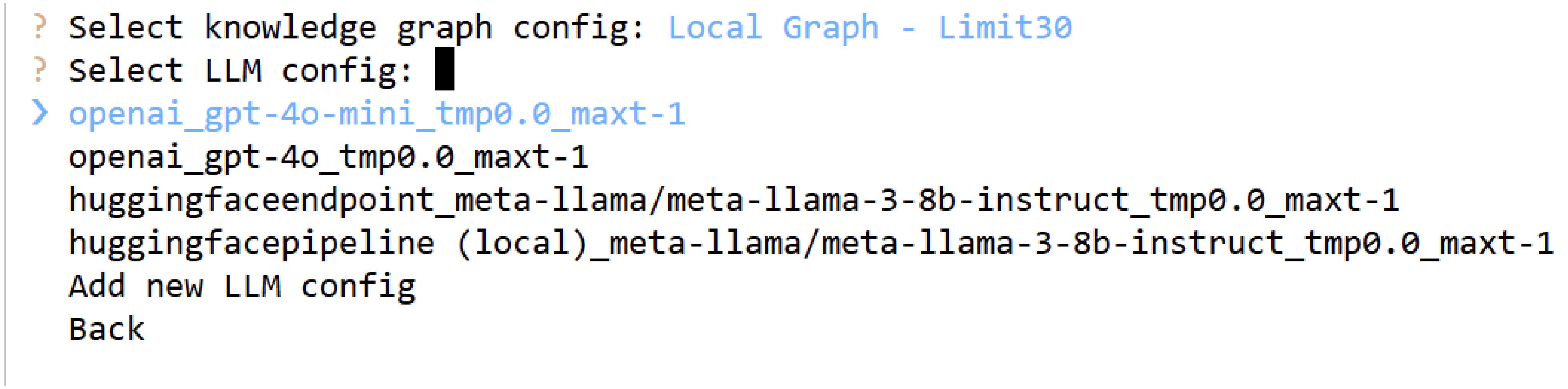
   2. Here you can select the retriever that you want to use. “Novel” is our new retriever. “tog” and “structgpt” are retrievers from the literature. “gretriever” is not working for this guide.



```
? Select retriever type:
❯ tog
  structgpt
  gretriever
  novel
  Back
```

User Guide for the Code Review: Question Answering with other Retrievers

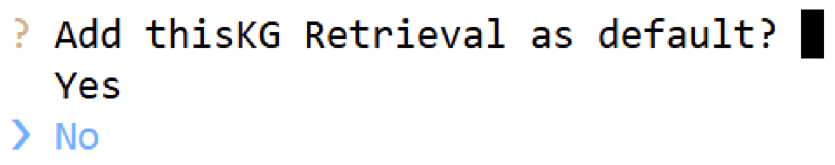
1. **Create Configuration:** Now we need to select the LLM config that the selected retriever will use for inference. We already pre-configurated some LLMs but for this guide we recommend you to use the first option “openai_gpt-4o-mini” because it is very cheap and in comparison almost the same or better performance as the other models. The huggingface models load the model onto your machine which requires good hardware. We also haven’t been able to make them work in a good way with the retrievers because they tend to not follow instructions as well.



```
? Select knowledge graph config: Local Graph - Limit30
? Select LLM config:
❯ openai_gpt-4o-mini_tmp0.0_maxt-1
  openai_gpt-4o_tmp0.0_maxt-1
  huggingfaceendpoint_meta-llama/meta-llama-3-8b-instruct_tmp0.0_maxt-1
  huggingfacepipeline (local)_meta-llama/meta-llama-3-8b-instruct_tmp0.0_maxt-1
  Add new LLM config
  Back
```

User Guide for the Code Review: Question Answering with other Retrievers

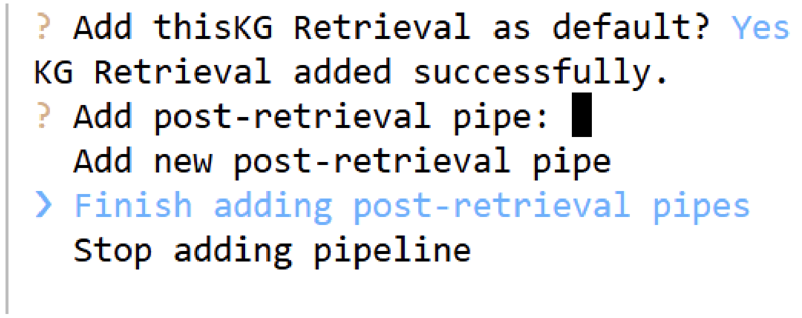
1. **Create Configuration:** The next selections allow you to configurate the retriever itself. Each retriever has its own parameters that can be set. For the code review we recommend to stick to the default parameters.
2. **Create Configuration:** The next step asks you to save this configuration as default. Saying “yes” to this, serializes the configuration to allow you to reuse it for the next run (or share it with other researchers)



```
? Add thisKG Retrieval as default?
  Yes
❯ No
```

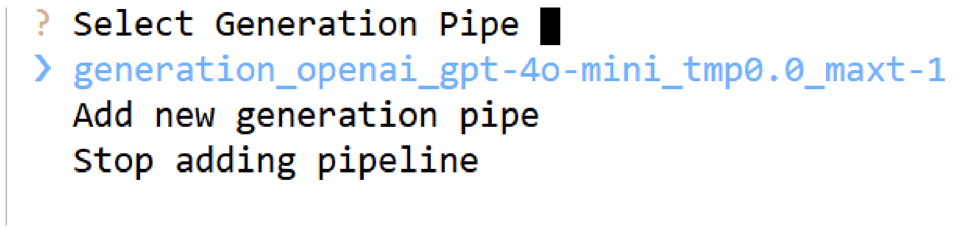
User Guide for the Code Review: Question Answering with other Retrievers

1. **Create Configuration:** When asked if you want to add a post-retrieval pipe select “Finish adding post-retrieval pipes”



```
? Add thisKG Retrieval as default? Yes
KG Retrieval added successfully.
? Add post-retrieval pipe: █
  Add new post-retrieval pipe
❯ Finish adding post-retrieval pipes
  Stop adding pipeline
```

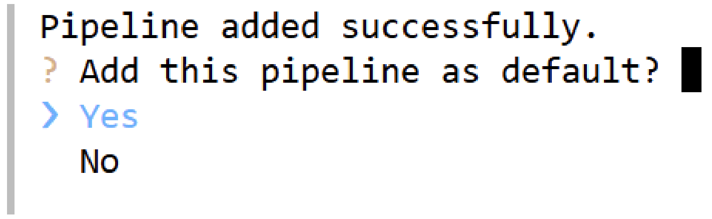
2. **Create Configuration:** The final pipe to add is the generation pipe which forms the final answer. We prepared a gpt-4o-mini generator. Select it “generation_openai_gpt-4o-mini”



```
? Select Generation Pipe █
❯ generation_openai_gpt-4o-mini_tmp0.0_maxt-1
  Add new generation pipe
  Stop adding pipeline
```

User Guide for the Code Review: Question Answering with other Retrievers

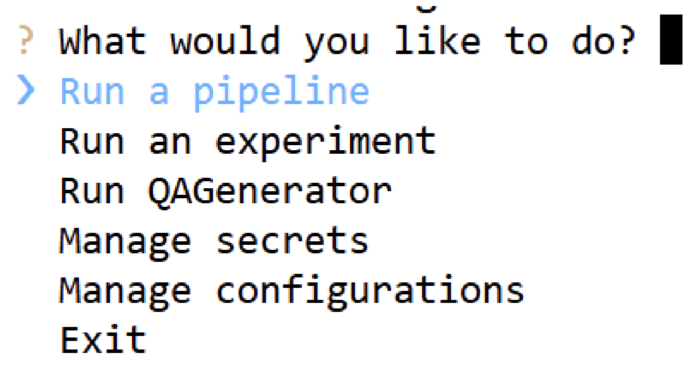
1. **Create Configuration:** The next step asks you to save this configuration as default. Saying "yes" to this, serializes the configuration to allow you to reuse it for the next run (or share it with other researchers)



```
Pipeline added successfully.
? Add this pipeline as default?
❯ Yes
  No
```

User Guide for the Code Review: Question Answering with other Retrievers

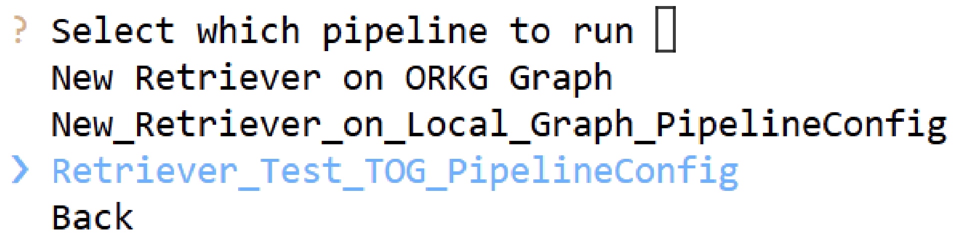
1. **Run the pipeline:** Now that the pipeline configuration has been added, you can run it. Go to the main menu and select “Run a pipeline”



```
? What would you like to do?
❯ Run a pipeline
  Run an experiment
  Run QAGenerator
  Manage secrets
  Manage configurations
  Exit
```

User Guide for the Code Review: Question Answering with other Retrievers

1. **Run the pipeline:** At the bottom of the selection you can see your newly added pipeline configuration. Select and run it



```
? Select which pipeline to run
  New Retriever on ORKG Graph
  New_Retriever_on_Local_Graph_PipelineConfig
❯ Retriever_Test_TOG_PipelineConfig
  Back
```

2. **(Open the log file to see process):** Open the log file to see the retrieval process. In your file explorer of choice, go to *sqa-system/logs* and open *log_file.log*

User Guide for the Code Review: Question Answering with other Retrievers

1. **Ask questions:** Depending on the graph you have selected, you can follow the instructions from the other guides to ask questions on the graph:

    1. *For ORKG press here*
    2. *For the local Graph press here*

User Guide for the Code Review
Experimentation

User Guide for the Code Review: Experimentation

- To allow to evaluate the performance of the retrievers, we can use the implemented ExperimentRunner.
- This guide will show you how to start an experiment

User Guide for the Code Review: Experimentation

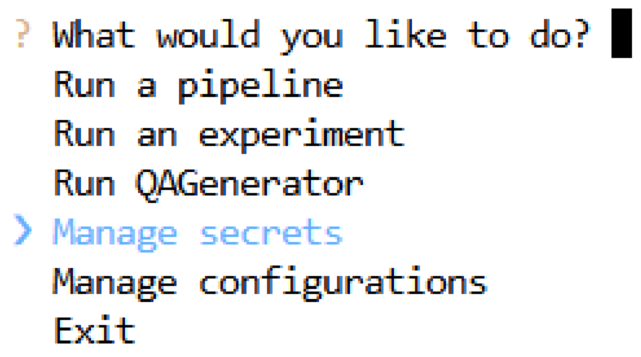
- For the code review we prepared a small experiment on the local graph. Let’s see what we are going to experiment with.
- Open the following link: Default_experiments.json It links to the json file in the repo that contains the experimentation configurations. The experiment for this code review is called “CodeReview_All_Retrievers_on_local” and should be at the top of the json.
- This is the experimentation config that we are going to run. Lets review what it does.
- The config contains a “base_pipeline_config” which in turn contains multiple pipes. If you have done the “Question Answering with other Retrievers” guide you should know by now what pipes are used for. In short: they form the pipeline we are going to execute. The base pipeline is basically a way for the **ExperimentRunner** to know, what the pipeline consists of.
- Another key is “parameter_ranges” which is used by the **ExperimentRunner** in conjunction with the “base_pipeline_config” to generate multiple pipelines that are then run in a batch process. This also allows to do hyperparameter tuning. However, in our case we are just going to replace the “RetrievalPipe” in the base pipeline with our three types of retriever configs “Tog”, “StructGPT” and “Novel”.

User Guide for the Code Review: Experimentation

- The “evaluators” key defines which metrics we want to apply on the results
- The “qa_dataset” key defines a config for the data loader to know which file contains the questions to be asked for the evaluation.
- The dataset we are going to be using is defined here. It’s a small dataset for the purpose of showcasing the experimentation functionality of the system.

User Guide for the Code Review: Experimentation

1. **Start the CLI-Controller:** Go to the app directory *cd sqa-system/src/app*
2. **Start the CLI-Controller:** Run the controller with *python cli_controller.py*

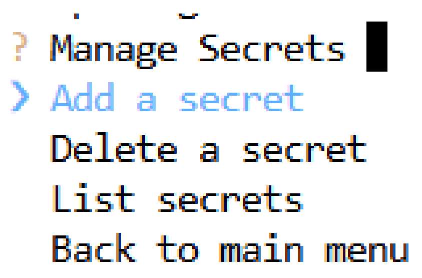


```
? What would you like to do?
  Run a pipeline
  Run an experiment
  Run QAGenerator
❯ Manage secrets
  Manage configurations
  Exit
```

3. **Navigate:** Select “Manage secrets” to add your openai api key
    1. **Note:** If you have done this before, maybe in another run or guide, you do not have to add your api key again.

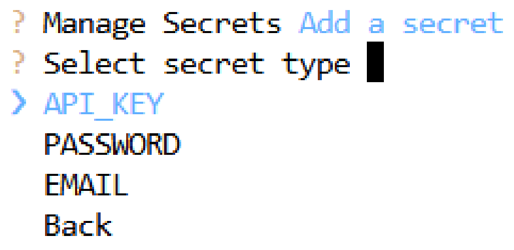
User Guide for the Code Review: Experimentation

1. **Add your OpenAI API key:** Select “Add a secret”



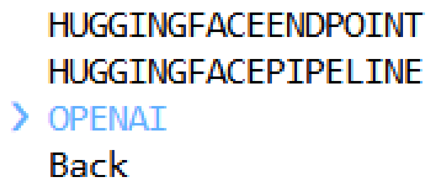
```
? Manage Secrets
❯ Add a secret
  Delete a secret
  List secrets
  Back to main menu
```

2. **Add your OpenAI API key:** Select “API_KEY”



```
? Manage Secrets Add a secret
? Select secret type
❯ API_KEY
  PASSWORD
  EMAIL
  Back
```

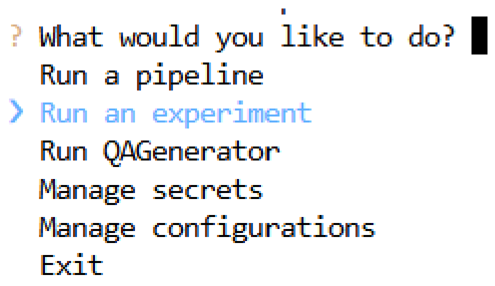
3. **Add your OpenAI API key:** Select “OPENAI”
4. **Add your OpenAI API key:** Add your key and confirm



```
  HUGGINGFACEENDPOINT
  HUGGINGFACEPIPELINE
❯ OPENAI
  Back
```

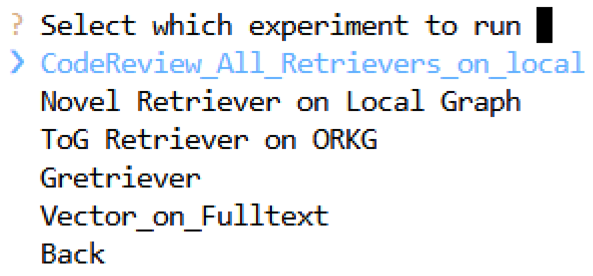
User Guide for the Code Review: Experimentation

1. **Run Experiment:** From the main menu select “Run an experiment”



```
? What would you like to do?
  Run a pipeline
❯ Run an experiment
  Run QAGenerator
  Manage secrets
  Manage configurations
  Exit
```

2. **Run Experiment:** Select “CodeReview_All_Retrievers_on_local”.



```
? Select which experiment to run
❯ CodeReview_All_Retrievers_on_local
  Novel Retriever on Local Graph
  ToG Retriever on ORKG
  Gretriever
  Vector_on_Fulltext
  Back
```

User Guide for the Code Review: Experimentation

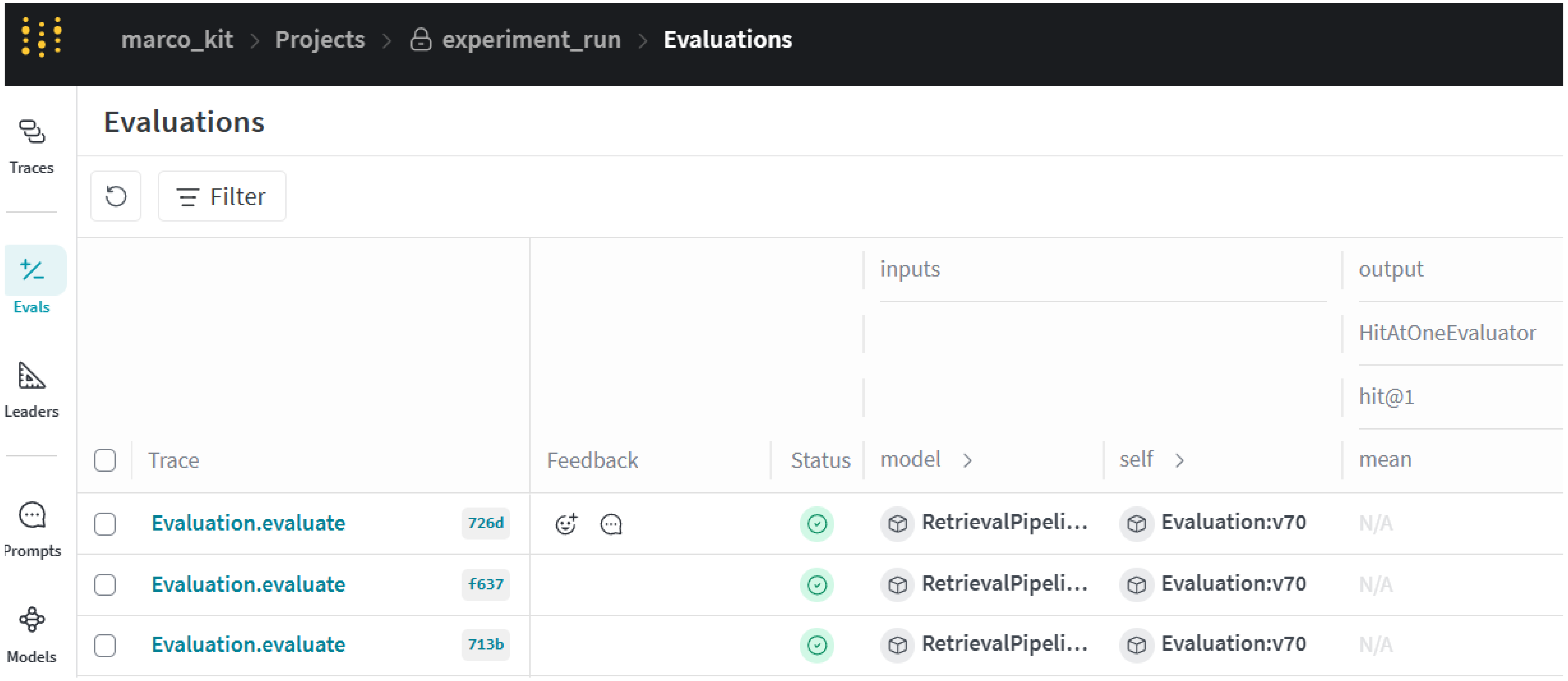
1. **(Open the log file to see process):** Open the log file to see the retrieval process. In your file explorer of choice, go to *sqa-system/logs* and open *log_file.log*

2. **Wait for the experiment to finish**

3. **Look at the results:** To see the results go to your Wandb Dashboard and find the project named “experiment-run”

User Guide for the Code Review: Experimentation

1. Find the “Evals” tab on the left side and press it

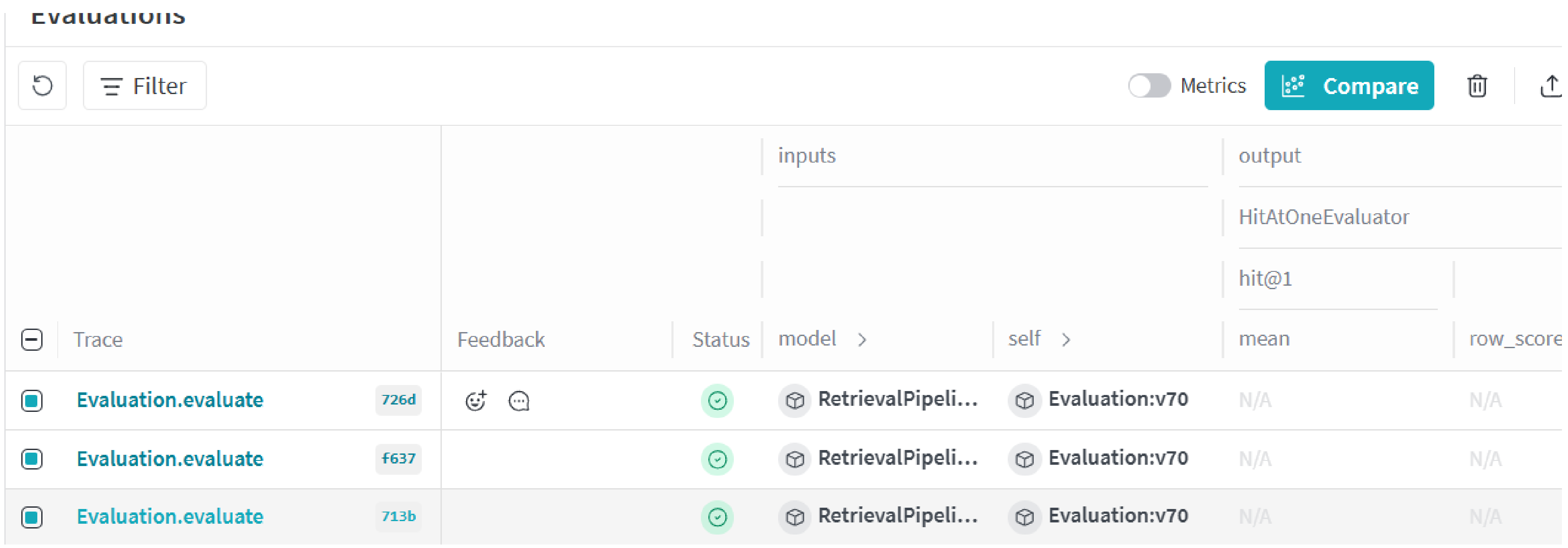


Evaluations

			inputs		output
					HitAtOneEvaluator
					hit@1
Trace	Feedback	Status	model	self	mean
Evaluation.evaluate 726d		✓	RetrievalPipeli...	Evaluation:v70	N/A
Evaluation.evaluate f637		✓	RetrievalPipeli...	Evaluation:v70	N/A
Evaluation.evaluate 713b		✓	RetrievalPipeli...	Evaluation:v70	N/A

User Guide for the Code Review: Experimentation

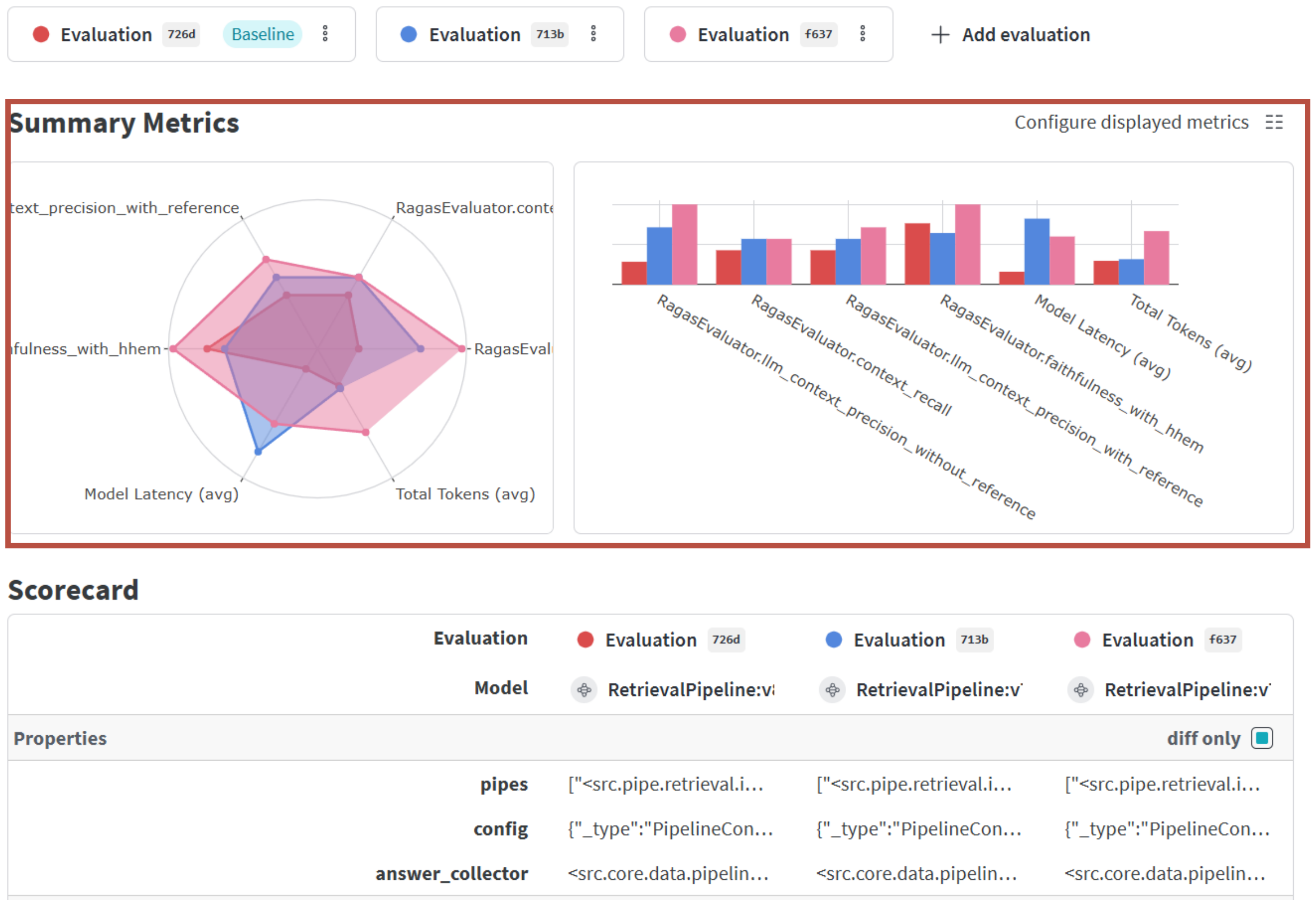
1. Select the top 3 evaluations. These are our 3 retriever experiments that just ran
2. After selecting, press “Compare” on the top right



			inputs		output	
					HitAtOneEvaluator	
					hit@1	
Trace	Feedback	Status	model	self	mean	row_score
Evaluation.evaluate 726d		✓	RetrievalPipeli...	Evaluation:v70	N/A	N/A
Evaluation.evaluate f637		✓	RetrievalPipeli...	Evaluation:v70	N/A	N/A
Evaluation.evaluate 713b		✓	RetrievalPipeli...	Evaluation:v70	N/A	N/A

User Guide for the Code Review: Experimentation

1. A overview should be opening that compares each of the retrievers
2. It shows an overview of the metrics of the retriever
3. Currently we don’t have all metrics implemented yet.

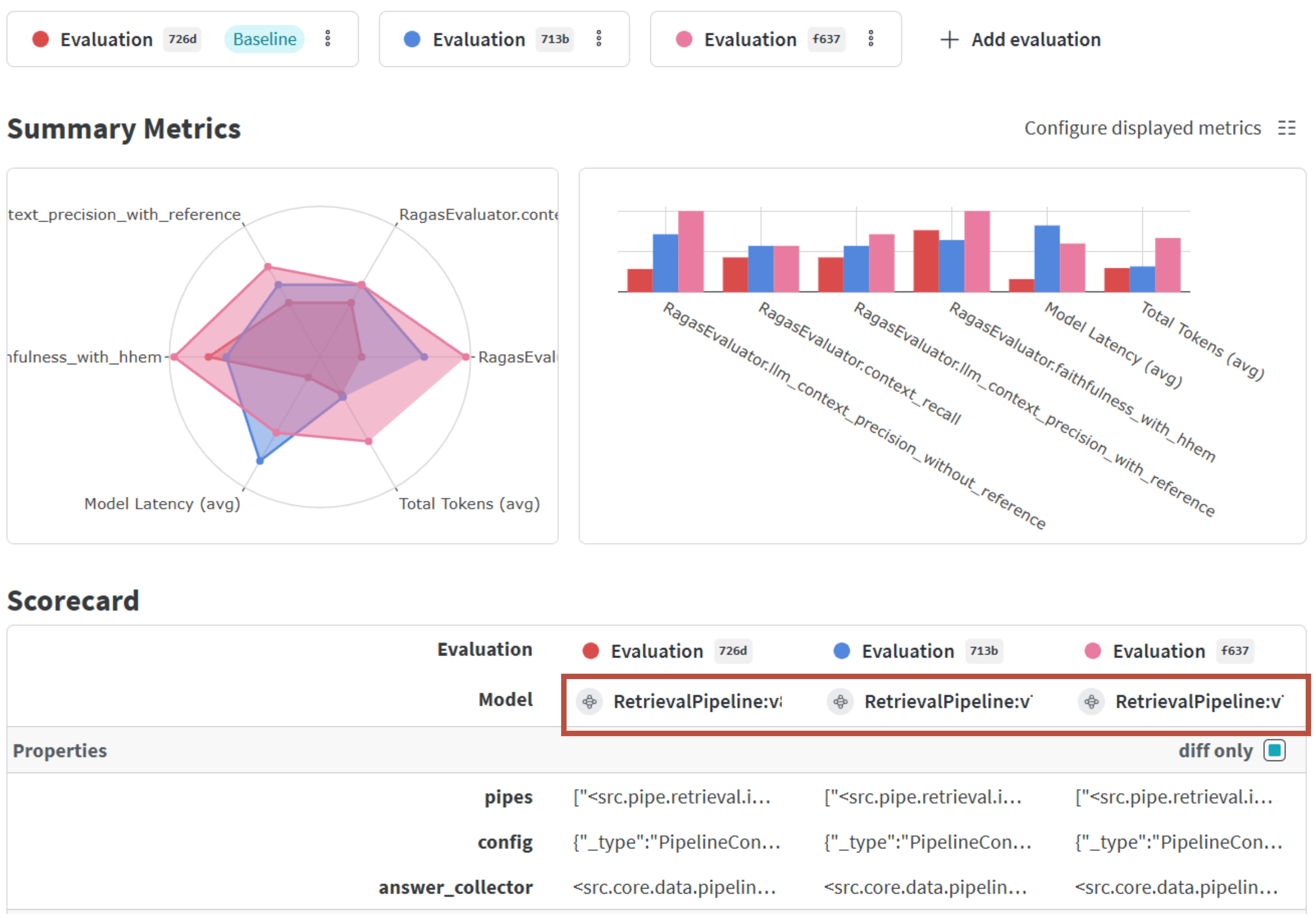

● Evaluation 726d Baseline
● Evaluation 713b
● Evaluation f637
+ Add evaluation
Summary Metrics
Configure displayed metrics
text_precision_with_reference
RagasEvaluator.conte
fulness_with_hhem
RagasEval
Model Latency (avg)
Total Tokens (avg)
RagasEvaluator.llm_context_precision_without_reference
RagasEvaluator.context_recall
RagasEvaluator.llm_context_precision_with_reference
RagasEvaluator.faithfulness_with_hhem
Model Latency (avg)
Total Tokens (avg)

Scorecard

Evaluation	● Evaluation 726d	● Evaluation 713b	● Evaluation f637
Model	RetrievalPipeline:v	RetrievalPipeline:v	RetrievalPipeline:v
Properties			diff only
pipes	["<src.pipe.retrieval.i…	["<src.pipe.retrieval.i…	["<src.pipe.retrieval.i…
config	{"_type":"PipelineCon…	{"_type":"PipelineCon…	{"_type":"PipelineCon…
answer_collector	<src.core.data.pipelin…	<src.core.data.pipelin…	<src.core.data.pipelin…

User Guide for the Code Review: Experimentation

1. Each retriever has one color assigned to it. To find out which retriever has which color. Press on the "Model" color.



Evaluation 726d Baseline | Evaluation 713b | Evaluation f637 | + Add evaluation

Summary Metrics

Configure displayed metrics

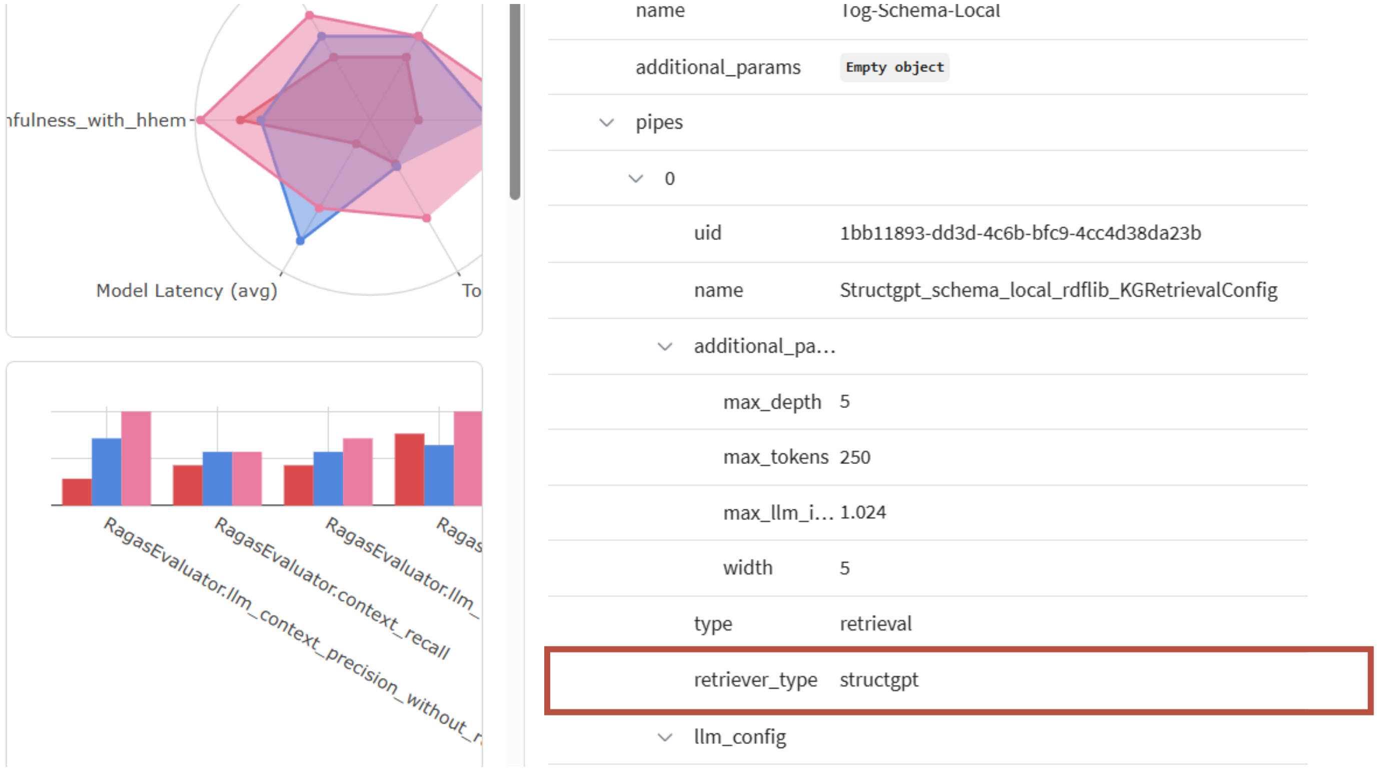
text_precision_with_reference
RagasEvaluator.conte
nfulness_with_hhem
RagasEval
Model Latency (avg)
Total Tokens (avg)
RagasEvaluator.llm_context_precision_without_reference
RagasEvaluator.context_recall
RagasEvaluator.llm_context_precision_with_reference
RagasEvaluator.faithfulness_with_hhem
Model Latency (avg)
Total Tokens (avg)

Scorecard

Evaluation	● Evaluation 726d	● Evaluation 713b	● Evaluation f637
Model	RetrievalPipeline:v	RetrievalPipeline:v	RetrievalPipeline:v
Properties			diff only
pipes	["<src.pipe.retrieval.i…	["<src.pipe.retrieval.i…	["<src.pipe.retrieval.i…
config	{"_type":"PipelineCon…	{"_type":"PipelineCon…	{"_type":"PipelineCon…
answer_collector	<src.core.data.pipelin…	<src.core.data.pipelin…	<src.core.data.pipelin…

User Guide for the Code Review: Experimentation

1. A sidebar should open
2. Look for the key "retriever_type" this indicates the retriever



ıfulness_with_hhem
Model Latency (avg)
To
RagasEvaluator.llm_context_precision_without_r
RagasEvaluator.context_recall
RagasEvaluator.llm_
Ragas

name	Tog-Schema-Local
additional_params	Empty object
pipes	
0	
uid	1bb11893-dd3d-4c6b-bfc9-4cc4d38da23b
name	Structgpt_schema_local_rdflib_KGRetrievalConfig
additional_pa...	
max_depth	5
max_tokens	250
max_llm_i...	1.024
width	5
type	retrieval
retriever_type	structgpt
llm_config	