CARLETON UNIVERSITY
APRIL 17, 2024

COURSE-KG

Generating knowledge graphs from course materials

WRITTEN BY SHRISH MOHAPATRA
SUPERVISED BY AVA MCKENNEY
COMP 4905 HONORS PROJECT

ABSTRACT

Traditionally, course materials are presented in a linear manner, usually through lectures conducted across the duration of a semester. This method of delivery can hinder students' understanding of advanced topics as it struggles to facilitate connections across different lectures. To address these issues, course-KG provides an automated system which processes lecture transcripts and organizes key information into a knowledge graph (KG). This medium provides students a contextually rich representation of course materials by capturing the semantic relationships between concepts. This is accomplished by leveraging state of the art advancements in Natural Language Processing (NLP) and Large Language Models (LLMs). Through the application of various pre/post processing tasks including text splitting, summarization, and multi-prompting, course-KG's modular framework is able to produce reasonable knowledge graphs representative of course content. The platform also provides an intuitive interface where users can interact with generated KGs and make any necessary modifications. Despite the promising results however, issues such as hallucinations, bipartite subgraphs, and extended task completion time, compromise the quality of KGs produced. Future enhancements such as overlapped text splitting, LLMs with larger context windows, and course-specific prompting, can address these weaknesses and enhance this system's KG generation capabilities.

ACKNOWLEDGEMENTS

I would like to thank Dr. Ava McKenney for her support and feedback during the development of course-KG. I was provided various lecture transcripts and materials to assist in my research of LLM-generated knowledge graphs, as well as useful feedback to improve the KG generation process.

TABLE OF CONTENTS

01 INTRODUCTION

The purpose of this project is to provide an alternative method of structuring course material using knowledge graphs [1]. This chapter will discuss the motivations for doing so, and the primary objectives this project aims to accomplish.

01-1 motivation

Traditionally, course materials are provided in a linear manner throughout the duration of the semester. For instance, an instructor may consider partitioning course content over a series of weekly lectures. While this process provides a structured approach, it suffers when trying to facilitate higher-level understanding of the relationships between core concepts.

For instance, consider an introductory programming course, such as COMP 1405 delivered at Carleton University [2]. During the early lectures of the semester, students will likely be learning fundamentals such as variables, looping structures, and lists. However, throughout the latter lectures, more advanced topics such as sorting algorithms and runtime complexity will be introduced. Many of these advanced topics require understanding of former topics. For example, sorting algorithms are only beneficial when one understands common looping structures. Zubaidah's research regarding critical thinking for students suggests that forming connections between concepts facilitates higher order thinking and enables students to achieve mastery in a subject [3].

01-2 objectives

The primary objective is to build a system which encourages students to review the connections between concepts introduced from class. However, this is currently not possible with existing education platforms such as Brightspace, which only allow for sequential organization of course materials [4]. As such, the project entails an automated procedure of processing course materials, specifically lecture transcripts, and organizing the important information extracted in a manner which allows students to review connections. The following describes the technical deliverables for the project:

1. Create a pipeline which loads lecture transcripts, processes them through machine learning (ML) models, and stores results in a database.
2. Leverage NLP algorithms to extract core concepts from lectures and capture semantic relationships between them.
3. Develop an interface to allow students to interact with generated knowledge graphs.

This chapter summarizes the motivations for re-organizing course materials to promote critical thinking for students. As mentioned previously, the method of organization used to provide an overview of course concepts and connections is a knowledge graph. The next chapter will discuss this concept in further detail, along with additional primer information to understand how the project aims to execute on the deliverables.

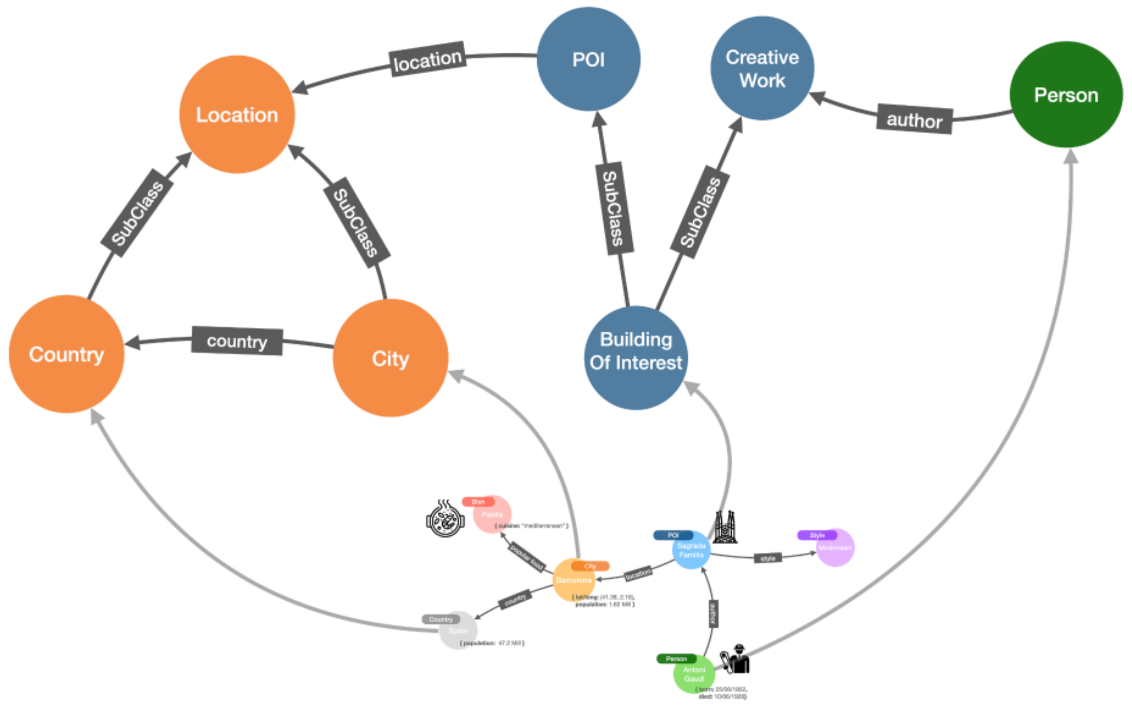
02 BACKGROUND

This chapter provides necessary context required to understand the project's methodology and the technologies used to achieve the objectives described earlier.

02-1 knowledge graphs

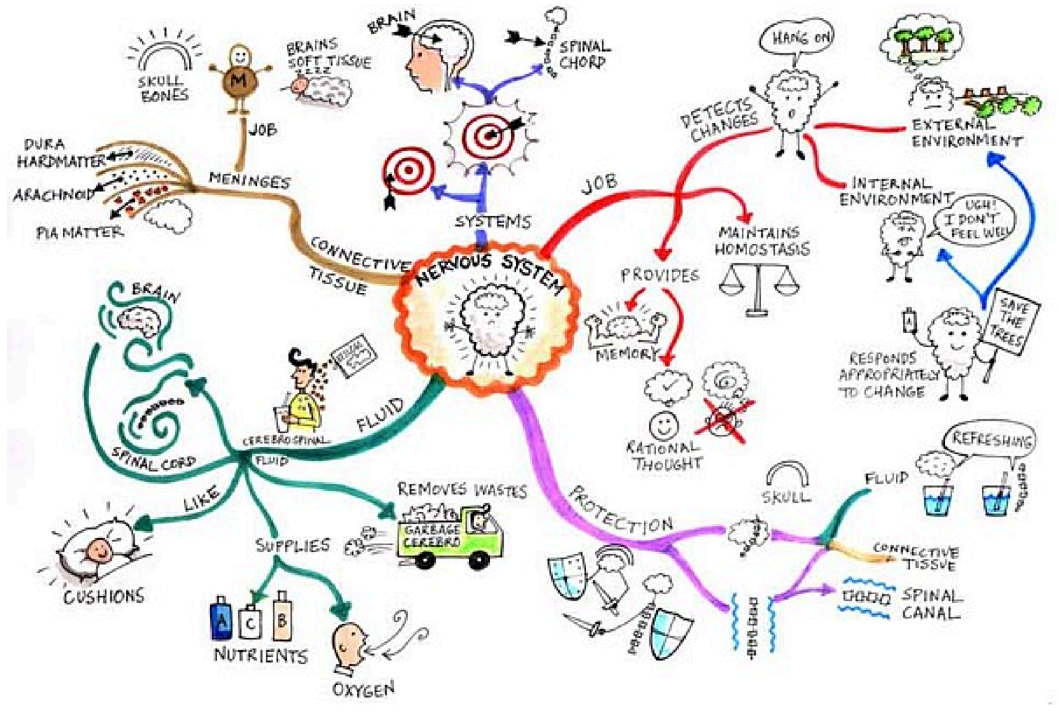
The term "knowledge graph" refers to a network of nodes and edges used to capture the semantic relationships between concepts [1]. The idea is that nodes can represent core concepts, while edges represent the relationships between them. It is often a technical term used in the context of machine learning and artificial intelligence, and there are various papers describing how one can automatically generate them from unstructured text [5]. The following figure demonstrates an example knowledge graph from Neo4j depicting the taxonomy of architecture works [6].

Figure 01: Example knowledge graph of architecture taxonomy



However, the basis of knowledge graphs can be traced from mind maps. Mind maps are a well known technique in the education industry as they are often used to visually describe a central topic, with subtopics branching out to describe new ideas [7]. The following figure demonstrates how students often use this visual technique to better understand advanced topics in biology, such as the nervous system.

Figure 02: Example mind map used to describe nervous system [7]



There is currently much debate regarding the definition of knowledge graphs and their distinction from mind maps [1]. However, this project aims to harness the benefits of both knowledge graphs and mind maps to provide students a powerful method of learning. By leveraging the intuitive nature of mind maps, and the extensive research already conducted on automated knowledge graph generation [5], course-KG can provide a novel method of delivering course content.

02-2 natural language processing

Natural language processing (NLP) refers to the use of machine learning models to process, interpret, and produce text/speech [8]. While the origins of NLP can be traced back to 1913 as a means to translate text from one language to another, it is currently one of the most popular branches in artificial intelligence due to its versatility in a wide array of applications [9]. While thorough knowledge of NLP is not required, the following terms are important for understanding the project's approach.

Tokenization is the process of converting large sequences of text into smaller atomic units for the machine learning models to interpret [8]. This process allows ML models to efficiently process large text data and identify the core elements within a sequence [9]. There are various tokenization methods including word tokenization, character tokenization, and subword tokenization [10]. The varying methods can improve the accuracy and retention of models, but often at the cost of performance, with smaller tokenization methods yielding longer processing times [10].

Embedding involves representing text data as numerical vectors in a high-dimensional space [8]. These vectors represent the semantic meaning behind the words in a manner which can be easily processed by machine learning models. By using vectors, calculations such as cosine similarity can be used to identify the similarity between two words in a computationally efficient manner [11].

Entity-relation (ER) extraction is a common NLP task which involves identifying core entities, such as people, organizations or concepts, from sequences of text and extracting the relationships between them [1]. This task underlies the basis for knowledge graph generation. ER tasks are typically accomplished by leveraging tokenization, embeddings, and rule-based extraction [5].

02-3 large language models

Large language models (LLMs) are machine learning models which are capable of a wide variety of NLP tasks through rigorous training on vast amounts of data [12]. The release of OpenAI's ChatGPT has led to the popularization of LLMs throughout the AI industry, namely due to its significant improvement across NLP tasks. Despite their effectiveness in numerous settings, the following aspects must be considered when using LLMs in production settings.

Significant computation costs. Many LLMs used today are notoriously expensive to run, with OpenAI's flagship GPT-4 model having an estimated 100 trillion parameters [13]. These models typically can only be run on dedicated servers with powerful graphics hardware [12]. However, the recent rise in open source models aims to mitigate this barrier, with Google's recent Gemma models starting at only 2 billion parameters [14]. At the cost of performance, these smaller models can be run locally on most commercial GPUs.

Context window refers to the amount of tokens an LLM can process at a given time [15]. Oftentimes this is referred to as the LLM's "memory", and is what allows these models to "remember" information provided earlier. Typically, models with less parameters have smaller context windows as they are not capable of tracking large sequences in memory. For example, Google's Gemma 2B and 7B model have context windows of 8 thousand tokens [14], while GPT-4 has a limit of 32 thousand [16].

Hallucination is a common issue with LLMs where the model produces incorrect and sometimes misleading information [12]. This issue is particularly tricky to handle, since oftentimes the model's output can appear reasonable with its use of natural human language. This topic is currently being investigated as researchers aim to implement mechanisms to reduce the occurrence of these inaccuracies.

This chapter provides a comprehensive understanding of the concepts used in this project. Course-KG uses LLMs to perform the NLP task of ER extraction for creating knowledge graphs based on course content.

03 METHODOLOGY

Building upon the background information provided in the previous chapter, this section will discuss the core methodologies used to generate knowledge graphs from unstructured text.

03-1 dataset

The dataset used for this project included lecture transcripts from a variety of computer science courses, provided by my supervisor, Ava McKenney. The following table summarizes key attributes of the dataset:

Figure 03: Summary of lecture transcript dataset

COURSE ID	COURSE NAME	TRANSCRIPTS
COMP1405-F19	Introduction to Computer Science I	25
COMP1405-F23	Introduction to Computer Science I	17
COMP1406-F23	Introduction to Computer Science II	17
COMP1406-W23	Introduction to Computer Science II	16
COMP2406-W20	Fundamentals of Web Applications	18
COMP4601-F23	Intelligent Web-based Information Systems	12

Transcripts of human conversation are traditionally difficult to process for multiple reasons. The first is that live transcription services can often make mistakes when converting audio to text, with accuracy depending on microphone quality, speaker accents, and background noise [17]. Additionally, lecture recordings tend to include additional information that may not be relevant to the general course. For instance, lecturers will often include additional remarks and tangents to keep students engaged, but will not necessarily need to be included in the knowledge graph.

03-2 LLM prompting

Due to the nuances with transcript data, the project takes advantage of LLMs and their ability to perform complex reasoning tasks. Specifically, the Gemma 2B and 7B models were used since they could be run locally [14]. As opposed to traditional ER extraction models which struggle parsing large sequences of text [18], LLMs are equipped with the resources needed to process human conversation data and extract relevant information.

This is accomplished by "prompting" the LLM, in which the user provides a set of instructions written in natural language to the model [19]. The LLM then processes these instructions and generates a text-based response. In general, most LLMs do not require a specific format when processing instructions. For instance, a user can prompt the model "what is the difference between nuclear fusion and fission?" and receive an LLM response aimed to address the question. The following prompt was used to request a JSON formatted knowledge graph based on a lecture transcript extracted from the dataset.

Figure 04: Example LLM prompt to generate KG

PROMPT
Instruction +
Transcript

Given the following transcript from a university lecture, list the nodes and edges in JSON format that form a knowledge graph based on key topics.
All right. So our topic this week will be about evaluating recommender systems. So over the past two weeks we've looked at two different approaches to recommendations. So we looked at the user based nearest neighbor and item based nearest neighbor...

LLM RESPONSE
Explanation +
JSON encoding

Certainly! Here's the JSON representation of the knowledge graph based on the provided lecture transcript:

```
{
 "nodes": [
  {
   "id": "Recommender Systems",
   "type": "Topic"
  },
  {
   "id": "User-based Nearest Neighbor",
   "type": "Approach"
  },
...
```

While this approach appears relatively simple, the effectiveness of LLMs can often result in useful and accurate responses. However, many of the issues mentioned in the *Background* chapter regarding LLMs result in the need for pre-processing and external memory management. These methods will be discussed further.

03-3 summarizing transcripts

Despite various modifications to the prompt instructions, initial tests suggested that the Gemma model had difficulties processing large amounts of transcript data, and distinguishing between key concepts and irrelevant remarks. These hallucinations resulted in vast knowledge graphs containing many unnecessary nodes.

To address this challenge, a summarization pre-processing stage was implemented. The motivation was to filter out irrelevant remarks from the transcript and extract key concepts. Summarization is another common NLP task with various ML models available. However, for simplicity, this project simply prompted the Gemma LLM with summarization instructions, before prompting the model again for KG generation. Despite the simplicity of the approach, the *Results* chapter reveals that this multi-stage-prompting technique resulted in significantly less hallucinations. See *Appendix A* for detailed prompt templates.

03-4 context window management

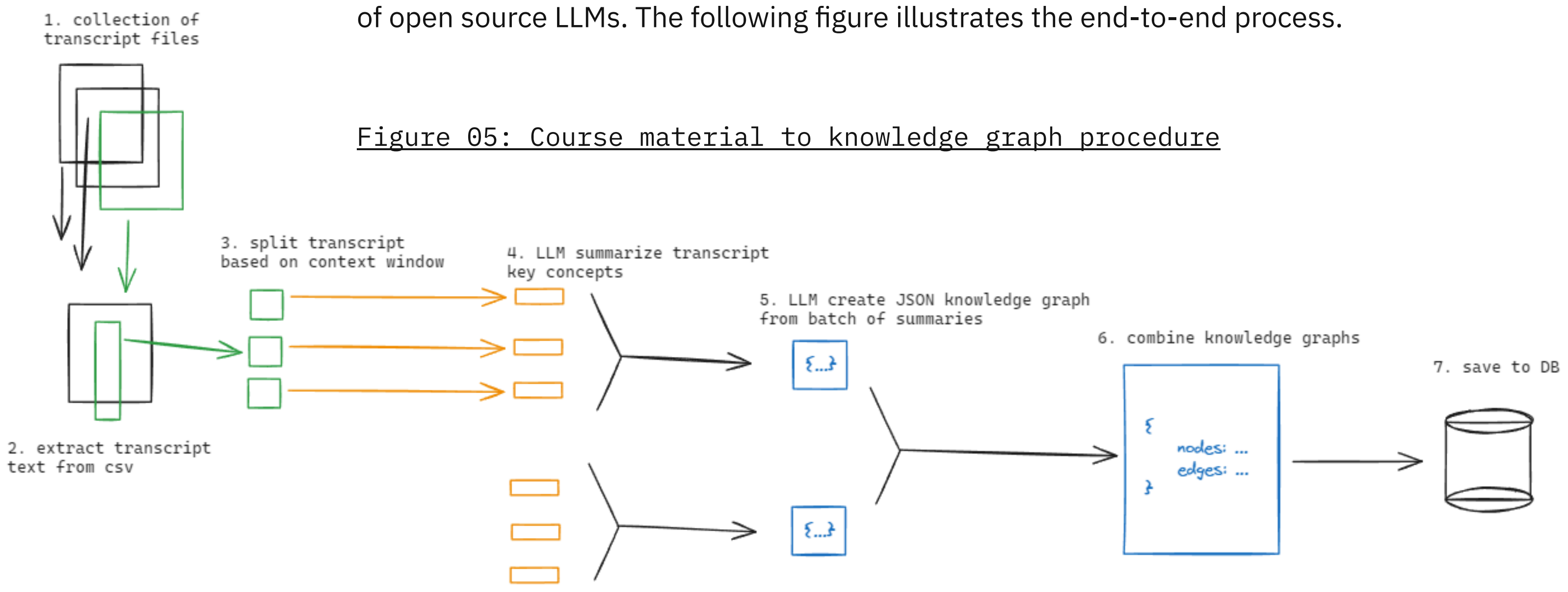
One of the limitations of LLMs mentioned in the *Background* chapter was the context window, the number of tokens an LLM can process at a time. Since the total length of a lecture transcript would exceed Gemma's 8K token limit, the need for dividing the transcript into smaller parts seemed necessary for the LLM to process. This "division of labor" is not a trivial task, since simply splitting text at an arbitrary point can result in the LLM receiving incoherent transcript snippets. The project's methodology to address this was to split transcripts by complete sentences, and incrementally adding portions of the transcript to an

LLM's prompt until a max token limit was reached. The *Results* chapter will further describe the implications of this approach. This form of text-splitting is not the only approach, with various NLP research publications describing alternative methods to ensure text fragments capture sufficient semantic information [20].

Although splitting LLM prompts helps address the context window limitations, a procedure is required to aggregate and maintain the responses produced. This process is referred to as "external memory management", since it requires the development of data structures which store LLM responses separate from the model's transformer neural network [21]. The project leverages arrays to track sequences of LLM responses and dictionaries/hashmaps to maintain the nodes and edges of a knowledge graph, the details of which will be explained in the *Software* chapter.

This chapter details the core methodologies used to generate knowledge graphs from lecture transcripts. LLM prompting allows for natural language instructions to generate KGs from transcripts. The use of pre/post processing tasks such as summarization and external memory management help address the limitations of open source LLMs. The following figure illustrates the end-to-end process.

Figure 05: Course material to knowledge graph procedure


1. collection of transcript files
2. extract transcript text from csv
3. split transcript based on context window
4. LLM summarize transcript key concepts
5. LLM create JSON knowledge graph from batch of summaries
{...}
{...}
6. combine knowledge graphs
{ nodes: ... edges: ... }
7. save to DB

04 SOFTWARE

While the *Methodologies* chapter outlined the key approaches used to generate KGs from course materials, this chapter will further discuss the technical details of the implementation.

04-1 ollama

Ollama is an open source tool used to run large language models, such as Gemma, on local hardware [22]. The tool provides a REST API client which can be used to prompt LLMs and receive model responses. The benefit of using a tool like Ollama is that it enables easy configuration and setup of new LLMs, allowing the project to leverage the latest models released regularly.

However, a key requirement must be met to run LLMs locally. As mentioned in the *Background* chapter, LLMs require GPU hardware to efficiently perform calculations needed when generating responses. Ollama supports CUDA, a computing API which allows developers to leverage NVIDIA GPUs for computation tasks [23]. The project's experiments were run on a NVIDIA Titan V to take advantage of CUDA and run LLMs locally [24].

04-2 text2kg

The *Methodology* chapter described a sequence of tasks needed to be performed to transform lecture transcripts into a JSON formatted knowledge graph. `text2kg` is a Python library created during the project which allows for the development and execution of these transformation tasks. Following the Pipeline design pattern [25], the library's modular design allows developers to easily create/modify text-to-KG pipelines and experiment with different mechanisms for pre/post processing data.

Every task in `text2kg` is implemented as a Python class which must implement a `process()` function. This function will handle the processing of data, such as summarizing transcripts provided within function arguments, and returning the LLM summaries as the return value. This object oriented approach allows for tasks to be configured via constructor parameters and supports inheritance for more advanced tasks. For instance, an abstract `LLMTask` was developed to handle Ollama REST API client connections and configuration settings. Tasks such as `CreateKnowledgdeGraphs` can inherit from `LLMTask`, taking advantage of the Ollama connection, and simply contain code related to the creation and validation of LLM-generated KGs.

The following table provides a summary of some of the task components developed and their motivation.

Figure 06: Summary of text2kg tasks

TASK CLASS NAME	**PURPOSE**
`LoadFolder`	Load transcript files from a specific folder. Can configure file extensions to extract (ex `.mp4.csv`)
`ExtractTranscripts`	Extract transcript text from Zoom-generated CSV files using `pandas` library.
`SplitTranscripts`	Split transcript text into smaller snippets based on `max_tokens`. Used to address context window limits.
`SummarizeTranscripts`	Invoke LLM to summarize transcripts. Can configure which model to use and the instruction prompt.
`CreateKnowledgeGraphs`	Invoke LLM to create KGs. Can configure which model to use and the instruction prompt.
`CombineKGs`	Rule-based approach to aggregate multiple LLM KGs into one KG per course. Involves use of stemming and Levenshtein distance to combine similar terms [42].
`FixKnowledgeGraphs`	Invoke "critic" LLM to fix generated KGs, adding/removing nodes/edges when necessary.
`StoreInDB`	Store generated KGs in NoSQL database.

While many of the tasks had trivial implementations, the process of producing valid JSON proved to be the most challenging. `CreateKnowledgeGraphs` was used to prompt an LLM to generate knowledge graphs based on transcript summaries. Despite the various pre-processing stages implemented however, there were various instances in which the LLM response contained invalid JSON encodings of KGs. To address this challenge, a mechanism was needed to verify the validity of the JSON output, and then reprompt the LLM again in case of failure. While this was a rudimentary approach, the *Results* chapter will illustrate the impact this mechanism had on performance.

To execute a series of `text2kg` tasks, the library provides a `Pipeline` object. Developers can configure which tasks they deem necessary for their pipeline, and specify the order in which tasks should be executed. The execution of a pipeline is delegated to a `PipelineEngine`, responsible for running the sequence of tasks and passing the required data between them. The following figure depicts how `text2kg` can be used. The `AirflowEngine` will be described in the next section.

Figure 07: Example text2kg pipeline

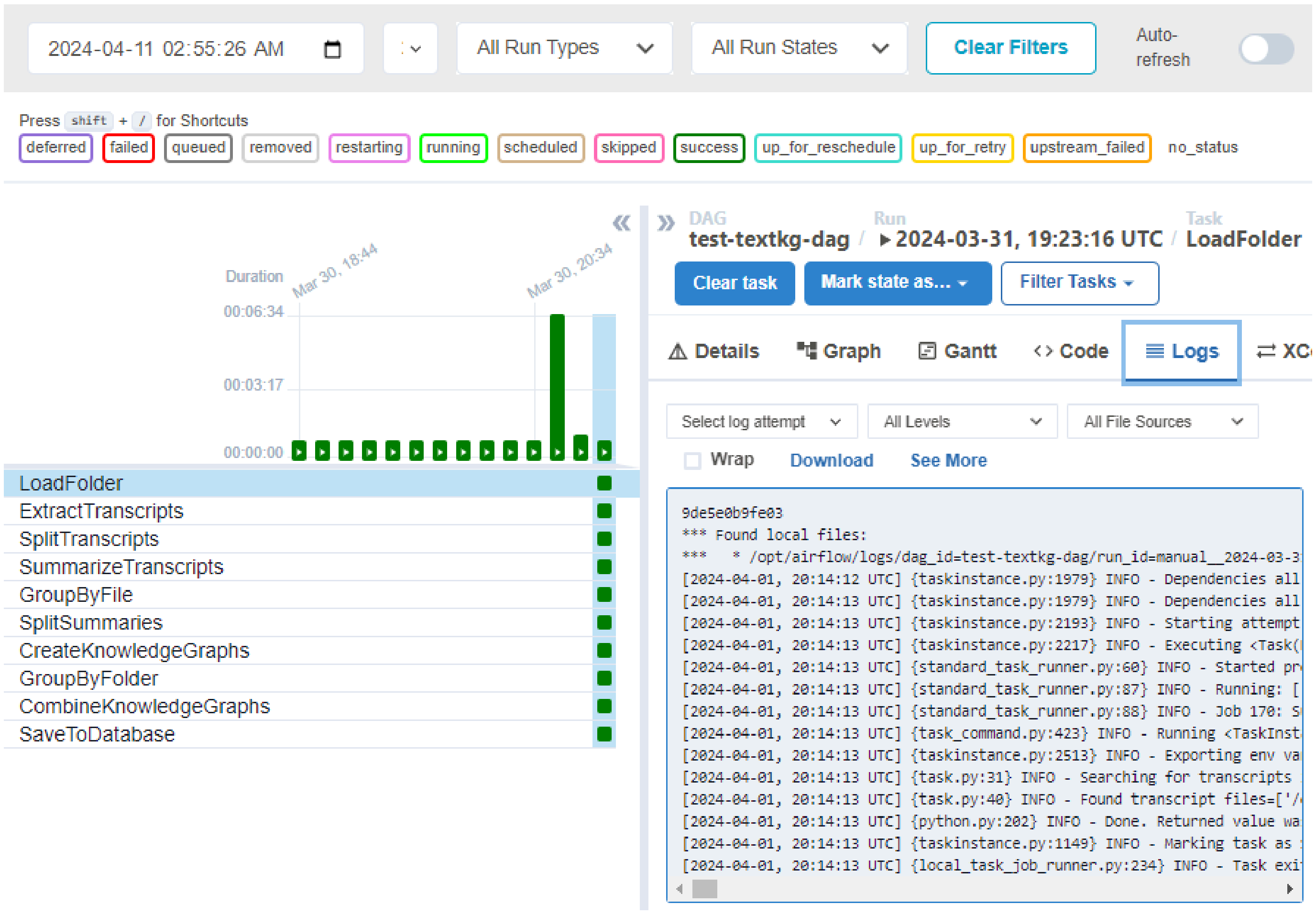
```
import text2kg.task as t2k
from text2kg.core import Pipeline, AirflowEngine

pipeline = Pipeline(
    tasks=[
        t2k.LoadFolder(),
        t2k.ExtractTranscripts(),
        t2k.SplitTranscripts(max_tokens=1000),
        t2k.SummarizeTranscripts(),
        t2k.GroupByFile(),
        t2k.SplitSummaries(max_tokens=700),
        t2k.CreateKnowledgeGraphs(),
        t2k.GroupByFolder(),
        t2k.CombineKnowledgeGraphs()
    ],
    pipeline_engine=AirflowEngine(...)
)
pipeline.run()
```

04-3 airflow

Apache Airflow is an open-source orchestration tool based on Python for executing data pipelines [26]. This project uses Airflow to handle the execution of `text2kg` pipelines, hence the development of an `AirflowEngine`. The tool provides useful observability utilities for logging task status and monitoring inter-pipeline communications. The following figure showcases a few of Airflow's observability features.

Figure 08: Airflow observability utilities with text2kg pipeline



```
9de5e0b9fe03
*** Found local files:
***   * /opt/airflow/logs/dag_id=test-textkg-dag/run_id=manual__2024-03-3
[2024-04-01, 20:14:12 UTC] {taskinstance.py:1979} INFO - Dependencies all
[2024-04-01, 20:14:13 UTC] {taskinstance.py:1979} INFO - Dependencies all
[2024-04-01, 20:14:13 UTC] {taskinstance.py:2193} INFO - Starting attempt
[2024-04-01, 20:14:13 UTC] {taskinstance.py:2217} INFO - Executing <Task(
[2024-04-01, 20:14:13 UTC] {standard_task_runner.py:60} INFO - Started pr
[2024-04-01, 20:14:13 UTC] {standard_task_runner.py:87} INFO - Running: [
[2024-04-01, 20:14:13 UTC] {standard_task_runner.py:88} INFO - Job 170: S
[2024-04-01, 20:14:13 UTC] {task_command.py:423} INFO - Running <TaskInst
[2024-04-01, 20:14:13 UTC] {taskinstance.py:2513} INFO - Exporting env va
[2024-04-01, 20:14:13 UTC] {task.py:31} INFO - Searching for transcripts
[2024-04-01, 20:14:13 UTC] {task.py:40} INFO - Found transcript files=['/
[2024-04-01, 20:14:13 UTC] {python.py:202} INFO - Done. Returned value wa
[2024-04-01, 20:14:13 UTC] {taskinstance.py:1149} INFO - Marking task as
[2024-04-01, 20:14:13 UTC] {local_task_job_runner.py:234} INFO - Task exi
```

Airflow is not the only tool of its kind, with various modern alternatives in data orchestration such as Prefect [27] and Dagster [28]. This is why `text2kg` abstracts the execution of a pipeline, allowing for compatibility with any observability tool on the market.

04-4 mongodb

MongoDB is a popular NoSQL database which uses JSON-like formats for storing data [29]. NoSQL databases are optimal for settings with semi-structured data that does not fit well with traditional SQL-based relational databases [30]. Since the generated knowledge graphs are represented as JSON nodes and edges, MongoDB provides a strong solution.

KGs are stored in MongoDB as a single document each. This is because the application allows users to view the entire knowledge graph through a single interface. Additionally, knowledge graphs are scoped by a “project”. For instance, “COMP 4601 F23” could be considered a project with a knowledge graph representing the specific term’s delivery of course’s concepts. This allows for the creation of multiple knowledge graphs per course, each differing in terms of general text-to-KG approaches and LLM models used.

04-5 fast API

FastAPI is a Python web framework for rapidly building APIs, including features such as automatic Swagger API documentation and data validation [31]. This framework is used for retrieving and modifying KGs from MongoDB via REST API endpoints. A detailed specification of the endpoints available can be found in *Appendix B*. This API can then be used across any web / mobile application. Although many modern database providers offer SDKs to interface with their data, using a separate API service allows for better security [32]. This is because database credentials are not being stored directly on client interfaces, and the API is able to provide proper authentication/authorization functionality.

04-6 react

React is a JavaScript library used for building web applications [33]. A single-page application (SPA) has been developed to allow users to interact with generated KGs in a visual manner. Using the `react-force-graph` library developed by Asturiano [34], simulations of graph physics allow for visually stunning representations of knowledge graphs. Detailed overview of the web application's design and features can be found in *Appendix C*.

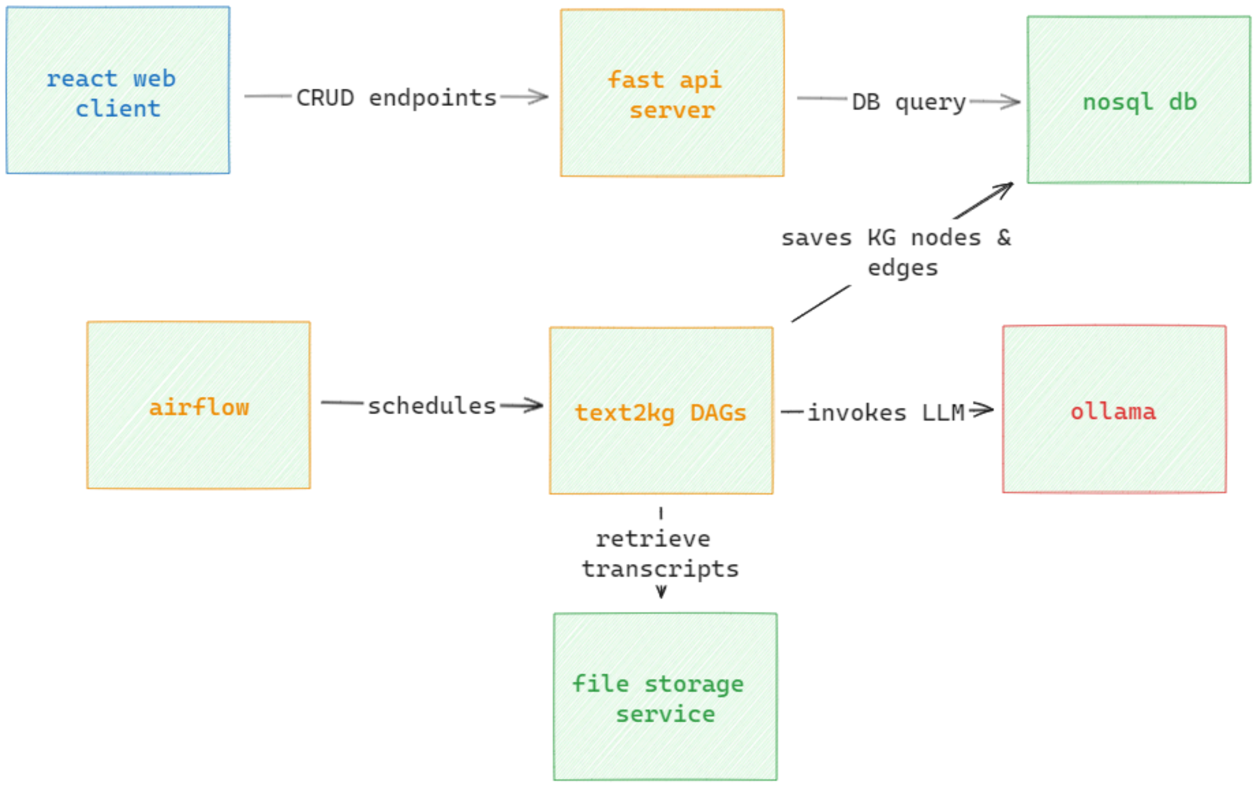
04-7 docker

Docker is a tool for deploying applications within containerized environments [35]. Due to the myriad of services required during the KG generation process, using Docker and Docker Compose proved to be the most effective.

Docker allows for isolated containers for each of the services. Each container can have their own set of dependencies, for example, a FastAPI container could require various Python libraries, whereas a React container may only need Node.js. Using the Docker NVIDIA Container toolkit, certain containers can have access to the host PC's GPU [36]. The Ollama container, for example, uses this toolkit to run LLMs efficiently on local hardware. Docker Compose allows for simple configuration of all services within a single file, allowing for the execution of the entire stack through a few commands [37].

This chapter provides detail regarding the various software components necessary for the generation of knowledge graphs from course materials. Ollama is used to run LLMs locally, `text2kg` is used for creating text-to-KG tasks in Python, Airflow is used to execute these pipelines, MongoDB is a NoSQL database for storing generated KGs, Fast API is used to query/modify KGs from the database, React provides a webclient for users to interact with KGs, and Docker provides a unified environment for running all these services together seamlessly. The following figure illustrates the tech stack used.

Figure 09: System design diagram for course-KG



05 RESULTS

This chapter will describe the project's results using the previously mentioned methodologies and software. The results will be divided into two sections, comparing metrics between different pipeline configurations, and a qualitative overview of generated KGs.

05-1 KG generation performance

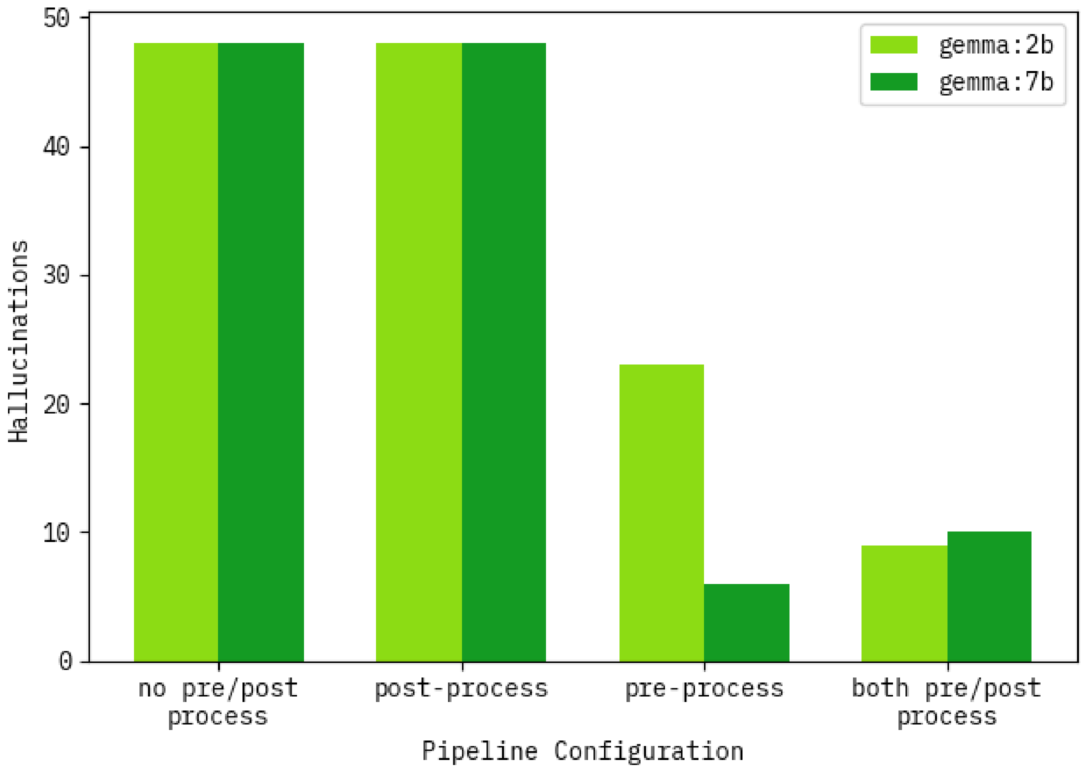
The objective of monitoring KG generation performance with different pipeline configurations is to determine which components play the most significant role, and the nuances of large language models. The following table describes the various configuration settings used during experiments.

Figure 10: KG Pipeline Configurations used for Experiments

LABEL	**DESCRIPTION**
`gemma:2b`	Using open source LLM with 2 billion parameters (~1.7 GB) [14].
`gemma:7b`	Using open source LLM with 7 billion parameters (~5.0 GB) [14].
`no pre/post process`	Pipeline consists of loading transcripts, prompting LLM to create KGs, and using rule-based aggregation methods to create unified KGs per course.
`pre-process`	In addition to the standard pipeline, this adds summarization and text-splitting stages to address context-window limitations.
`post-process`	In addition to the standard pipeline, this adds an additional LLM prompt stage at the end to "fix knowledge graphs". This attempts to improve the quality and remove unnecessary nodes/edges.
`both pre/post process`	Standard pipeline with additional tasks from both `pre-process` and `post-process` configurations.

Each pipeline configuration was tasked with processing lecture transcripts from COMP 4601-F23 for consistency. The course consisted of 12 lecture transcripts for the pipelines to process and generate KGs from.

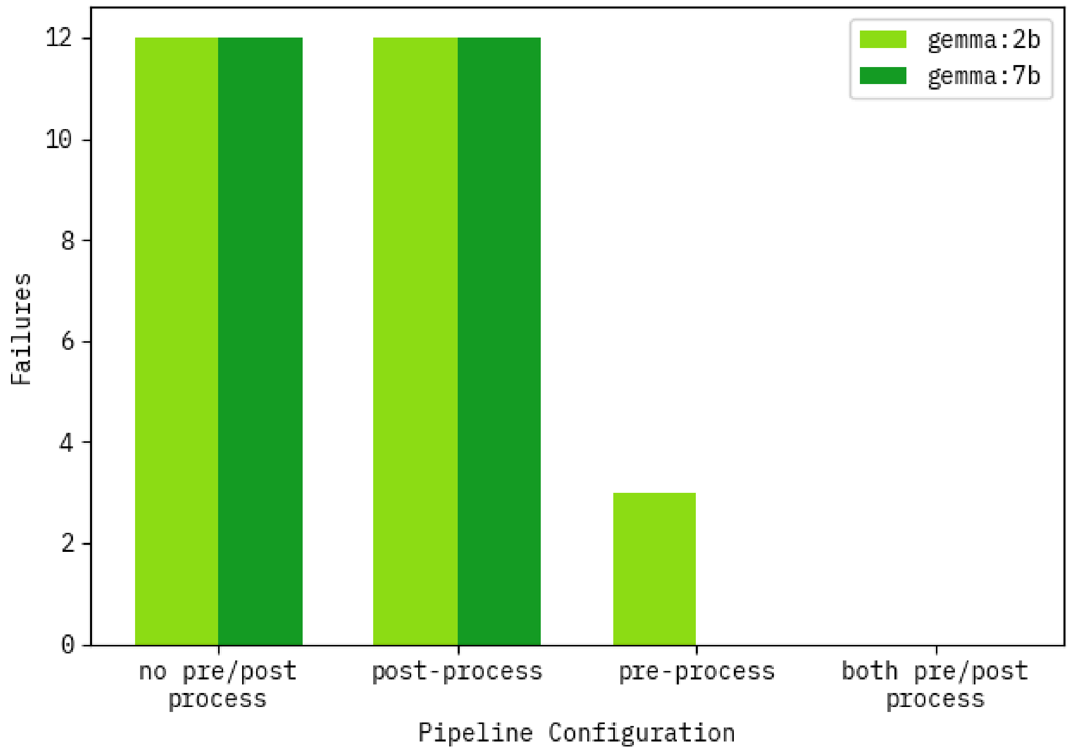
Figure 11: Hallucination Rate



As mentioned in the *Background* chapter, hallucinations are events where the LLM produces incorrect responses. During the KG generation process, a hallucination was documented when invalid KGs were produced. Issues include invalid JSON formatting, inconsistent nodes and edges, and incomplete KG information.

As seen in the figure above, the inclusion of pre-processing, dramatically reduced the hallucination rate. This is likely due to the impact of context window limitations, and how without pre-processing the LLM likely "forgets" earlier instructions. Moreover, the larger 7B model seems to experience less hallucinations than its 2B counterpart when using pre-processing only. This is likely due to the better performance larger models tend to yield. However, when using both processing methods, the hallucination rate of gemma:2b is reduced significantly. This demonstrates the value of adding a “critic-LLM” to help fix inconsistencies with the generated KGs.

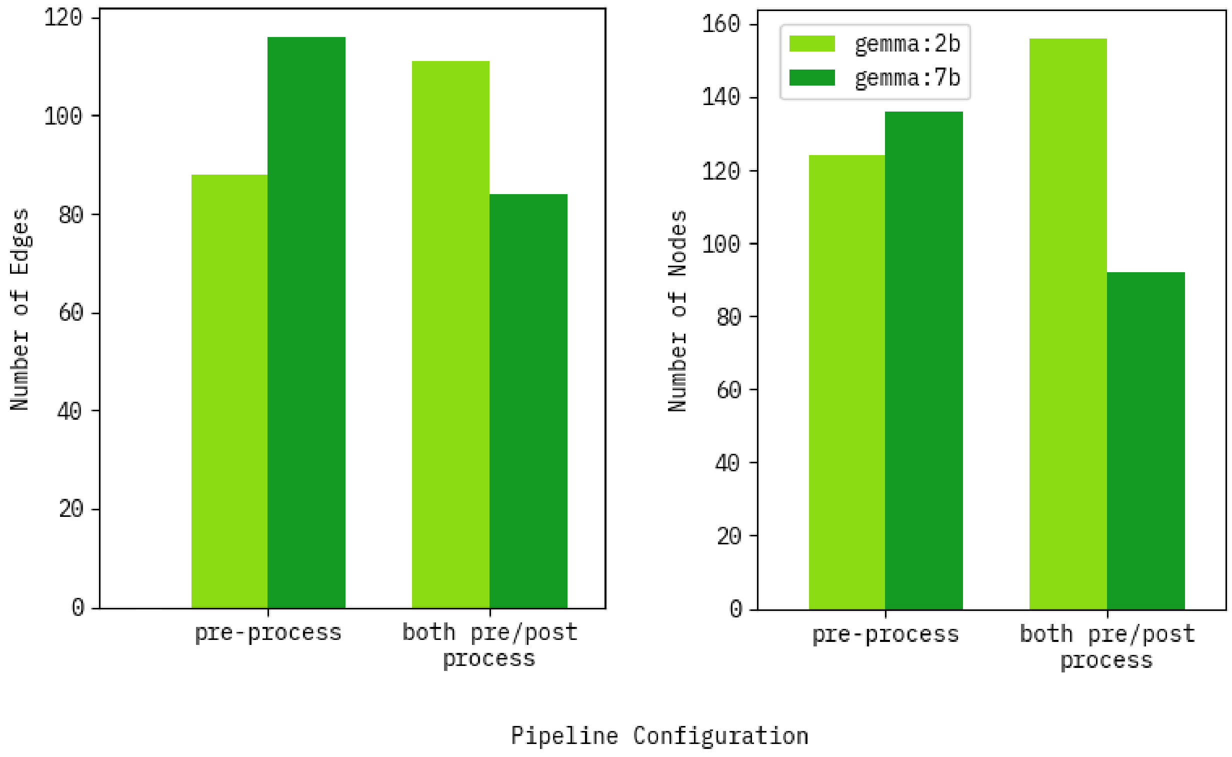
Figure 12: Failure Rate



During the *Software* chapter, a retry mechanism was described such that in the event of an LLM hallucination the pipeline would reprompt the LLM again up to 3 times with the same instructions. If after 3 attempts the LLM failed to produce valid JSON KGs, a failure was recorded.

This figure reinforces the notion that pre-processing is crucial for successful KG generation. The introduction of text splitting and summarization eliminated all failures with the gemma:7b model. Moreover, the inclusion of both text splitting and the LLM critic agent mitigated failures for the smaller gemma:2b model.

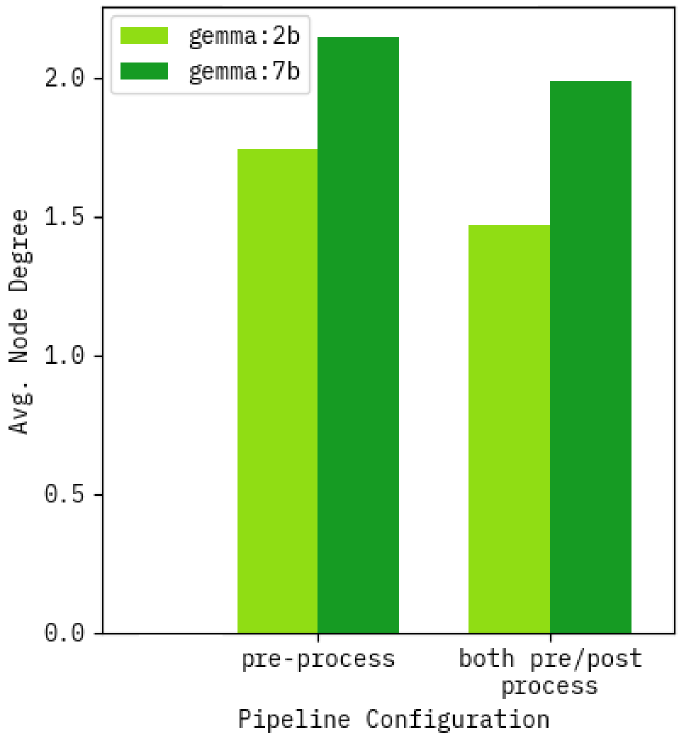
Figure 13: Number of Nodes & Edges



Tracking the number of nodes and edges within generated KGs provides insights regarding the behaviors of pre/post processing components and the impact of LLM parameter sizes. The "post-process" only and "no pre/post process" configurations were excluded from this figure since they did not produce any valid KGs.

This figure suggests that the presence of text splitting pre-processing and LLM-critic post-processing affects the two LLM models differently. With pre-processing stages only, the larger gemma 7B model produced more nodes and edges than the 2B model. However, after combining the critic component, the 7B model reduces the number of nodes and edges, while the 2B model adds more. This behavior is difficult to rationalize, however one reason could be the different training sets the models may have been trained with. Regardless, a KG with more/less nodes/edges does not necessarily indicate higher quality. This metric rather reflects the different model's tendencies when generating KGs.

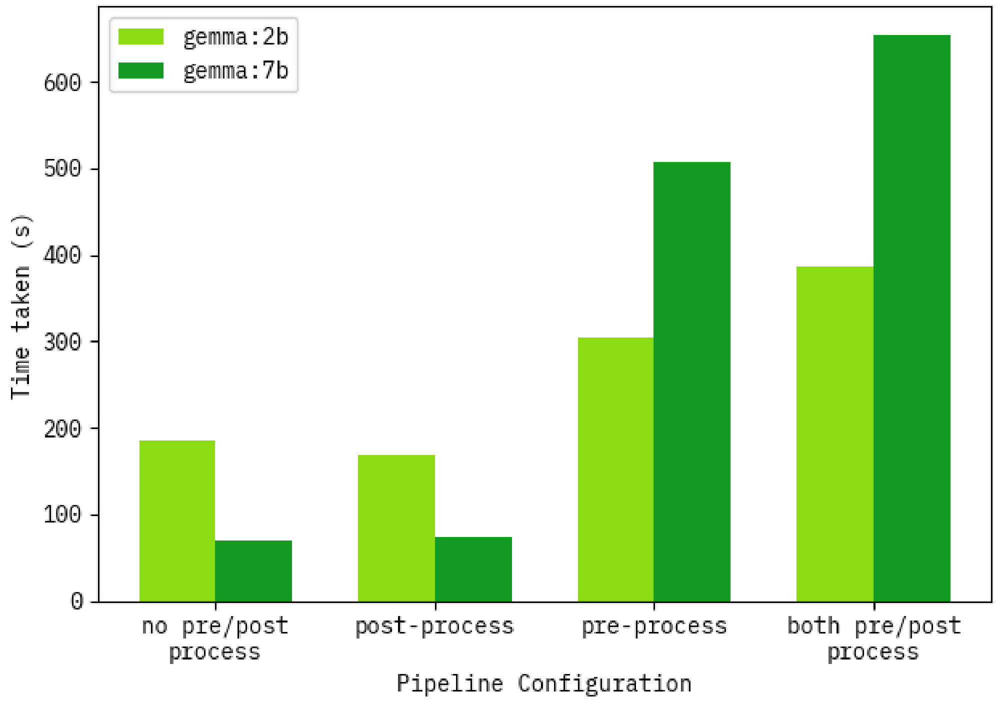
Figure 14: Average Node Degree



In addition to node and edge count, a useful metric to measure is the average node degree. The degree of a node refers to the number of edges leading into/away from the node [38]. In a KG, higher average node degrees could suggest more connections being formed between concepts.

This figure demonstrates that the larger gemma 7B model produces knowledge graphs with higher average node degree, suggesting more connections between the generated concept nodes. This could be indicative of the model's enhanced reasoning capabilities due to its larger number of parameters. Additionally, the smaller 2B model seems to produce KGs with lower node degree when using post processing components. Given that the previous figure illustrated how the 2B model seems to add more nodes during the critic stage, this could mean it is adding new nodes without linking them to existing concepts. Despite these results, this metric cannot truly reflect the quality of the KG produced. For instance, the 7B model could be creating connections between unrelated concepts.

Figure 15: Task Completion Time



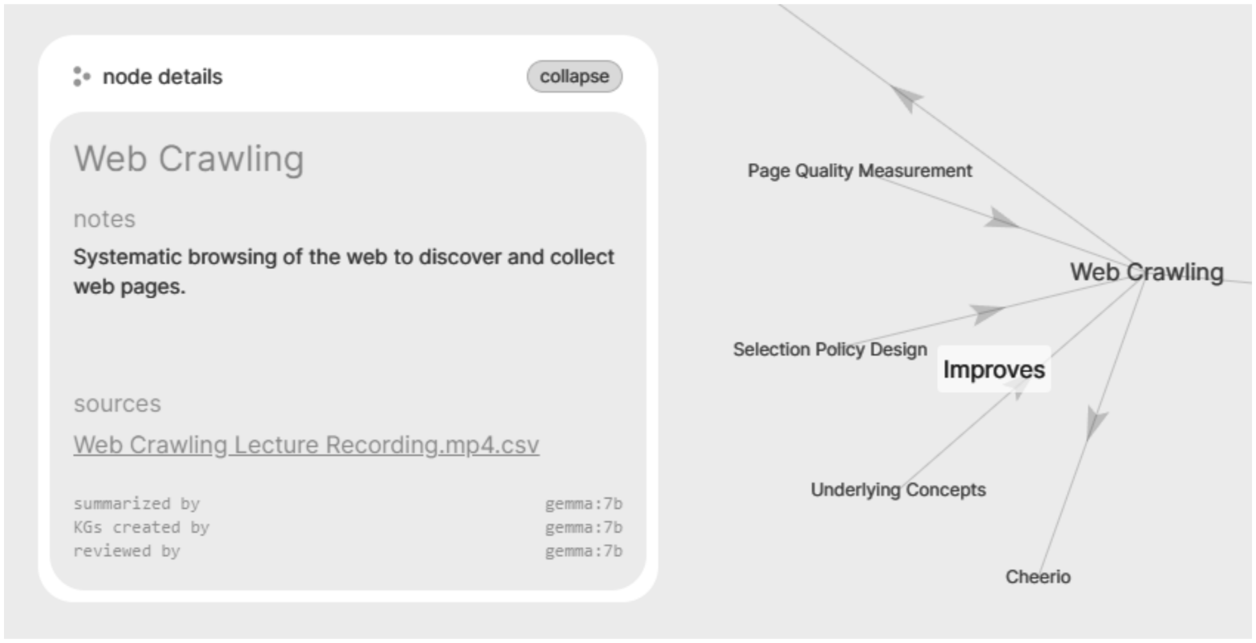
The final metric which was documented included task completion time. This was measured by recording the time between when the pipeline first received transcripts and when KGs were produced. However, the task completion time for the “no pre/post process” and “post-process” only configurations also consider the time spent prompting the LLM to correct mistakes.

As expected, the inclusion of multiple processing stages increases the task completion time. Both the gemma 2B and 7B models experienced a 44% and 86% increase in task completion time respectively when including the text splitting summarization stages. Additionally, the LLM-critic stage increased completion time by 21% and 22% for the 2B and 7B models. The pre-processing stage seems to have a bigger impact on completion time due to the multiple LLM prompting subtasks required to meet context window limitations. Moreover, across all successful runs, the larger 7B model took 40% more time to produce KGs than its 2B counterpart. This suggests a tradeoff between enhanced reasoning found in larger models and task completion time.

05-2 qualitative view of KGs

While the empirical results from the previous section can provide potential indicators for KG quality, the true value of these LLM generated structures can only be determined through qualitative assessments of the graph. The development of a React webclient, as described in the *Software* chapter, reveals insights regarding the generated KGs.

Figure 16: Rational creation of nodes and edges


node details
collapse

Web Crawling

notes

Systematic browsing of the web to discover and collect web pages.

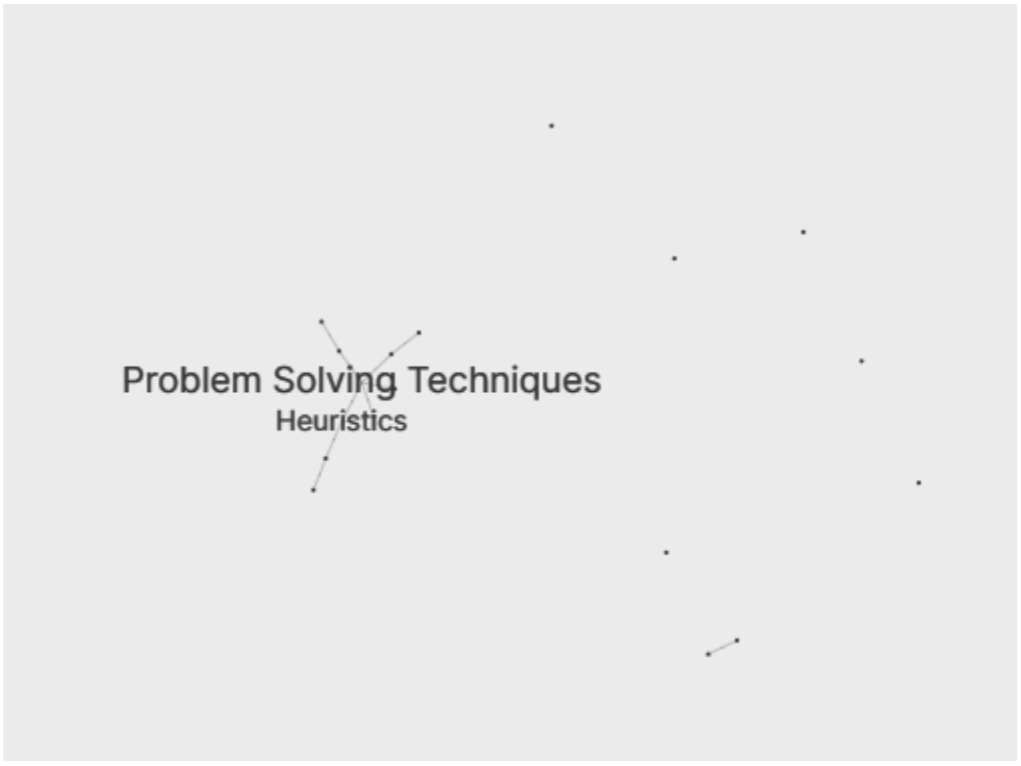
sources

Web Crawling Lecture Recording.mp4.csv

summarized by gemma:7b
KGs created by gemma:7b
reviewed by gemma:7b

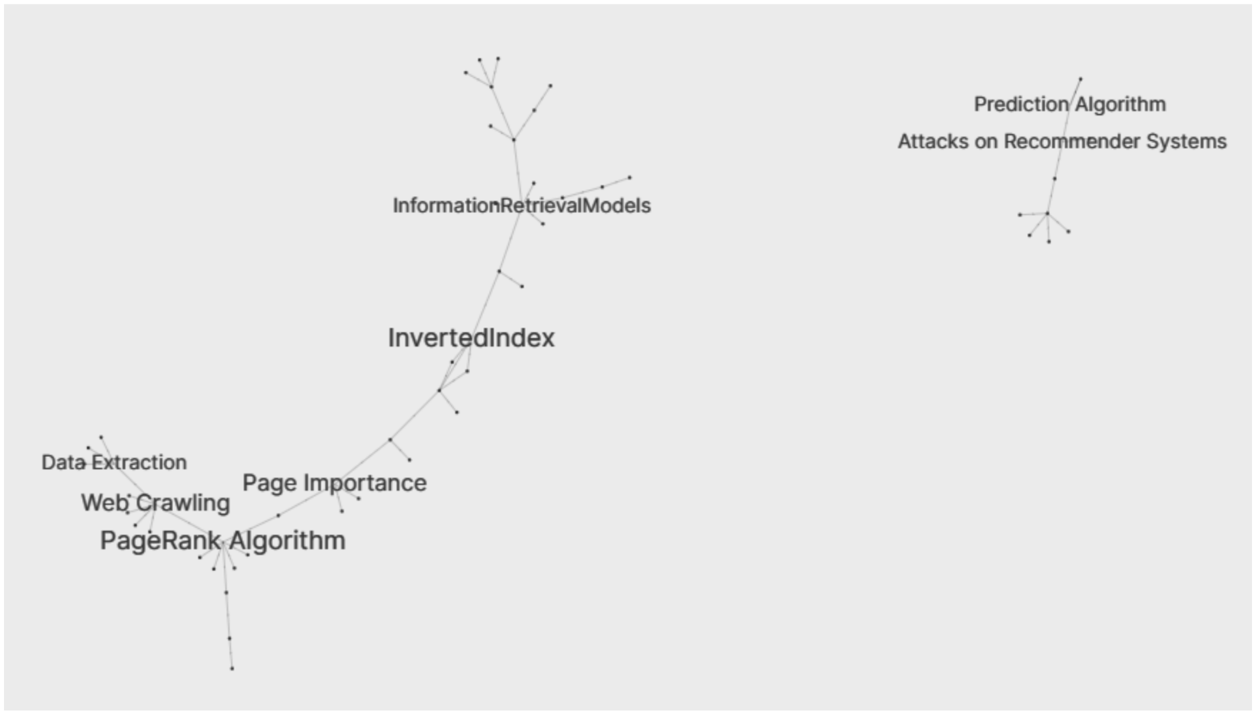
This figure demonstrates the successful creation of nodes and edges from COMP4601, generated by gemma 7B with all pre/post processing stages included. The LLM produced nodes such as “Selection Policy Design” and “Web Crawling”, and created an edge between these concepts; “improves”. By doing so, the KG represents the rational idea that “Selection Policy Design is used to improve Web Crawling procedures”. Moreover, the LLM-produced summary for the “Web Crawling” concept is accurate and representative of the course content.

Figure 17: Various sparse nodes



While Figure 16 suggests the LLM is capable of producing reasonable nodes and edges, Figure 17 demonstrates shortcomings with the KG generation process. The various “dots” in the figure represent isolated nodes with no incoming/outgoing edges. These nodes are likely due to hallucinations during the KG generation process, as the LLM either includes nodes which are not relevant to the course, or forgets to add connections to existing concepts.

Figure 18: Bipartite sub-KGs



An interesting phenomena with the KGs produced by LLMs includes various bipartite sub-knowledge graphs. Figure 18 depicts two distinct sub-KGs with no edges between them. In a true KG representing course concepts, this is very unlikely, since oftentimes majority of course concepts are interconnected. This behavior is likely a consequence of the text-splitting pre-processing used to address context window limitations. Since each LLM prompt includes lecture transcripts which do not overlap with others, the model does not have the opportunity to form larger connections across lectures. Potential solutions to this will be discussed in the *Discussion* chapter.

This chapter summarizes the results of the KG generation process described in chapters prior. It appears that the inclusion of both pre and post processing stages, and the selection of LLMs with larger model parameters, play a significant role in mitigating hallucinations and potentially producing higher quality KGs. Additionally, qualitative observations reveal the LLM's advanced reasoning capabilities with its generation of valid nodes and edges indicative of concepts discussed in the course. However, the creation of isolated nodes and bipartite sub-graphs suggest that the KGs produced are not perfect. The following *Discussion* chapter will discuss the consequences of these results and potential next steps.

06 DISCUSSION

This chapter provides analysis of the results described earlier, as well as some insight regarding the future of LLM development and their applications within the education sector.

06-1 KG analysis of results

Despite the success found with LLMs generating KGs, the lack of performance from smaller models is evident. Rudimentary re-prompt mechanisms were required to address the hallucinations experienced by the Gemma models, and produce JSON output in a reliable manner. Moreover, managing smaller context windows through “external memory management” adds significant overhead and increases task completion time. This behavior is analogous to the mechanism used when computers run out of RAM and begin swapping data to/from the hard drive [40]. If a computer is constantly running out of memory, it can spend the majority of the computation time swapping instead of accomplishing meaningful tasks.

Additionally, many of the successful KGs produced by LLMs contained bipartite subgraphs which had little overlap. This is likely a result of the text-splitting processes and how the LLM is not provided opportunities to connect ideas from separate lectures. A potential solution to this behavior would be to modify the text-splitting process such that there is overlap between transcript fragments. While this may not allow for cross-lecture connections, adding overlap between transcripts does provide the model the ability form deeper connections within concepts introduced during a lecture.

06-2 next steps

While this project seems to suggest that the next step should be adding more pre/post processing stages, rapid LLM development may contradict this. The Browser Company, a startup that has recently launched an AI browser named Arc, shared their difficulties when integrating AI features into their platform [41]. They pointed out that investing in the development of memory-management tools for supporting LLM agents could become futile in the long run, as larger models with bigger context windows may render these tools obsolete [41]. Already during the development of course-KG, the developers of Gemma released Gemini 1.5 Pro with a context window of one million tokens [43]. Fortunately, `text2kg`'s modular design allows flexibility with using other LLM models and adjusting pre/post processing tasks. When using more powerful models, developers can remove unnecessary tasks from the pipeline and likely achieve better performance.

In the context of education applications, course-KG offers promising results for organizing concepts in a non-linear manner. Subsequent development could involve experimenting with different course types to determine KG generation behavior. There are currently differences with knowledge graph structure between first year computer science courses such as COMP 1405 and fourth year COMP 4601. Therefore, it is worth investigating if courses of different academic backgrounds can influence KG generation. Additionally, it would be beneficial to add support for loading course materials in different formats such as PDF, PowerPoint slides, audio, and more. `text2kg`'s pipeline design allows for developers to write file format-specific data loaders which will then be compatible with the rest of the KG-generation pipeline. Moreover, the use of natural language prompts enhances the accessibility of the KG generation process. Professors with domain-knowledge for their course can experiment with different prompt structures to improve KG generation without requiring technical expertise.

06-3 conclusion

Many assumptions regarding open source LLMs suggest that they can only be used as chatbots and require significant fine tuning, a time consuming and resource intensive process of training the model [39], to achieve any real value in a production setting. However, the development of course-KG, in particular the performance of `text2kg` pipelines using Gemma LLMs, reveals otherwise. Through various pre-processing, multi prompting, context window splitting, and post-processing stages, an LLM is able to provide reasonable elements to form a knowledge graph from lecture transcripts. This tool allows for non-linear methods of organizing course content, reflecting the various connections needed to achieve mastery of a subject. Using the webclient, students are able to interact with the LLM generated KGs to review core concepts and recognize relationships between them, facilitating higher order learning. However, limitations with the KG generation process include hallucinations producing invalid JSON and bipartite subgraphs with missing connections. Nevertheless, the framework can be improved by including overlapped transcript splitting, leveraging LLMs with larger context windows, and customizing prompts to align with the specific context of each course.

07 REFERENCES

[1] IBM, “What is a Knowledge Graph?,” *www.ibm.com*, 2024. https://www.ibm.com/topics/knowledge-graph

[2] Carleton University, “Computer Science (COMP) < Carleton University,” *Carleton University Calendars*, 2024. https://calendar.carleton.ca/undergrad/courses/COMP/

[3] S. Zubaidah, N. M. Fuad, S. Mahanal, and E. Suarsini, “Improving Creative Thinking Skills of Students through Differentiated Science Inquiry Integrated with Mind Map,” *Journal of Turkish Science Education*, vol. 14, no. 4, pp. 77–91, Dec. 2017, Available: https://www.tused.org/index.php/tused/article/view/175

[4] D2L, “Brightspace,” *D2L*, 2024. https://www.d2l.com/brightspace/

[5] D. Zeng, K. Liu, S. Lai, G. Zhou, and J. Zhao, “Relation Classification via Convolutional Deep Neural Network,” 2014. Available: https://aclanthology.org/C14-1220.pdf

[6] Enzo, “What Is a Knowledge Graph?,” *Neo4j*, Jul. 18, 2023. https://neo4j.com/blog/what-is-knowledge-graph/ (accessed Apr. 14, 2024).

[7] S. Ward, “What is a mind map?,” *Mural*, Aug. 30, 2023. https://www.mural.co/blog/mind-mapping

[8] IBM, “What is Natural Language Processing?,” *IBM*, 2023. https://www.ibm.com/topics/natural-language-processing

[9] K. D. Foote, “A Brief History of Natural Language Processing (NLP),” *Dataversity*, May 22, 2023. https://www.dataversity.net/a-brief-history-of-natural-language-processing-nlp

[10] A. Khanna, “Tokenization Techniques in Natural Language Processing in Python,” *Medium*, Feb. 10, 2022. https://medium.com/@ajay_khanna/tokenization-techniques-in-natural-language-processing-67bb22088c75

[11] S. Prabhakaran, “Cosine Similarity – Understanding the math and how it works,” *Machine Learning Plus*, Oct. 22, 2018. https://www.machinelearningplus.com/nlp/cosine-similarity/

[12] IBM, “What Are Large Language models?,” *IBM*, 2024. https://www.ibm.com/topics/large-language-models

[13] M. Lubbad, “The Ultimate Guide to GPT-4 Parameters: Everything You Need to Know about NLP’s Game-Changer,” *Medium*, Mar. 20, 2023.

https://medium.com/@mlubbad/the-ultimate-guide-to-gpt-4-parameters-everything-you-need-to-know-about-nlps-game-changer-109b8767855a
[14] J. Banks and T. Warkentin, “Gemma: Introducing new state-of-the-art open models,” *Google*, Feb. 21, 2024.
https://blog.google/technology/developers/gemma-open-models/
[15] K. Chandler, “Context Windows: The Short-term Memory of Large Language Models,” *Medium*, Nov. 03, 2023.
https://medium.com/@crskilpatrick807/context-windows-the-short-term-memory-of-large-language-models-ab878fc6f9b5
[16] OpenAI, “Join us for OpenAI’s first developer conference on November 6 in San Francisco,” *OpenAI*, Sep. 06, 2023.
https://openai.com/blog/announcing-openai-devday
[17] B. Sisman, J. Yamagishi, S. King, and H. Li, “An Overview of Voice Conversion and Its Challenges: From Statistical Modeling to Deep Learning,” *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 29, no. 1, pp. 132–157, 2021, doi: https://doi.org/10.1109/taslp.2020.3038524.
[18] G. Bello-Orgaz, J. J. Jung, and D. Camacho, “Social big data: Recent achievements and new challenges,” *Information Fusion*, vol. 28, no. 1, pp. 45–59, Mar. 2016, doi: https://doi.org/10.1016/j.inffus.2015.08.005.
[19] Hugging Face, “LLM prompting guide,” *Hugging Face*, 2024.
https://huggingface.co/docs/transformers/main/en/tasks/prompting
[20] B. Lester *et al.*, “Training LLMs over Neurally Compressed Text,” *arXiv (Cornell University)*, vol. 1, no. 1, Apr. 2024, doi:
https://doi.org/10.48550/arxiv.2404.03626.
[21] A. Modarressi, A. Imani, M. Fayyaz, and H. Schütze, “RET-LLM: Towards a General Read-Write Memory for Large Language Models,” *arXiv.org*, May 23, 2023. https://arxiv.org/abs/2305.14322
[22] Ollama, “Ollama,” *ollama.com*, 2024. https://ollama.com/
[23] F. Oh, “What Is CUDA,” *NVIDIA*, Sep. 10, 2012.
https://blogs.nvidia.com/blog/what-is-cuda-2/
[24] NVIDIA, “NVIDIA TITAN V Transforms the PC into AI Supercomputer,” *NVIDIA Newsroom*, Dec. 07, 2017.
https://nvidianews.nvidia.com/news/nvidia-titan-v-transforms-the-pc-into-ai-supercomputer (accessed Apr. 15, 2024).

[25] G. Bonnot, “The Pipeline Design Pattern,” *Medium*, Apr. 04, 2020. https://medium.com/@bonnotguillaume/software-architecture-the-pipeline-design-pattern-from-zero-to-hero-b5c43d8a4e60

[26] Apache Software Foundation, “Apache Airflow,” *Apache Airflow*, 2024. https://airflow.apache.org/

[27] Prefect Technologies, “Prefect - The New Standard in Dataflow Automation,” *Prefect*, 2024. https://www.prefect.io/

[28] Dagster Labs, “Dagster,” *dagster.io*, 2024. https://dagster.io/

[29] MongoDB, “The most popular database for modern apps,” *MongoDB*, 2019. https://www.mongodb.com/

[30] MongoDB, “NoSQL Databases Explained,” *MongoDB*, 2019. https://www.mongodb.com/nosql-explained

[31] FastAPI, “FastAPI,” *FastAPI*, 2024. https://fastapi.tiangolo.com/

[32] Okta, “8 Ways to Secure Your Microservices Architecture,” *Okta*, May 22, 2020. https://www.okta.com/resources/whitepaper/8-ways-to-secure-your-microservices-architecture/

[33] Meta Open Source, “React,” *react.dev*, 2024. https://react.dev/

[34] V. Asturiano, “react-force-graph,” *GitHub*, Aug. 21, 2022. https://github.com/vasturiano/react-force-graph

[35] Docker, “Enterprise Application Container Platform,” *Docker*, 2018. https://www.docker.com/

[36] NVIDIA, “NVIDIA Container Toolkit 1.14.5 documentation,” *NVIDIA Docs*, Apr. 11, 2024. https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/install-guide.html

[37] Docker, “Overview of Docker Compose,” *Docker Documentation*, Feb. 10, 2020. https://docs.docker.com/compose/

[38] Massachusetts Institute of Technology, “Urban Operations Research,” *web.mit.edu*, 1965. https://web.mit.edu/urban_or_book/www/book/chapter6/6.1.html

[39] S. Das, “Fine Tune Large Language Model (LLM) on a Custom Dataset with QLoRA,” *Medium*, Jan. 25, 2024. https://dassum.medium.com/fine-tune-large-language-model-llm-on-a-custom-dataset-with-qlora-fb60abdeba07

[40] P. M. Bays, “Evaluating and excluding swap errors in analogue tests of working memory,” *Scientific Reports*, vol. 6, no. 1, Jan. 2016, doi: https://doi.org/10.1038/srep19203.

[41] The Browser Company, “What if Arc browser can’t build reliable AI?,” *Youtube*, Apr. 05, 2024. https://www.youtube.com/watch?v=y3FpsSDIbJ8

[42] S. Grashchenko, “Levenshtein Distance Computation | Baeldung on Computer Science,” *Baeldung*, Aug. 03, 2020. https://www.baeldung.com/cs/levenshtein-distance-computation

[43] S. Pichai and D. Hassabis, “Our next-generation model: Gemini 1.5,” *Google*, Feb. 15, 2024. https://blog.google/technology/ai/google-gemini-next-generation-model-february-2024/#sundar-note

[44] Pydantic, “Pydantic,” *Pydantic Docs*, 2024. https://docs.pydantic.dev/latest/

[45] K. Eby, “The Power of Iterative Design and Process,” *Smartsheet*, Jan. 02, 2019. https://www.smartsheet.com/iterative-process-guide

[46] Microsoft, “Microsoft OneNote | The digital note-taking app for your devices,” *Onenote*, 2019. https://www.onenote.com/signin?wdorigin=ondc

[47] Figma, “Figma: the Collaborative Interface Design tool.,” *Figma*, 2024. https://www.figma.com/

08 APPENDIX

08-A Prompt Instructions

The following prompt template was used for tasking the Gemma LLM to summarize lecture transcripts. Note the use of markdown headings (#) to signify different elements of the prompt.

```
# System Instructions
Summarize the following lecture transcript to identify key concepts relevant to the class.
# Lecture Transcript
{transcript}
```

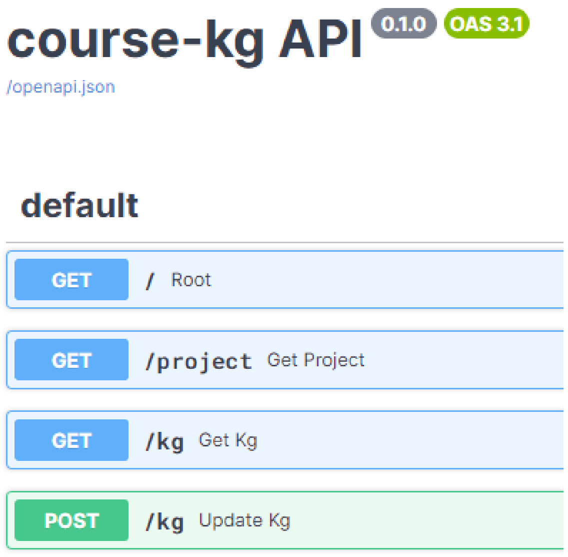
The next prompt template was used to generate JSON knowledge graphs based on transcript summaries.

```
# System Instructions
Given the following key concepts extracted from a university lecture transcript, list the nodes and edges in JSON format that can form a knowledge graph based on the key topics. Only create nodes based on large concepts. Only create an edge if the source concept is a prerequisite to understand the target concept.
# Output Format Instructions
Produce only JSON output exactly like this: ```json\n{nodes: [{id: <name of concept>}...], edges: [{source: <id of source concept node>, target: <id of target concept node>}]}\n```
# Key Concepts
{transcript}
```

Since these prompts are written with natural language, they can be easily customized to augment the knowledge graph generation process.

08-B REST API documentation

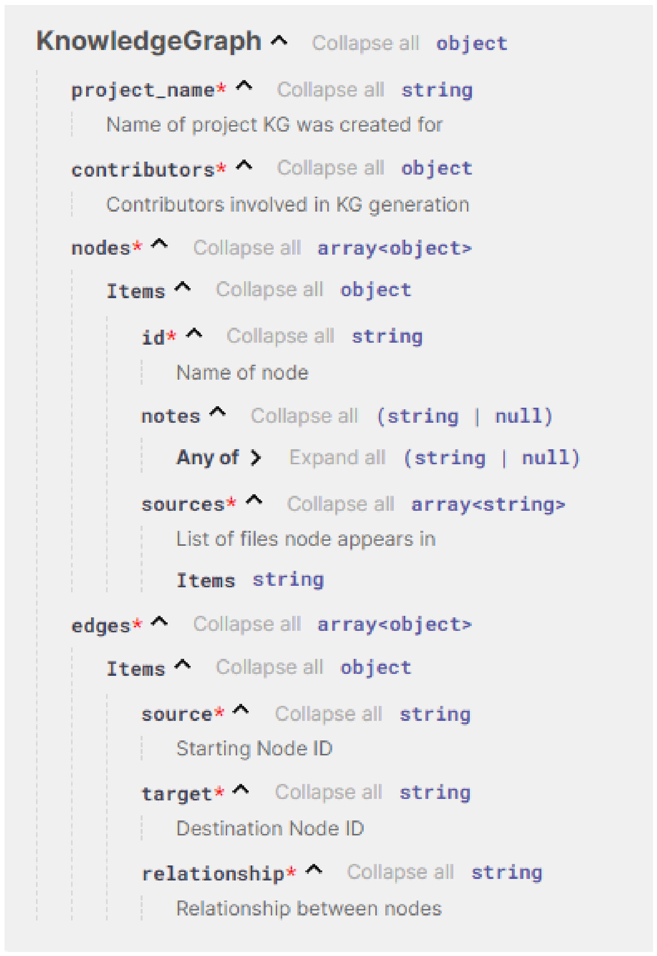
The following REST API documentation was generated using FastAPI's automatic documentation feature using Swagger UI [31]. This interface is available on http://localhost:8000/docs. Developers are able to view each endpoint and the query parameters / request body required, along with schemas for the expected responses. These schemas are generated using Pydantic models [44] representing the data retrieved from MongoDB.



course-kg API 0.1.0 OAS 3.1
/openapi.json

default

GET / Root
GET /project Get Project
GET /kg Get Kg
POST /kg Update Kg



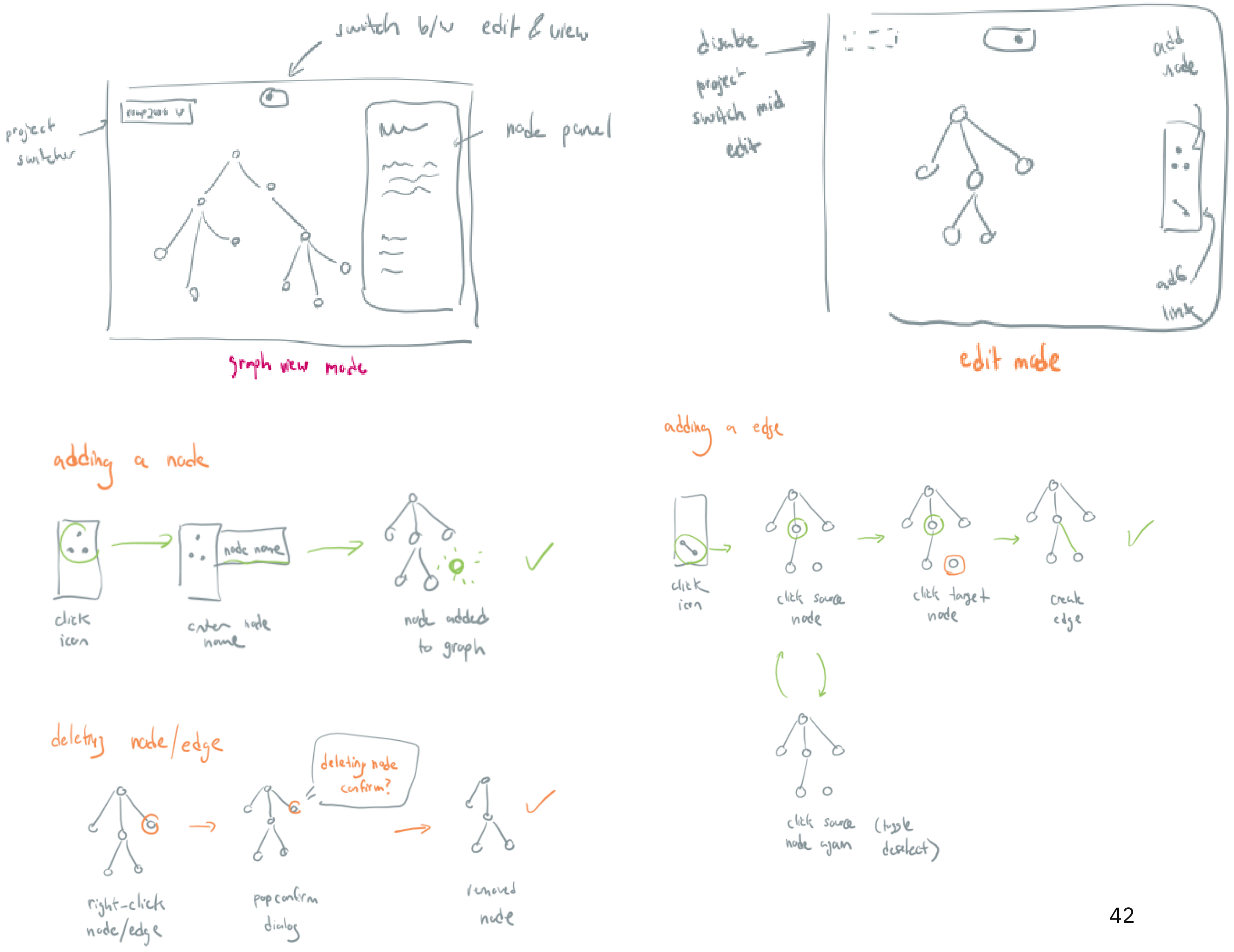
```
KnowledgeGraph ^  Collapse all  object
  project_name* ^  Collapse all  string
    Name of project KG was created for
  contributors* ^  Collapse all  object
    Contributors involved in KG generation
  nodes* ^  Collapse all  array<object>
    Items ^  Collapse all  object
      id* ^  Collapse all  string
        Name of node
      notes ^  Collapse all  (string | null)
        Any of >  Expand all  (string | null)
      sources* ^  Collapse all  array<string>
        List of files node appears in
        Items  string
  edges* ^  Collapse all  array<object>
    Items ^  Collapse all  object
      source* ^  Collapse all  string
        Starting Node ID
      target* ^  Collapse all  string
        Destination Node ID
      relationship* ^  Collapse all  string
        Relationship between nodes
```

08-C frontend application

The development of the frontend application follows the iterative design process of creating wireframes, high fidelity mockups, and functional prototypes [45].

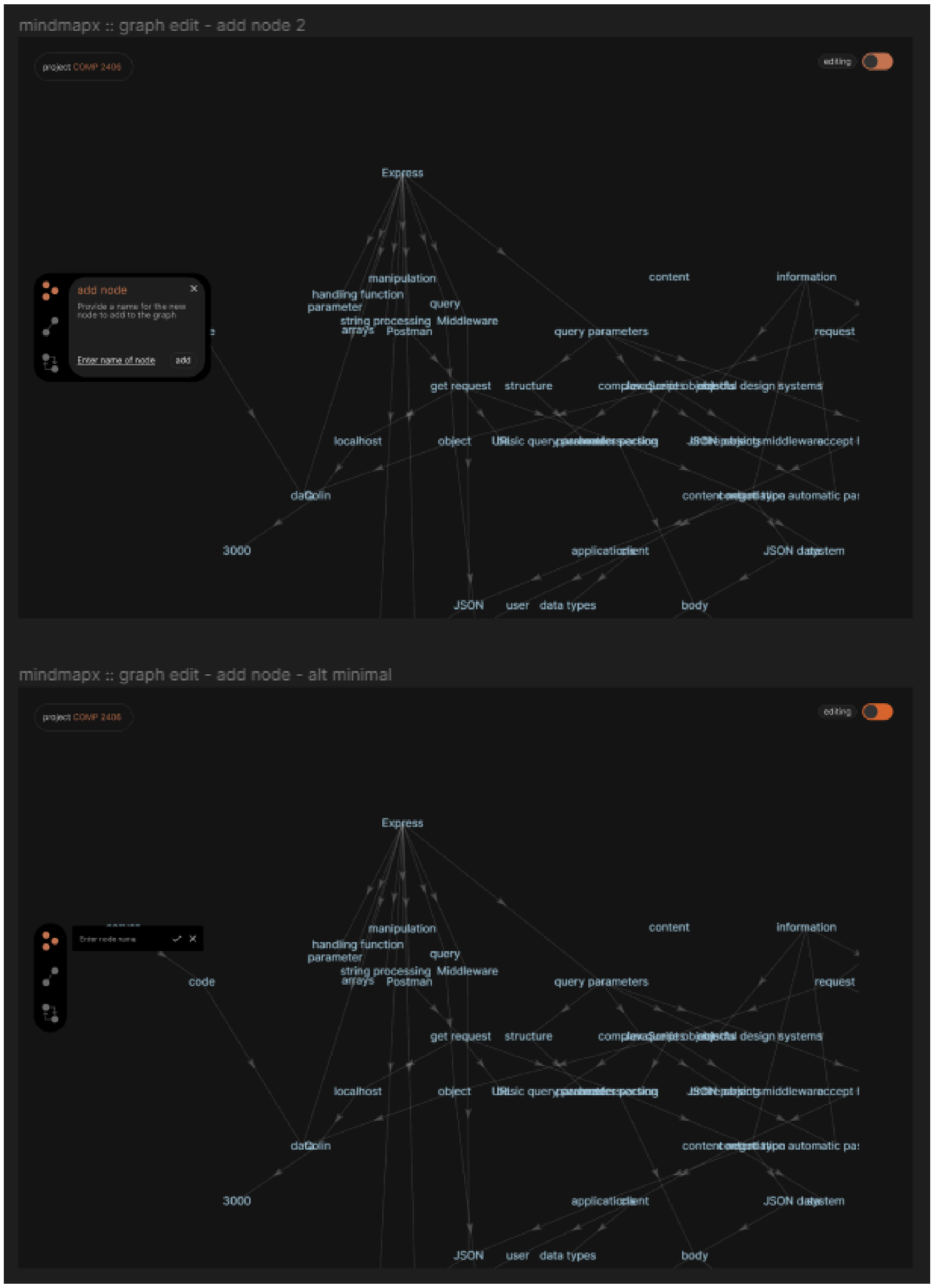
The following wireframes consist of rough drawings created using OneNote [46] which outline the user flow and the various UI components. These are used to inform the development of the web client.

Figure 19: Wireframes of KG webclient



The following mockups were created using Figma [47], an interface design tool used to quickly build products. Various alternate designs were created to experiment with different interfaces and their impact on usability.

Figure 20: High fidelity mockups for KG webclient



Using the mockups and wireframes as a reference, the webclient was implemented using React [33]. This application can be accessed on http://localhost:3000/. The remainder of this section will discuss the various features available.

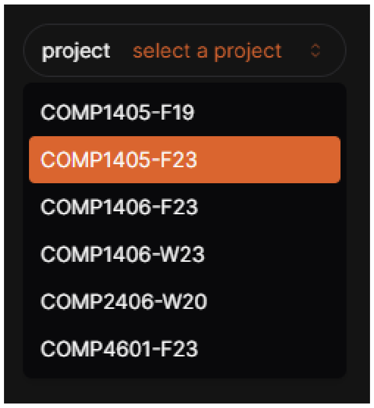
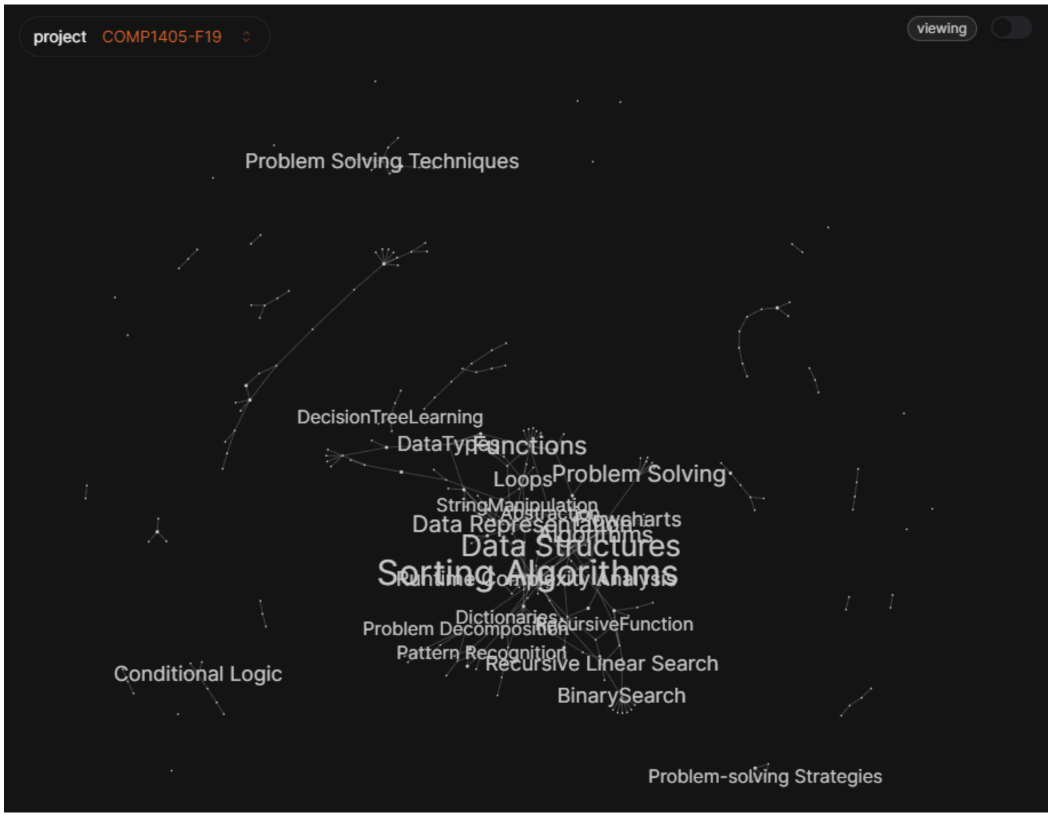


Figure 21: Selecting a project

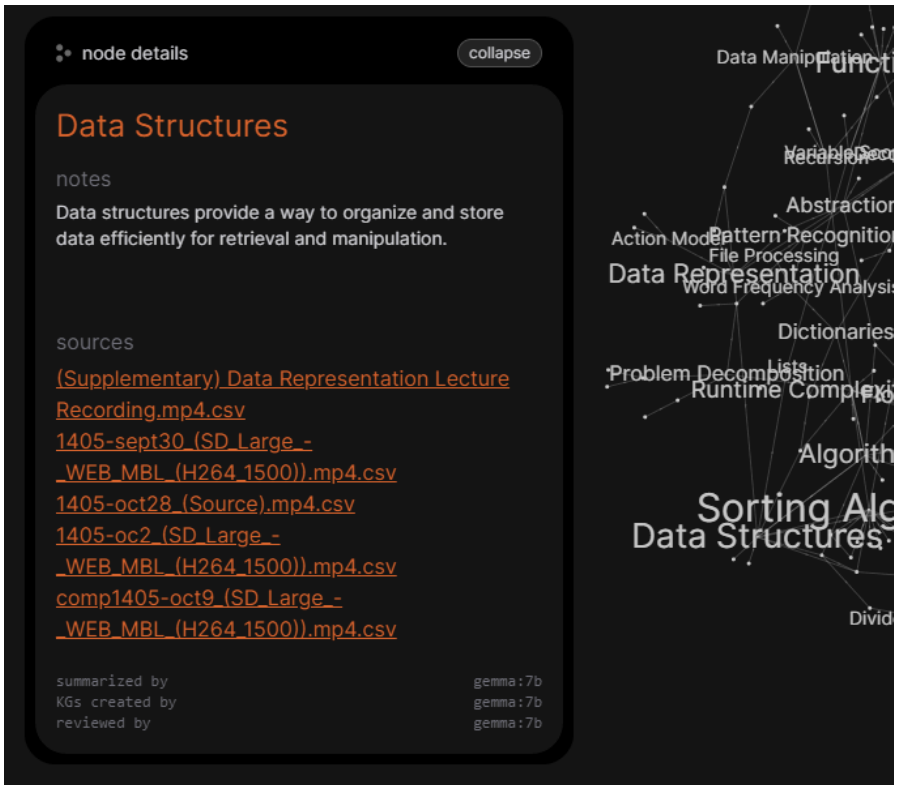
The first step the user should follow when reaching the application is to select a "project". A project consists of a knowledge graph generated by lecture transcripts for a particular course. These KGs are stored in a MongoDB, and are retrieved by the webclient through the REST API service. Upon selecting a project, Figure 22 demonstrates how the KG may appear.

Figure 22: Viewing a KG



Users are able to navigate through the knowledge graph by using the mouse to zoom, pan, and drag nodes. At certain zoom levels, the node labels will not appear and instead be replaced by a “dot”. Every node is assigned a value based on its degree, the number of edges leading to/from a node. Thus as the user zooms out, only nodes with higher values will be visible. Users can also hover over edges to view the “relationship” between concepts.

Figure 23: Viewing node details



node details

collapse

Data Structures

notes

Data structures provide a way to organize and store data efficiently for retrieval and manipulation.

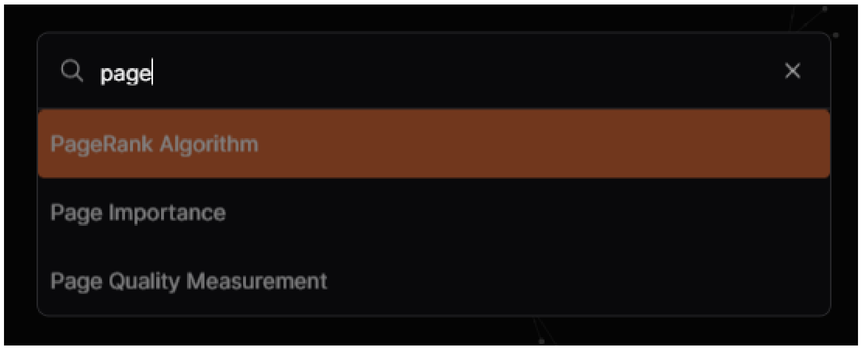
sources

(Supplementary) Data Representation Lecture Recording.mp4.csv
1405-sept30_(SD_Large_-_WEB_MBL_(H264_1500)).mp4.csv
1405-oct28_(Source).mp4.csv
1405-oc2_(SD_Large_-_WEB_MBL_(H264_1500)).mp4.csv
comp1405-oct9_(SD_Large_-_WEB_MBL_(H264_1500)).mp4.csv

```
summarized by     gemma:7b
KGs created by    gemma:7b
reviewed by       gemma:7b
```

Upon right-clicking a node, a “node details” panel will appear providing additional information regarding a concept. This includes an LLM-generated summary of the node based on the context of the lecture transcript it was extracted from. As well as a list of sources representing the various lectures the concept was mentioned in.

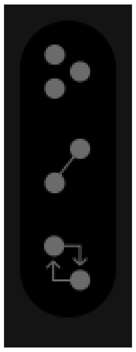
Figure 24: Searching for nodes by ID



The webclient also supports the ability to search for nodes by their ID. By using Ctrl+K, a popup search bar will appear. After choosing a node and hitting Enter, the interface will pan the camera towards the specific node.

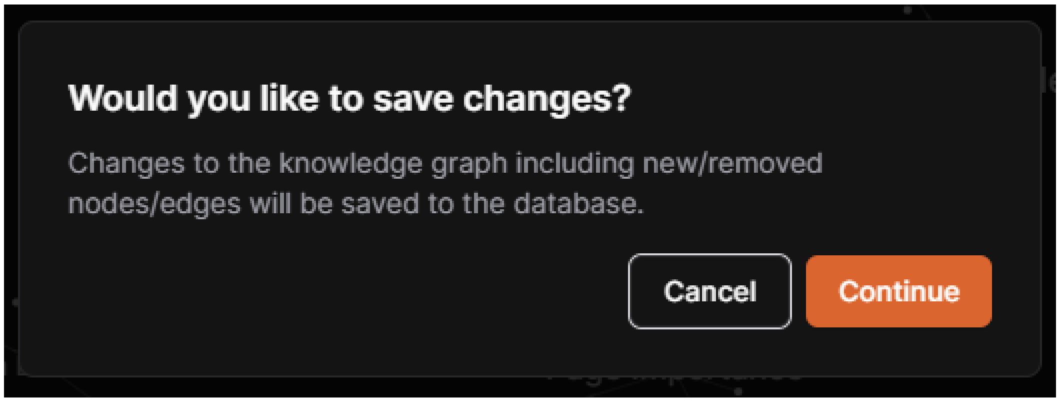
Figure 25: Switching to edit mode

Users looking to modify the KG can use the edit mode switch. This will lock the project switching feature, and reveal a KG-editing toolbar.



Using this toolbar, users can add nodes, add edges, and sort the graph. By clicking on the corresponding icons, instructions will appear regarding the specific steps needed. However, the sort feature only works if the generated KG is a Directed Acyclic Graph (DAG), which oftentimes is not the case as the LLMs can create cyclical connections between nodes. Users can also right-click nodes and edges to delete them. Be wary that there is currently no undo-feature.

Figure 26: Saving KG changes


Would you like to save changes?

Changes to the knowledge graph including new/removed nodes/edges will be saved to the database.

Any changes made during the "editing" mode are only saved locally during a user's browser session. To propagate these changes to the database, the user can use Ctrl+S to save KG modifications. Be wary that these changes will affect all users accessing this KG. In the future, a proper authentication/authorization mechanism will be required to control which user types have permission.