The LLM-Based Mini-Lecture Generation for Enhanced Learning Project

HiLabs Inc. 1

Gabrielle Christensen
Grace Corpron
Jessica Engel
Peter Hoila

December 10, 2023

CSCI 370 Fall 2023

Prof. Bodeau

Table 1: Revision History

Revision	Date	Comments
1	Sep 1, 2023	Completed Sections: I. Introduction II. Functional Requirements III. Non-functional Requirements IV. Risks V. Definition of Done XIII. Team Profile Updated Sections: References Appendix A: Key Terms
2	Sep 17, 2023	Completed Sections: VI. System Architecture
3	Oct 22, 2023	Completed Sections: VII. Software Test and Quality VIII. Ethical Considerations Updated Sections: VI. System Architecture References
4	Nov 12, 2023	Completed Sections: XII. Acknowledgments IX. Project Completion Status X. Future Work XI. Lessons Learned Updated Sections: VI. System Architecture VII. Software Test and Quality Appendix A: Key Terms
5	Dec 4, 2023	Completed Sections: XI. Technical Design Updated all previous sections

Table of Contents

I. Introduction..... 4
II. Functional requirements..... 4
III. Non-functional requirements..... 5
IV. Risks..... 5
V. Definition of done..... 6
VI. System Architecture..... 7
VII. Technical Design..... 7
VIII. Software Test and Quality..... 9
IX. Ethical Considerations..... 11
X. Results..... 13
XI. Future Work..... 14
XII. Lessons Learned..... 15
XIII. Acknowledgements..... 16
XIV. Team Profile..... 17
References..... 17
Appendix A..... 18

I. Introduction

HiLabs Inc. is a tech startup dedicated to providing AI tools for educational professionals and students alike. In alignment with this mission, Hi Labs Inc. has proposed the Large-Language-Model-Based Mini-Lecture Generation Project, abbreviated as the LLM-LGP (Large Language Model Lecture Generation Project).

With the recent rise in publicly available generative AI, students have taken advantage of AI tools to enhance their learning. One might prompt a generative AI model to explain a subject to them as if they were a novice or young child or asking for assistance on the answer to a practice exam given for a class. Currently, there is no guarantee that the AI model will return accurate or reliable information. Receiving the desired level of complexity within an explanation additionally requires an understanding of prompt engineering techniques on the user end; those who are less skilled in prompt engineering may get disproportionately lower quality results from generative AI tools. Large Language Models (LLM) are incapable of distributing results in creative formats that are not simply blocks of text. This project addressed all of the above issues; using enhanced prompt engineering and fine-tuning to ensure that students can receive accurate information, and customized explanations without the extra step of prompting an AI.

The LLM-LGP allows students to easily generate a lecture on a given subject customized to their own learning needs. A user will be prompted to input specifications for a lecture, and from those specifications, a slide deck will be generated that teaches the users the basics of a desired topic. The project was tested on open-source educational content and utilized an open-source LLM. Upon completion of the project, HiLabs will be responsible for the continued maintenance of the software.

This project is a separate version of an existing lecture generation project by Hi Labs, in which an OpenAI LLM model and professor-uploaded content were used. The initial project proved to be successful, with approximately 120 users at the peak of its popularity. However, the cost of an OpenAI key made the service expensive to run for the company. The LLM-LGP primarily differs in its use of an open-source LLM, which requires learning how best to interface with said LLM: an undertaking that HiLabs has designated to us. These adjustments will greatly reduce the cost of running the service for HiLabs and allow the product to be accessible to all students for free.

II. Functional requirements

The primary components of this project include the LLM summarization based on uploaded lecture material, implementing API wrappers, and user specification for desired output. To allow for easy user customization and an interface to upload materials, the product requires a functional front-end interface.

Specific functionality includes

- Utilization of an LLM to output a modified, easily digestible lecture
- Ability to take user specifications and convert them into a prompt

- Prompt engineering that can produce a reliable output that is ingestible by a Python library to make slides
- Capability to upload PDF files
- Capability to parse uploaded PDF files to feed into the LLM
- Functions wrapped in an API that can be called using JavaScript and/or TypeScript

III. Non-functional requirements

In addition to the main functionality of the product, we must consider constraints regarding computing power, cost of technologies, and copyright of educational materials. Non-functional requirements include:

- LLM must be pre-trained, open-source, and able to be run locally on a personal machine or through an API
- Educational content for testing and model training must be open-source
- LLM context must be limited to reduce computing power
- The generated lecture must be output into a .pptx or pdf

IV. Risks

Before beginning development, several technical risks and skill risks were identified in order to develop plans to combat them throughout the duration of the project. Technical risks are risks we encountered with the software we use (as our project doesn't have any hardware specifications). These can be reduced by implementing clean coding practices, along with comprehensive code reviews and extensive testing. Skill risks are risks we encountered in learning new languages or how to use new technologies. Incorrect understanding of these skills throughout our learning could have impacted the development and functionality of our project. All skill risks listed below were mitigated by keeping open communication between each other.

Table 2: Risks

Identified Risk	Likelihood	Impact	Risk Mitigation Plan
Technical risks			
We may experience difficulties in calling and tuning the LLM model	Very Likely	Major	We experimented with different local LLMs as well as with the OpenAI models
The model may generate false or inaccurate information	Likely	Moderate	We researched prompt engineering or limit the scope of the LLM prompts

We may encounter issues integrating our code to be callable from typescript/javascript.	Unlikely	Minor	We tried different methods of integration (Ajax request, child process, fetching)
Skill risks			
No team member has experience in tuning an LLM	Certain	Major	We researched and tried different LLM models with the help of our client
No team member has experience in developing an API	Certain	Minor	We researched Python API creation tools such as flask
No team member has experience in prompt engineering	Certain	Moderate	We researched prompt engineering techniques and try them on different configurations of LLMs
Some team members have integrated languages before, but never JavaScript with Python	Certain	Minor	We tried different methods of integration (Ajax request, child process, fetching)
Operational risks			
The user does not supply the LLM with enough material to generate a lecture	Likely	Moderate	We are unable to mitigate this risk at this time.

V. Definition of done

Our minimum viable product (MVP) is a functional end-to-end product to provide proof of concept for the LLM-LGP. The MVP features a mock user interface that may be swapped with HiLabs' own interface. It includes an API that takes in various lecture materials and, if desired by the user, specifications as to how the output should be written and formatted. It then outputs a modified, easily digestible lecture using a custom-trained LLM. This API can be called from JavaScript in order to integrate with HiLabs' front end. The integration with their current codebase has not been tested as we don't have access to their current GitHub repository. The product was delivered by giving our client access to our team's GitHub repository. This includes the API code as well as the configuration of the chosen LLM, but not the LLM itself.

VI. System Architecture

Figure 1: System architecture draft

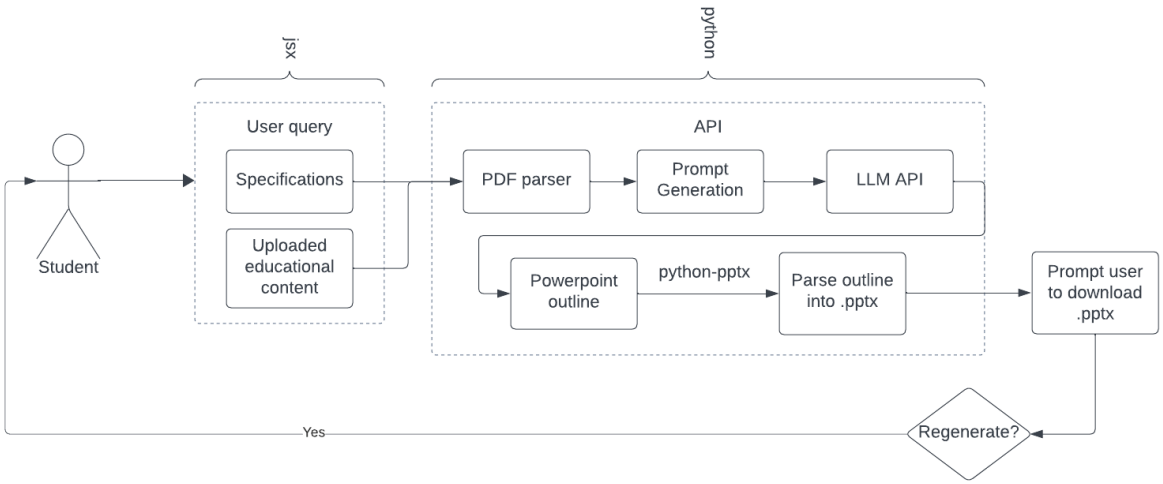


Figure 1 demonstrates the system architecture for the LLM-LGP. The user query, entered by the student, contains both the prompt specifications provided by the user, and the lecture material uploaded by the user. The prompt specifications are provided as selectable options to ensure the prompt language is accurate every time.

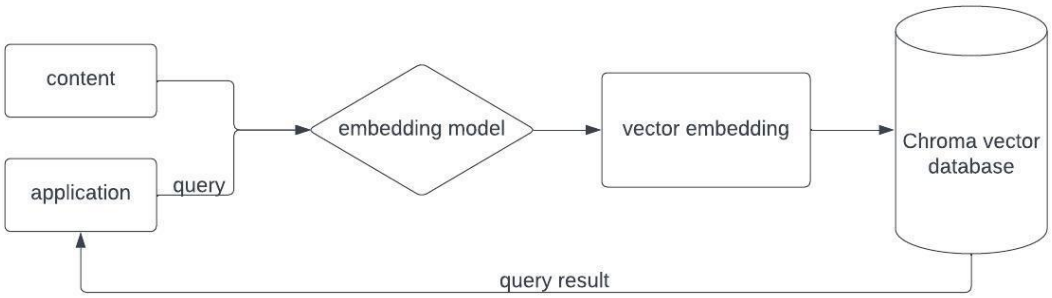
Both of the aspects from the user query are fed into a prompt. The LLM is fine-tuned to answer the specific prompt format with the user specifications, ensuring the results are always reputable. The LLM then learns specific language and vocabulary on the user's topic through a dataset of open-source educational content. From this, the LLM outputs a PowerPoint outline as a general text file.

The generated outline is then parsed through a set of functions using Python's python-pptx library, in which Python looks for specific keywords/formatting within the text to create titles, text boxes, and slide breaks. Once a PowerPoint is generated from the outline, the user is then prompted to download the presentation as either a .pptx or .pdf.

VII. Technical Design

This project requires reading large amounts of uploaded data and ensuring that data is readable by an LLM. To achieve this, we utilized Chroma DB: a vector database. Instead of passing the text content of entire documents as context to an LLM, we can find just the most relevant text chunks. This is done by splitting each document into text chunks and then embedding each chunk as a vector based on the contents of the text chunk. This step needs to be done only once for each upload and the vector database is used for all future queries. The queries are also transformed into a vector and compared against the vector database to find the most similar vectors. These vectors are then mapped back to their original text chunks and passed as context to the LLM. The text chunks also retain metadata including the file and page number, which allows us to create citations for our slides.

Figure 2: Vector database architecture



Parsing through uploaded files is one of the key functionalities within our project. PDF files are not always guaranteed to be clean. Reading plain text from a PDF is a simple task, but content such as images or equations is not as easily read. The parse_docs function, as shown in Figure 2, was able to utilize both PyPDFLoader for loading readable PDF files and an unstructured document loader to handle abnormal formatting within the uploaded files. If the page content was empty from the first path utilizing the PyPDFLoader, Unstructured.io's document loader API was called instead. It should be noted that this functionality is still being explored since files that are partially readable won't trigger the use of the unstructured document loader. Ideally, the unstructured document loader would be used on every file, however, this would come with high computing costs due to the image recognition techniques used.

Figure 3: Document parsing function

```
def parse_docs(dir, use_openai=False):
    if use_openai:
        embeddings = OpenAIEmbeddings()
    else:
        embeddings = OllamaEmbeddings()
    dir = os.path.abspath(dir)
    vector_store = Chroma("langchain", embeddings)
    # iterate through each file
    for file in os.listdir(dir):
        filepath = os.path.join(dir, file)
        loader = PyPDFLoader(filepath)
        documents = loader.load()
        # if the page_content is unreadable, use Unstructured API
        if documents[0].page_content == "":
            loader = UnstructuredAPIFileLoader(
                file_path=filepath,
                api_key=UAPI_KEY
            )
            documents = loader.load()
```

We lastly had to implement highly robust prompt engineering techniques to ensure a reliable output from the LLM every time. LLMs often include conversational lines such as “Sure, here is a presentation on your topic.” Having this output within the generated slides would decrease the quality of the presentation. The prompt in Figure 3 was effectively able to generate an output without conversational lines by explicitly prohibiting extraneous phrases or non-delimited lines.

Figure 3: Prompt template

```
template = """Create a powerpoint for a presentation meant for a student to learn about "{topic}" without
additional explanation needed. Ensure the content is worded such that it is tailored to the user's preferred
complexity level and current understanding. On a scale of 1-5, the complexity level is {complexity}. On a scale
of 1-5, the current understanding level is {understanding}. Use "h:" for title headers and do not include any
extraneous phrases such as "of course" or "certainly". Do not exceed 20 words for each bullet point. Include a
title for the presentation "t: and do not include any line that doesn't start with a delimiter."
```

Strictly following these delimiters is imperative to successfully run our PowerPoint conversion script. However, we recognized that this prompt may not be entirely failsafe. Therefore, in the PowerPoint generation module, we used regex patterns to further account for LLM consistency and filtered some common conversational words.

Figure 4: PPTX generation

```
# variety of regex patterns to account for LLM inconsistency
title_pattern = re.compile(r'(?i)^(t|title)\d*:')
header_pattern = re.compile(r'(?i)^((h *|slide *)\d*:)|^(I|II|III|IV|V|VI|VII|VIII|IX|X|XI|XII)(:|.)')
bullet_pattern = re.compile(r'^(\*|-):*')
conversation_words = ["certainly", "sure", "note:"]  # these words likely mean the LLM is conversing with the user
dividing_lines = ["--", "=="]
```

VIII. Software Test and Quality

In order to ensure that the software works as expected, a testing plan is implemented for each of the functional requirements for the minimum viable product. All unit tests described below are implemented using the pytest Python library [1].

Use of an LLM to output a modified, easily digestible lecture

To test this requirement, we verified that the LLM-generated lecture is “easily digestible”. The definition of “easily digestible” is variable, and the content of the LLM’s response is non-deterministic (the output is different every time). Therefore, direct testing of these aspects is not feasible. However, we are able to test the specific word count of each bullet point in a generated lecture. It defeats the purpose of a PowerPoint output format if reading each bullet point is like reading a block of text. Our team has determined that a word count of 20 is an appropriate maximum for each bullet point. We split the string of each bullet point by space and

counted each word within the split array. The results of this test should confirm that no splits exceed a length of 20. We were unable to implement automated testing for this over the duration of this project, but manual testing does confirm that the output lectures have been easily digestible under these standards.

Capability to upload lectures, homework, and other class material as input

The LLM-LGP cannot function if the user uploads an incorrect file type. Currently, the software only accepts PDF files; the PDF parser will not work if it is not given a PDF. From our mock user interface, we tested that the user has a correct filename input format, the file they uploaded exists, and there weren't any errors uploading the file. This test's purpose is to identify edge cases where the input file is not a PDF file type or the file is null. The result of this test was that we were able to successfully move forwards when the correct file was uploaded, but when an incorrect file was uploaded, the program would not move forwards and the user was prompted to upload a file of the correct format instead. This was additionally verified with pytests that simulated API calls containing a valid and an invalid document type.

Ability to take user specifications and convert them into a prompt

User specifications must be failsafe. While we expect HiLabs to replace our makeshift front end with their own front end upon completion of the project, we nevertheless want to ensure that faulty user input does not interfere with the rest of the project. Having our own mock UI allows for more efficient API testing. We implemented unit tests using pytest in order to ensure user specifications were correctly processed and converted. If a user does not fill out all requested fields for difficulty level, specific topic, etc, the program should not continue until they have done so in order to prevent holes in the prompt. An edge case of this testing is that a user requests a specific topic for the lecture to focus on, but their request does not pertain to the content of the pdf (e.g. a user enters "wildflowers" when the uploaded lecture material pertains to parts of the brain). In this instance, we implemented testing for whether the lecture material actually contains the subtopic. If the parsed PDF does not contain any instances of any part of the specific string the user is requesting, it should throw an error message to the user and not continue. This test should still pass in cases where the user enters a similar string, such as an input of "vaccines" when the lecture only contains the singular "vaccine".

Functions wrapped in an API that can be called using JavaScript and/or TypeScript

We tested that all JSON data passed to the API is posted with no errors. This required us to test that all calls made to the API at each stage of the lecture generation process resulted in success, and if not, a message was returned specifying why an error occurred. This is made easy by the fact the flask API implements codes to categorize responses. The results of this test should be that all 3-digit API response codes are an int (i) such that $100 \leq i \leq 200$. These results

were confirmed in our testing when we successfully confirmed that the API could be called, and returned the correct response code for the information passed to it.

Prompt engineering to produce a reliable output that can be ingested by a Python library to generate slides

We tested this by ensuring that every line of the output matched the expected format. The purpose of testing this is to ensure that the parser can create a PowerPoint from the outline. There is little room for error since the parser might break if it receives input that is in the wrong format. We did this by uploading a mock outline and ensuring that the output title and slides had the text and format we expected. Because of the nature of this test, we were unable to implement automated testing. However, through qualitative examination, we were able to confirm that the prompts generated can reliably produce the expected output.

Capability to parse said material to feed into the LLM

We tested this capability by creating mock educational materials, such as PDFs with filler content. We ensured that all text from the material is read and processed by checking that all text appears in the parser. The purpose of this test is to ensure the program can correctly interpret the material that the student inputs. Edge cases include corrupted PDF files where text is overlapped, hidden, or displayed in a non-sequential format. Such situations are difficult to handle with the parser, therefore, the acceptability threshold is low for these tests.

IX. Ethical Considerations

The ultimate goal of the LLM-LGP is to improve the lives of students. Despite our intent, there is still a possibility of violating ethics when implementing this project such as protecting the intellectual property of the company HiLabs, providing accurate information, and giving students information about how the model was trained. Recognizing these problems is pivotal to creating a platform that students and instructors feel comfortable and confident using. The ACM Code of Ethics and Professional Conduct is an ethical framework that outlines principles that all software engineers should adhere to, and will be referenced in these ethical considerations [2].

One of our most critical considerations is protecting the intellectual property of professors. Many professors do not want their course content to be shared outside of the classroom. Faculty-generated property, like much intellectual property, is subject to copyright laws, whether ownership falls under a faculty member themselves, a publisher, or the university. This fact begs the question, how does AI mitigate copyright infringement? Authors Sarah Silverman, Christopher Golden, and Richard Kadrey filed suit against OpenAI in July for this very issue [3]. They allege that OpenAI's ChatGPT and Meta's LLaMA train their models on illegally obtained works through piracy sites like Library Genesis, Bibliotik, and Z-library. The plaintiffs claim that they did not consent to the use of their copyrighted books as training material. This case is still ongoing, but it nevertheless shows that the definition of copyright

becomes blurred in the context of AI. We are in danger of violating ACM Principle 1.5: Respect the work required to produce new ideas, inventions, creative works, and computing artifacts… respect copyrights, patents, trade secrets, license agreements, and other methods of protecting authors' works," [2].

Even if an AI were to correctly obtain its sources, what are the chances it is able to accurately cite a source? AI systems are notorious for hallucinating sources when prompted to cite; it will claim something is true, and cite a source, but there is no guarantee that a piece of information actually came from a source, or if the URL it sources is real. This additionally pertains to ACM Principle 1.5: Respect the work required to produce new ideas [2]. Citing sources is an academic standard not only for students to show they conducted proper research, but also to give credit to and acknowledge other researchers' ideas.

In the context of the LLM-LGP, we cannot control how the AI model we use is trained. Not every AI company is transparent about the datasets used to train their models; if every AI company used the same public dataset, there may be less competition among them. Therefore, mitigating the copyright issues associated with our chosen LLM's training was out of scope. However, we took action to ensure the LLM correctly cited and to ensure any piece of intellectual property is kept private once it is handed over to the LLM-LGP. This is only possible with an offline LLM model since online models such as ChatGPT learn from user input. Many companies have banned their employees from uploading private documents to ChatGPT.

Students should only be able to see their own prompts and uploaded material so that a professor's lecture material is not shared with everyone using the LLM-LGP. This required us to uphold ACM Principle 1.6: Respect privacy [2]. It required us to design the system such that it upholds ACM Principle 2.9: Design and implement systems that are robustly and usably secure [2]. For citations, we implemented strong prompt engineering to explicitly tell the LLM that it may not use sources outside of what the student has uploaded. Accurate information is still not an absolute guarantee, even with the most sophisticated prompt engineering, we recommend that HiLabs implements a disclaimer in their own final user interface to inform users of the potential for the LLM to generate false information, and that fact-checking is crucial when using the software. Including this disclaimer allowed us to uphold ACM Principle 1.3: Be honest and trustworthy [2]. Specifically, where 1.3 states, "A computing professional should be transparent and provide full disclosure of all pertinent system capabilities, limitations, and potential problems to the appropriate parties," [2].

There may be cases where a student claims an LLM-LGP-generated slide deck as their own intellectual property. This, by proxy, violates the section of ACM Principle 1.3 that states "making deliberately false or misleading claims… [violates] the code," [2]. While we were unable to implement this into our final product, we recommend that a watermark be included at the bottom of each slide that states the deck was AI-generated using the LLM-LGP tool. It is out of our control whether a user chooses to remove this watermark, but we will have nevertheless done our due diligence to be transparent about the origins of the slide deck.

Generated lectures must be accessible to all, such as those with dyslexia, color blindness, and impaired vision. This is to be in accordance with ACM Principle 1.4: Be fair and take action not to discriminate [2]. To do this, we ensured the slides use a large font that is friendly to those with dyslexia, and ensured text and images are colored in a way that has high contrast. The text is additionally screen-readable for those utilizing screen-readers.

The Michael Davis framework is an ethical decision-making framework that assists in guiding individuals to make ethical decisions [4]. By applying the Michael Davis tests to the LLM-LGP, we can uncover more ethical risks associated with the project, and create plans to mitigate them. The primary issue at stake is that students need a way to effectively summarize course material. Stakeholders include students, professors, and other educational professionals. The choice we made is to develop the LLM-LGP in order to address this issue.

- Harm test: Does this option do less harm than any alternative?

We have the potential to hurt students' ability to learn through over-reliance on the LLM-LGP. Critical thinking and synthesis are critical skills in academia, and those skills may suffer for those who rely on this software. In order to mitigate this, we encourage students to engage with the generated summary, validate the information within it, and use it to determine strong and weak points within their own learning. There is a balance between technology as a tool and the development of essential learning skills.

- Professional test: What might my profession's ethics committee say about this option?

From the perspective of the teaching profession, some professors will be more resistant to AI tools than others. At Colorado School of Mines and many other universities, there are specific guidelines outlined by the institution for the use of generative AI [5]. Education professionals understand that the use of GenAI amongst students is inevitable. Tools such as the LLM-LGP are not going away, and it is up to each institution to decide what policies and guidelines will be put in place.

X. Results

The project completion status is with respect to the functional and non-functional requirements outlined in sections II and III, and the initial project description given by HiLabs, which is as follows:

- Utilizing LLMs to interpret and analyze educational content, extracting key themes, concepts, and insights
- Developing algorithms that can structure these insights into concise, informative minilectures, tailored to different learning styles and levels
- Integrating NLP techniques to ensure the generated content is coherent, engaging, and pedagogically sound
- Creating an API or interface to enable seamless integration of this system into our existing platform, allowing educators to easily customize and deploy mini-lectures

Features implemented

- The pre-trained, open-source, locally-run Ollama LLM is utilized to produce an easily digestible lecture.
- User specifications are retrieved from the API in order to generate a prompt based on a premade template. The user specifications allow for adjustment to current knowledge level, desired complexity, and whether the lecture should provide more information or elaboration on a specific topic. The premade template instructs the LLM to use slide, title, and bullet-point delimiters that are easily ingested by the python-pptx library.
- Only PDF files can be uploaded by the user.
- All functions are wrapped in an API that, as validated through testing, can be called using JavaScript and TypeScript.
- Vector stores were implemented through LangChain in order to limit LLM context and improve performance.
- Testing was performed using open-source educational content, such as through LibreTexts and MIT OpenCourseWare.
- The generated lecture can be downloaded by the user as a .pptx.
- A functional front-end interface, but not web-deployable nor visually sophisticated.

Features not implemented

- The user cannot interface directly with the LLM at this time. This was a primary goal and is not difficult to implement on its own (simply pass the user-generated prompt instead of the premade one). However, the user breaking the program (e.g. instructing the LLM to use different delimiters for the slides) was a greater concern that we could not mitigate.
- Different files such as .docx, .doc, or .pptx cannot be uploaded. Each of these file types can be easily converted to a .pdf, so we do not expect this to be an immediate issue.
- While functional with JavaScript and TypeScript, the API has not been tested for integration with HiLabs' system. As outlined in section V, we did not receive access to the existing codebase over the duration of this project.
- Natural Language Processing was not used to test whether each lecture was pedagogically sound.

Based on our ability to implement most of the key functionality of this project, our team feels we were effectively able to meet the minimum viable product requirements.

XI. Future Work

Despite the MVP requirements being met, there are still more aspects that can be improved upon or added to the LLM-LGP. Our project contains an extensive readme for any future developers to be able to continue working, and Images are an important aspect of captivating and informative slideshows. While our product produced a slideshow with valid and

comprehensive text, adding relevant images was beyond the scope of our timeframe. In the future, we recommend that images be incorporated into the PowerPoints that the user receives. One possible approach to this requires an AI that is able to interpret the preexisting images that are in the material (such as textbooks) that the user uploads, and cut and paste these images into relevant spots in the slideshow. This would require an AI with image interpretation capabilities, the ability to implement this into the code base, and the ability to insert an image into the generated presentation and adjust the size without any corruption. We estimate the duration of this project would be one month, and would require access to an LLM with image processing capabilities, as well as the computational power to run this LLM. Despite the time and computational costs, this would result in a significant change in the quality of the slide deck that the LLM-LGP produces.

An additional recommendation pertains to enhancing LLM processing time by improving upon its infrastructure. In keeping with the ethical considerations of this project, we have implemented citation generation within the slides. This has driven the generation time to 2-3 minutes per slide. This can be greatly improved upon through a project with a higher focus on making improvements to the system architecture. Required knowledge and skills may include an understanding of machine learning, parallel processing techniques, and LLM architecture. We estimate that this could be a 3-month project.

We recommend that additional document loaders be explored in the future. Currently, the LLM-LGP is only able to parse PDF materials, which primarily restricts the user to online textbook content and slideshows. Opening the project to allow for YouTube transcripts (if a professor uploaded a video of a lecture) or HTML files (where content is on websites such as OpenStax or LibreText) could broaden the possibilities for educational content users can select from. We estimate that this could be a 2-month project.

XII. Lessons Learned

The struggles our team faced throughout the development of the LLM-LGP provided each of us with valuable software engineering skills and experience. We learned a few key lessons in working together this semester.

We have learned it is absolutely critical to document continuously throughout a project. Not only does this allow for a reduction in stress at the end of the project, but it helps ensure all collaborators have the resources to understand the entire project. If team members are left in the dark during the development of individual elements, integration of those elements becomes much more challenging.

Test Driven Development is more effective than testing after all individual parts are completed, and results in higher-quality code. This includes testing all of the components together as you create them. Creating them and then attempting to put them together results in unclean and often broken code. Testing everything together from the beginning takes less time than troubleshooting at the very end.

XIII. Acknowledgements

We would like to extend special thanks to our client Loc Hoang for his valuable technical advice throughout the duration of this project. The suggestions you provided for the blocks we faced allowed the project to run smoothly. Your flexibility and patience were immensely appreciated. Each one of us has benefited from your insight, and we sincerely hope our work on this project has in turn benefited HiLabs' mission.

We would additionally like to thank our faculty advisor Donna Bodeau for her guidance. This project would not be what it is today without your attention to detail and design expertise. With your help, we have all walked out of this class not only better programmers, but more competent and well-rounded engineers.

XIV. Team Profile

Gabrielle Christensen
Senior
Computer Science
Highlands Ranch, CO

Peter Hoila
Senior
Computer Science
Lakewood, CO

Grace Corpron
Senior
Computer Science
Focus in Robotics and Intelligent Systems
Portland, OR

Jessica Engel
Senior
Computer Science
San Antonio, Texas

References

[1] "pytest: helps you write better programs," pytest, https://docs.pytest.org/en/7.4.x/ (accessed Oct. 22, 2023).

[2] “ACM Code of Ethics and Professional Conduct,” Association for Computing Machinery, https://www.acm.org/code-of-ethics (accessed Oct. 22, 2023).

[3] W. Davis, “Sarah Silverman is suing openai and meta for copyright infringement,” The Verge, https://www.theverge.com/2023/7/9/23788741/sarah-silverman-openai-meta-chatgpt-llama-copyright-infringement-chatbots-artificial-intelligence-ai (accessed Oct. 22, 2023).

[4] M. Davis, “Seven Step Method for Ethical Decision-Making,” Online Ethics Center for Engineering and Science, https://onlineethics.org/cases/seven-step-method-ethical-decision-making (accessed Oct. 22, 2023).

[5] “Guidelines for using Generative Artificial Intelligence at mines,” Colorado School of Mines, https://www.mines.edu/academic-affairs/genai/ (accessed Oct. 22, 2023).

Appendix A

Table 1: Key terms

Term	Definition
Application Programming Interface (API)	A software interface that allows two or more computer programs to communicate with each other.
Large Language Model (LLM)	An AI system capable of understanding and generating human-like text.
Natural Language Processing (NLP)	A branch of AI concerned with studying and developing how machines understand written and spoken word.
Parsing	The process of analyzing a document and transforming it into a format (e.g., text) that software can interpret.
Pytest	A testing framework for the Python coding language, that allows users to implement tests for their code.
Test-Driven Development	A software development process entailing the

	creation of unit tests for every software aspect before its development, and continual and constant testing as the software grows.

Installation instructions:

Install Flask to run the GUI from the Python file.

- You can install it using ```pip3 install Flask```
- Run the GUI using ```flask run``` in the project home directory.

Install Ollama

Getting Ollama to work with the API can be difficult depending on your system. Your local machine must have an Nvidia chip if it is a Windows machine. For Macs the project can run on any Macbook, however, you will not get satisfactory token speed unless you have at least 8 GB of RAM to run the 3B models, 16 GB to run the 7B models, and 32 GB to run the 13B models.

You should be able to install Ollama simply by following these directions
https://github.com/jmorganca/ollama

Parsing Documents

To read content from the directory ./Documents run content_reader.py using ```python3 content_reader.py```

Creating a PowerPoint

When you run content_reader.py it will store the result in response.txt. Then that file is read by the PowerPoint generator which you can run using ```python3 pptx_creator.py```