THE DEVELOPMENT OF LLM TOOLS FOR GENERATING EDUCATIONAL ANIMATIONS IN MATHEMATICS

BY

ELAINE CHUNG HUI LIN

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

JUNE 2025

COPYRIGHT STATEMENT

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my supervisor, Prof. Ts. Dr. Liew Soung Yue, for his invaluable guidance, encouragement, and unwavering support throughout the course of this project. His expertise and insightful feedback have been instrumental in shaping my understanding and approach, allowing me to grow academically and professionally. His mentorship has provided me with the confidence and motivation to strive for excellence and to pursue knowledge with greater determination.

I would also like to extend my heartfelt appreciation to Chee Henn, for his continuous support and practical guidance during this project. His willingness to share knowledge, offer constructive suggestions, and clarify doubts has been a great help. His assistance have ensured the smooth progress of my work, and I am truly grateful for his generosity and mentorship.

Finally, I extend my heartfelt gratitude to my parents and family for their love, support, and encouragement throughout my academic journey. Their sacrifices, patience, and unwavering belief in my abilities have provided me with the strength to overcome challenges and persevere through difficulties. Their unconditional support has been the foundation of my achievements, and this milestone would not have been possible without their guidance, inspiration, and trust, which has continuously motivated me to pursue excellence and strive toward my goals.

ABSTRACT

Educational videos have become a crucial content-delivery tool in all levels of education, particularly in blended and online learning environments. Creating high-quality educational animations for mathematics instruction remains resource-intensive and technically challenging, limiting educators' ability to produce engaging visual content for blended and online learning environments. Traditional animation workflows require specialized technical skills and substantial time investment, creating barriers between pedagogical expertise and effective content delivery. This project introduces a novel Domain-Specific Language (DSL) that enables Large Language Models (LLMs) to automatically convert natural language descriptions of mathematical concepts into structured animation commands. The core innovation lies in bridging natural language input with precise animation instructions, allowing educators to generate visualizations for complex topics such as function graphing, geometric proofs, and calculus derivatives using simple text descriptions. The system architecture integrates GPT-4o for intelligent content interpretation, a custom TypeScript animation library supporting mathematical visualizations and transformations, and Google Text-to-Speech for synchronized narration. Users input high-level prompts describing mathematical concepts, which the LLM processes through the DSL to generate modular, reusable animation scripts that render as web-based visualizations. This work contributes a reproducible framework for AI-driven educational content creation that democratizes animation production for mathematics instruction. The modular architecture and open DSL design provide immediate practical value for mathematics educators while establishing scalable foundations for broader STEM disciplines. The system represents a significant advancement in the intersection of artificial intelligence, computer graphics, and educational technology. By lowering the barriers to educational animation creation, this project aims to enhance the learning experience for mathematics students, with potential extension to broader STEM discipdlines.

Area of Study: Educational Technology, Natural Language Processing

Keywords: Large Language Models, Artificial Intelligence, Animation Systems, Mathematical Visualization, Text-to-Animation, Domain-Specific Language

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
ANTLR4	Another Tool for Language Recognition 4
API	Application Programming Interface
AST	Abstract Syntax Tree
CAD	Computer-Aided Design
CLT	Cognitive Load Theory
DAG	Dynamic Acyclic Graph
DSL	Domain-Specific Language
EEE	Electrical and Electronics Engineering
EMA	Explanatory Math Animations
EPUB	Electronic Publication
GPT	Generative Pre-Trained Model
gTTS	Google Text-to-Speech
IIFE	Immediately Invoked Function Expression
IR	Intermediate Representation
LLM	Large Language Model
MANIM	Mathematical Animation Engine
NLP	Natural Language Processing
npm	Node Package Manager
OOP	Object-Oriented Programming
PDF	Portable Document Format
RAG	Retrieval-Augmented Generation
RLHF	Reinforcement Learning from Human Feedback
STEM	Science, Technology, Engineering, and Mathematics
UML	Unified Modeling Language

Chapter 1

Introduction

This section outlines the rationale behind the project to develop tools for Large Language Models (LLMs) to generate educational videos. It identifies current challenges in creating animations for complex STEM topics and the motivation to create an improvised tool that enables users to develop high-quality animations.

With the rapid advancement of generative AI, new possibilities have emerged across many industries, including education, healthcare, entertainment, and business operations. This emergence of new technology is transforming how we perceive, acquire, and process information. At the same time, video animation has revolutionized education. It has expanded the creative possibilities for presenting and supporting educational content. This trend in animation-based learning has been further enhanced by the integration of augmented and virtual reality technologies.

Young people today are used to receiving and processing information in brief, rapid segments, rather than in longer, more sustained forms. They prefer, and are adept at understanding concise, visual, and rapidly presented information [1]. In a study [2], researchers examined the effectiveness of animated videos as educational tools in engineering drawing courses. Their aim was to improve students' imagination and visualization abilities. Results showed that compared to conventional text-based instruction, animated videos enhanced students' comprehension, motivation, and engagement with the subject matter.

The multimedia elements supplement traditional textbooks and lectures, which help reduce students' cognitive load while improving their ability to visualize and imagine complex objects. Therefore, there is a growing need for tools that facilitate the creation of engaging and effective educational animation presentations. The focus of this paper will be on developing a TypeScript-based library that enables LLMs to generate animated educational content, making it easier for educators to create high-quality instructional videos. This approach aims to enhance learning experiences, improve accessibility, and reduce the technical barriers associated with animation development.

1.1 Problem Statement and Motivation

Creating animations for Science, Technology, Engineering, and Mathematics (STEM) education remains a significant challenge in the modern world. STEM concepts are often abstract and complex. STEM concepts are often abstract and complex. Translating these abstract ideas into visual representations that are both accurate and easy to understand requires deep knowledge of the subject matter, creativity in visualization, and proficiency in programming. As the content becomes more complex, educators from non-animation design disciplines may face a steep learning curve to create useful scientific animations for education in these domains. For instance, extensive training sessions were needed for secondary school teachers to create animated graphs using a math software, Maple [3]. This technical difficulty can be overwhelming for many educators who may have strong STEM knowledge but limited animation skills. While it's not entirely unusable for those outside its primary subject areas, their ability to fully utilize the library is restricted without a more comprehensive understanding [4]. The time required to be familiar with programming may also detract from the primary goal of creating high-quality mathematical animations. Although practical, the trade-off between time consumption and maintaining the animation quality can be significant.

In addition to technical challenges, financial barriers further complicate the production of educational animations. Proprietary animation software and tools can be expensive. While these paid options are highly secure and stable with built-in assets and professional features, they often come with higher system requirements, making them less favourable. Additionally, they often necessitate licensing and user training [5]. Although open-source alternatives are available, they tend to be limited in scope, offering only a fixed set of examples with minimal adaptability and creative potential. Therefore, a new approach should be explored to enable educators with minimal experience in programming or animation to visualize their teaching materials using modern tools.

The significance of video animation in education cannot be overstated. Animation is a powerful medium for communicating scientific and mathematical concepts [6]. They make complex concepts easier to understand by providing visual representations that cater to different learning styles. Difficult topics can be broken down into simpler, more digestible segments through animation. Additionally, animated videos are highly accessible, fostering remote and self-paced learning on a global scale. Once created, video animation content can also be reused, thereby reducing educational costs in the long run.

Educational animation is effective across all ages, and many educators are constantly seeking new methods to help students visualize complex, abstract ideas, especially given the changing needs and attitudes of students after the COVID-19 pandemic. Learning materials need to be engaging, flexible, and adaptable to suit various learning styles and situations. Simply presenting information as text or web pages can overwhelm students [7]. For instance, a study found that undergraduate Electrical and Electronics Engineering (EEE) students generally struggle to understand the curriculum due to its mathematical, abstract, and theoretical nature. Many students respond better to narrated animations than mathematical proofs and equations, but often, instructors are limited to providing static graphs and diagrams [7]. As a result, students are left to create their own mental representations of the concepts, which can be intimidating.

One of the key considerations when designing educational materials, including video content, is to balance cognitive load [8]. Cognitive load refers to the amount of mental effort required in working memory. To enhance information processing and improve learning outcomes, it's crucial to reduce the cognitive burden on working memory. Cognitive load theory (CLT) posits that human's working memory has a limited capacity for processing information. There are three types of cognitive load in learning, that is intrinsic load, extraneous load, and germane load. Details of each type of cognitive load can be found in Table 1.3.1.

Instructional design aims to prevent cognitive overload by managing these loads effectively. Well-designed multimedia element can help achieve this goal by presenting information in multiple formats, engaging both visual and auditory senses, which can distribute cognitive load across different processing channels. When multiple senses are engaged, information retention and understanding can be enhanced.

In the context of learning, animated videos can significantly reduce cognitive load required to mentally visualize complex 3D objects from 2D representations, freeing up mental resources for deeper understanding and concepts. Additionally, compared to traditional textbook-based learning, multimedia learning relieves learners from spatial-temporal constraints. Multimedia learning is asynchronous, which does not require students and instructors to be present at the same time. Therefore, to optimize learning experiences, this project proposes the development of tools for LLMs to generate educational animations with minimal human intervention.

Table 1.1.1 Cognitive Load Theory (CLT)

Types of cognitive load	**Definition**	**Load management**
Intrinsic load	Inherent difficulty of the topic under study, regardless of how it is presented, and is under no influence of extraneous load and germane load.	- Break complex topics into manageable parts - Incorporate interactive and self-paced learning
Extraneous load	Cognitive effort that is ineffective in helping the learner accomplish the desired learning outcome. This type of load results from poorly presented lessons but may also result from factors like stereotype threat or the imposter syndrome	- Filter out non-essential tasks - Present only information that is relevant to the material
Germane load	Level of cognitive activity and mental effort required to use memory and intelligence to integrate new information	- Provide concrete visualizations of abstract or complex concepts

1.2 Objectives

1.2.1 Main Objective

This project’s primary objective is to **develop and integrate animation tools that can be controlled by LLMs**. The goal is to harness the capabilities of LLMs to create animations tailored specifically for educational content in Mathematics. These animations will be designed to visualize complex concepts and illustrate step-by-step processes in a manner that enhances comprehension and engagement for learners.

1.2.2 Sub-Objectives

To achieve the main objective, the project is structured into several sub-objectives.

First, a **modular animation pipeline** will be developed to ensure flexibility and reusability. This involves designing individual modules for shape transformations, motion generation, and scene composition. Each module will have well-defined APIs that can be easily interpreted and controlled by an LLM, allowing for a seamless and structured animation process. The modular approach will enable customization and scalability, making it easier to extend the system with additional features in the future.

Second, a **motion generation system** will be implemented to produce smooth and natural-looking animations. This system will incorporate algorithms that generate fluid motion paths, ensuring visually appealing transitions. A library of common animations, including shape transformations, creation, interpolation, and morphing, will be developed to provide a foundation for automated animation generation. These predefined animations will serve as building blocks, enabling LLMs to generate complex animations efficiently.

Lastly, an **application** will be developed to demonstrate the capabilities of the animation tool. This application will serve as an interface that integrates the modular animation pipeline, allowing users to input educational content and natural language prompts and receive corresponding animations. By providing an interactive platform, the application will showcase the tool's ability to generate high-quality animations in an intuitive manner. This will not only validate the system’s effectiveness but also provide a user-friendly way for educators and students to create educational animations with minimal effort.

1.3 Project Scope

The primary scope of this project is to streamline the creation of educational content, particularly within the mathematics discipline. By the end of the project, a tool will be developed for Large Language Models (LLMs) to create high quality, resourceful, and professional educational animations. To showcase the tool’s capabilities, a web application will be created for demonstration. While the application itself will not require extensive user interface design, the emphasis will be on ensuring the quality of the generated animations. The study intersects multiple domains, including LLMs, educational animation, machine learning, and information visualization, computer graphics and multimedia processing.

In addition to these advancements, the project will focus on enhancing the tool's efficiency, enabling video generation with minimal resources while achieving high accuracy. The focused area will be mathematics, with the objective of developing students' mathematical reasoning, problem-solving skills, and foundational knowledge for more advanced studies. The main presentation styles will include problem-solving tutorials, detailed conceptual explanations, and step-by-step visualizations of processes and algorithms. The goal is to help students build a strong understanding of mathematical topics like geometry and algebra, giving them a solid foundation for more advanced learning. By making the content clear, organized, and engaging, the project aims to make mathematics education more effective and easier to access.

1.4 Contributions

The development of AI-powered animation tools has the potential to reshape the educational landscape and democratize access to high-quality learning materials. By addressing the high costs and technical barriers traditionally associated with animation video production, the project empowers educators—especially those outside well-funded institutions—to create engaging, effective educational content. This shift promotes educational equity by enabling more voices to contribute to the global pool of learning resources.

Leveraging the capabilities of LLMs, the system automates complex tasks such as scriptwriting, animation, and voice-over generation. This intelligent automation reduces the need for specialized technical expertise, making the creation of animated videos more accessible and significantly more affordable. Educators can now focus on pedagogy and content, while the system handles the production—lowering time, cost, and effort.

By allowing users to describe educational concepts in natural language, the project introduces an intuitive interface that simplifies the content creation process. This not only enhances user experience but also opens up opportunities for personalized, self-paced learning—where students can explore challenging topics at their own rhythm, reinforcing understanding and long-term retention.

The project fosters innovation and sustainability in educational technology by making animation production scalable and adaptable to a wide range of disciplines and audiences. Through automation, accessibility, and intuitive design, it drives forward the vision of inclusive and lifelong learning, contributing to a future where high-quality education is not limited by resources but expanded by creativity and technology.

1.5 Report Organization

This report is organized into seven chapters: Chapter 1 Introduction, Chapter 2 Literature Review, Chapter 3 System Methodology and Approach, Chapter 4 System Design, Chapter 5 System Implementation, Chapter 6 System Evaluation and Discussion, Chapter 7 Conclusion and Recommendation.

The first chapter introduces the project by outlining the problem statement, project background and motivation, scope, objectives, project contribution, highlights of achievements, and the overall report organisation.

The second chapter presents a literature and system review of existing works, evaluating their strengths and weaknesses. It also compares these systems with the proposed solution, highlighting the shortcomings of prior studies and justifying the need for a novel approach.

The third chapter discusses the system methodology and approach, covering the overall system architecture, use case diagrams, and activity diagrams, which together provide an understanding of the system flow and functionalities.

Chapter 4 focuses on system design, detailing the system block diagram, component specifications, and system data flow.

Chapter 5 describes the system implementation, including the hardware and software setup, configuration of the development environment, project settings, and environment variables. It also includes the system's end-to-end operation with screenshots and discusses the challenges encountered during the implementation phase.

Chapter 6 evaluates the system through testing and performance metrics, presenting test outcomes, discussing challenges faced, and assessing the extent to which the project objectives were achieved.

Finally, Chapter 7 concludes the project with a summary of contributions and provides recommendations for possible future improvements.

Chapter 2

Literature Review

This chapter is divided into two sections. Section 2.1 and 2.2 examines existing methods and approaches for creating educational video animations. Section 2.3 evaluates and compares these methods, highlighting their strengths and weaknesses.

2.1 Literature Review on Previous Works

2.1.1 Manim Library

Demonstrating the Potential of Visualization in Education with the Manim Python Library: Examples from Algorithms and Data Structures [4]

The paper covers the design and implementation process of visualizations using the Community Manim Python library [11], incorporating feedback and evaluations from both students and the instructor. One significant drawback highlighted in the paper is that generating new animations in real-time, such as during a live classroom setting, is often not feasible due to the rendering process. The slow rendering speeds of Manim not only impact its use in live environments but also significantly affect production time. Educators and content creators may need to go through several iterations before achieving a satisfactory output, which can be time-consuming and frustrating.

Furthermore, despite its mathematical capabilities, Manim lacks built-in functionality specifically tailored for animating common computer science constructs. Educators had to create custom implementations for visualizing data structures, such as binary trees. While this allows for flexibility, it also means that educators must invest significant time in developing the basic building blocks creating more complex visualizations related to algorithms and data structures.

The paper also highlights that while Manim is ideal for fields such as mathematics, physics, computer science, data science, economics, and statistics due to its mathematical representation capabilities, its complexity can be a barrier for educators in other fields. The ability to manipulate animations on the fly and show the relativity of topics is a strength, but it comes at the cost of accessibility for those unfamiliar with Manim's core concepts. To resolve this issue, the library should be further developed to be more accessible to educators who may not have strong backgrounds in mathematics or computer science. This would make the advanced features more user-friendly for a broader range of academic disciplines.

The researchers analysed the effectiveness and usability of the system by gathering feedback from two groups of students. Average students found the visualizations to be essential to their learning process, as they would otherwise require more effort to understand the course material. In contrast, high-achieving students acknowledged the usefulness of the visualizations but preferred to draw their own visual representations. This suggests that further research may be needed to determine how to best serve both groups.

2.1.2 Manim-based EMA System

Autogeneration of Explanatory Math Animation [12]

This paper presents a prototype technology for automating the generation of Explanatory Math Animations (EMA). The authors highlighted a limitation in digital learning, where math teachers often need to create multiple videos for similar types of problems. Despite this effort, the videos still fall short of fully covering all the necessary material in a lesson. To address this issue, the researchers proposed a prototype that incorporates the use of Manim, a Python-based tool, to automate EMA production, specifically for K-12 education. The EMA system uses predefined algorithms and templates to generate visual explanations and step-by-step solutions based on user inputs. The system framework consists of five components: the user interface, database, problem classification, problem-solving tool, and explanatory module. The figure below shows the system framework design and process flow for EMA auto-generation. System inputs are mathematical problems taken from users. Once the system receives input questions, it classifies them. Problem-solving tools then take user variables and output data required by the explanatory module. The explanatory module is responsible for fetching this data and generating detailed explanation text, voice-over and video output.

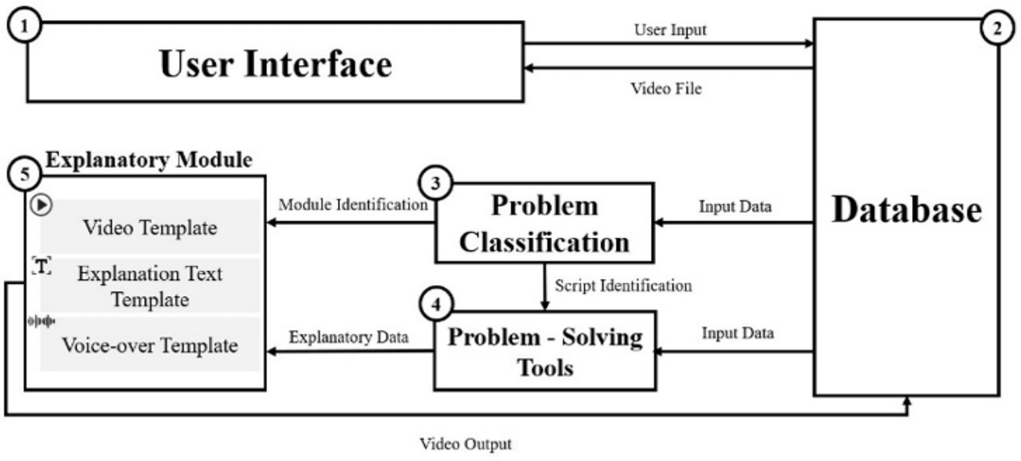


Figure 2.1.1 System framework design and process flow for EMA auto-generation

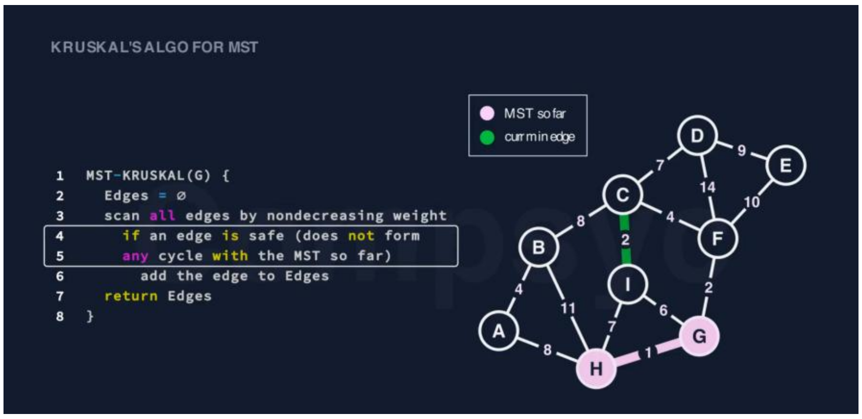
According to their results and conclusions, the auto-generated animations showed better quality in dynamic presentations compared to manually created videos. This approach is novel and time efficient, also reducing human involvement. However, there are still areas for improvement, such as adding more templates and utilising more sophisticated machine learning algorithms to classify problems more accurately. Furthermore, enhancing the explanatory module to produce dynamic templates for different classes of questions would be beneficial.

2.1.3 AAnim Library

AAnim: An Animation Engine for Visualizing Algorithms and Data Structures for Educators [13]

In this paper, the authors developed a Python library, AAnim based on Manim [11], for visualising computer science topics, specifically for data structures and algorithms. Their library makes significant improvements by addressing the limitations found in both the Manim Library and EMA generation. AAnim allows content creators to produce videos without needing specialized expertise or prior knowledge of Manim. While earlier approaches required the manual design of data structure layouts, AAnim automates this process, dynamically generating the layout based on input data, thereby reducing the creators' workload significantly. The AAnim library takes queries of list of list of data structures and outputs a video file demonstrating the evolution of data structures with its corresponding changes in the algorithm's pseudocode.

Figure 2.3.1 below shows a snapshot of one of their videos. A guided walkthrough of the pseudocodes line by line helps viewers visualise each step in the data structure depicted on the right. However, one downside of their implementation is the lack of voice-over. All explanations are provided solely through animation. This absence can make it more challenging for some viewers to fully grasp the concepts, as they may find it harder to follow along without an audio narration to complement the visual elements. Incorporating a voice-over component would make the explanation more comprehensive and easier to follow, especially for viewers who may struggle with reading the text quickly or interpreting the animated pseudocode walkthrough.



KRUSKAL'S ALGO FOR MST

```
MST-KRUSKAL(G) {
  Edges = ∅
  scan all edges by nondecreasing weight
    if an edge is safe (does not form
    any cycle with the MST so far)
      add the edge to Edges
  return Edges
}
```

Figure 2.1.2 Walkthrough of pseudocode with visualizations of the data structure [14]

While the AAnim library solves some significant limitations of previous works, it has its own drawbacks in live environments. The time-consuming rendering process also slows down content creation, as users often need multiple attempts to achieve the desired results. Adding a quick-view feature to the Manim tool could help address these issues. This feature would let users see a draft of their animation immediately, enabling them to make changes and improvements more quickly without having to wait for the entire animation to finish rendering [13].

2.1.4 dsDraw

dsDraw: Programmable Animations and Animated Programs [5]

In this paper, the authors introduced dsDraw, which is a lightweight web-based tool designed for instructors to create whiteboard-style animated lectures with synchronized voice narration. It uses a C-like programming syntax to create high-quality visualisations. Users can generate graphics via mouse, keyboard, or console commands allowing them to produce animations using pre-built tools and eliminating the need for manual coding. This makes the application more user-friendly as compared to previous approaches. The resulting videos are exportable to common formats.

The main motivation behind dsDraw is to address limitations in some of the most common web-based tools for creating lecture content. Other web-based tools such as LectureScribe [15], VisuAlgo [16], and Algorithm Visualizer [17] are limited to predefined data structures. In contrast, dsDraw is more powerful and flexible, capable of creating animations of arbitrary complexity. Key features of dsDraw include built-in functions to enhance the editing workflow, a program trace feature for step-by-step execution, a variable view to display current memory state, and audio recording functionality. dsDraw also offers great control over the animations. For example, the "wait" and "interpolate" commands can be used to align narration with animation. Despite being powerful and intuitive, the video animation creation process isn't fully automated. It requires thoughtful input and dedication to produce effective educational content. To resolve this limitation, several future improvements could be considered. Introducing features like template-based animation generation or AI-assisted animation creation would streamline the process, thus increasing automation.

2.2 Review of the Existing Systems/Applications

2.2.1 MathMatrixMovies

MathMatrixMovies [18] is an AI-powered tool designed to create personalized, animated math explainer videos based on simple text prompts. It leverages Google's Gemini Pro 1.5 language model and Manim, a popular math animation library, to break down complex mathematical concepts into engaging visual content. The creators use the Manim Voiceover plugin along with the Azure Text-to-Speech (TTS) service to generate narrations. Users can input a prompt, select an age range reflecting the educational level, for example preschool to graduate students, and choose preferred languages like English, Spanish, Hindi, or Tamil. This makes math more accessible and tailored to various learning levels. The figure below shows a workflow of how math questions is turned into educational videos.

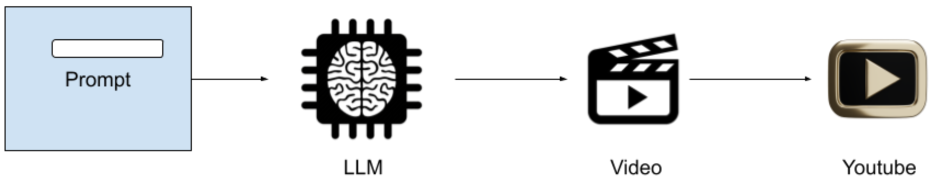


Figure 2.2.1 User workflow from prompt to video

First, the user inputs a math problem, and the system calls Gemini Pro with a predefined prompt format. The model then generates Manim code which is processed by the LLM. However, one limitation is that the Manim code may not compile successfully, requiring the LLM to be re-prompted along with the error text for further attempts. The number of retries can go up to eight times. Subsequently, two videos are created, which are the initial video and the final video. After the initial video is produced, it is presented to the user via the frontend, while in the background, the video is divided into 30 keyframes. Gemini is then used to iterate on the video to enhance quality. The need for iterations and repeated passes in generating videos with MathMatrixMovies indicates that the initial outputs may not always be reliable or consistent. This poses a limitation in real-world applications, leading to increased computational costs, as each retry demands further processing. This may also introduce challenges for users without access to high-end hardware or sufficient cloud resources.

To mitigate the issues related to retries and computational costs, several approaches could be explored. Firstly, improving the initial accuracy of the LLM when generating Manim code would reduce the need for repeated passes. This could be achieved by integrating more robust error-handling mechanisms directly into the LLM, ensuring that common errors in Manim code are anticipated and corrected before they occur.

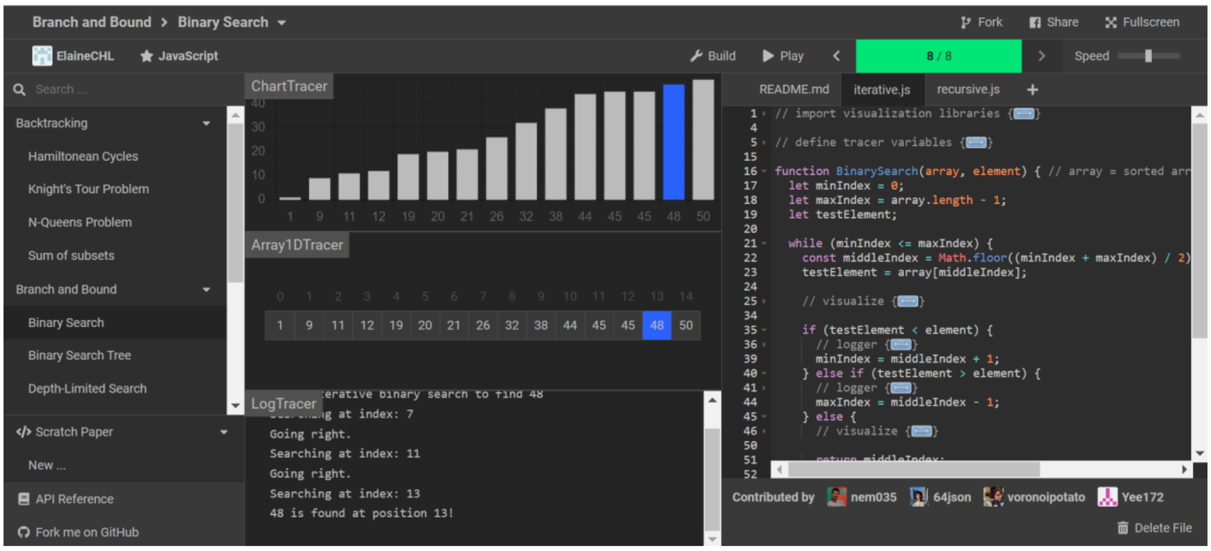
Another solution is to introduce pre-compilation validation, where the system checks for basic errors before running the full Manim rendering. This would catch obvious mistakes early, reducing the need for complete retries.

2.2.2 Algorithm Visualizer

Algorithm Visualizer [17] is a web-based tool developed by a community of contributors, led by "thomasthomas882" on GitHub. This open-source project aims to help users better understand algorithms, particularly sorting, searching and graph traversal algorithms, through interactive visualizations. The platform allows users to visualize these algorithms step-by-step in real time, either with randomly generated data or by inputting their own data. Users can modify parameters to explore different outcomes, enhancing their learning experience. Many other tools, like Sorting Algorithms Animations [19], only work with predefined data, which limits flexibility in experimentation.

Compared to static animated videos, the interactive nature of Algorithm Visualizer fosters active learning. Users have complete control over the material, such as stepping through each phase of the algorithm at their own pace. This promotes deeper engagement and leads to better retention and understanding. The tool also encourages hands-on experimentation, as users can input custom data to explore various scenarios and observe how different inputs impact outcomes. This hands-on approach fosters curiosity and critical thinking, making it more effective than passive video-based learning, which can be presentation-heavy.

Figure 2.2.1 shows the interface of Algorithm Visualizer solving a binary search problem. The right-hand panel displays the underlying code for the visualized algorithm, assisting users in linking the visual demonstration to the actual code implementation. Additionally, users can modify the code, adjust dataset parameters, and explore the inner workings of the algorithm. The tool also provides logging features, which chronologically record actions during the algorithm's execution. This helps users understand the intermediate steps and decisions the algorithm makes, offering valuable insights into its logic and data processing.



```
0   1   2   3   4   5   6   7   8   9   10  11  12  13  14
1   9   11  12  19  20  21  26  32  38  44  45  45  48  50
```

```
erative binary search to find 48
g at index: 7
Going right.
Searching at index: 11
Going right.
Searching at index: 13
48 is found at position 13!
```

```
// import visualization libraries {}

// define tracer variables {}

function BinarySearch(array, element) { // array = sorted arr
  let minIndex = 0;
  let maxIndex = array.length - 1;
  let testElement;

  while (minIndex <= maxIndex) {
    const middleIndex = Math.floor((minIndex + maxIndex) / 2)
    testElement = array[middleIndex];

    // visualize {}

    if (testElement < element) {
      // logger {}
      minIndex = middleIndex + 1;
    } else if (testElement > element) {
      // logger {}
      maxIndex = middleIndex - 1;
    } else {
      // visualize {}

      return middleIndex;
```

Figure 2.2.2 Interface of binary search algorithm in Algorithm Visualizer

One of the key strengths of Algorithm Visualizer is its ability to break down complex algorithms into manageable, atomic steps, making it easier for users to grasp fundamental concepts. The tool's real-time interactivity and customizable features allow users to experiment freely, deepening their comprehension. Additionally, the built-in code editor connects the visualization to the actual implementation, making it an excellent resource for learning both theory and coding.

However, the tool has some limitations. For users unfamiliar with algorithm theory or coding, the learning curve can be steep, especially when it comes to modifying the underlying code or parameters. The scope of the tool is also somewhat limited to fundamental algorithms, which may not fully address the needs of advanced learners looking for more complex or specialized algorithms. To address these limitations, LLMs offer a powerful solution by simplifying the visualization process without requiring users to deal with any code. Over time, as LLMs evolve, they can also expand to cover a wider range of algorithms, including more advanced and specialized ones, ensuring that both beginners and advanced learners benefit from the video.

2.3 Summary of the Existing Systems

Table 2.3.1 Comparison between methods used in previous works

Method	Strengths	Limitations
Manim	1. Provides great flexibility for the creation of complex visualizations in various fields	1. Slow rendering speeds for real-time animation generation. 2. Lack of built-in functionality for common constructs 3. Too complex for educators outside of the animation field.
Manim-Based EMA System	1. Minimal human involvement 2. Offers high-quality, dynamic presentations. 3. Time-efficient and suitable for producing educational content at scale.	1. Lacks templates and uses basic algorithms for problem classification 2. Limited customization options for different types of questions
AAnim Library	1. Reduces workload on content creators by automating layout design and generating high-quality visualizations	1. Lacks voice-over functionality 2. Slow rendering process hinders its use in live environments
dsDraw	1. Accessible to users without coding skills 2. user-friendly with all essential features 3. Provides great control over animations, allowing highly customized and complex visualizations.	1. The animation creation process is not fully automated, requiring significant user input and dedication to produce effective educational content.
MathMatrixMovies	1. Automates creation of explainer videos from text input. 2. Supports multiple languages	1. Requires iterations for error handling, slowing down the process. 2. High computational costs

	and different educational levels. 3. Interactive and customizable.	
Algorithm Visualizer	1. Offers interactive, real-time visualization of various algorithms. 2. Highly customizable 3. Connects visual representation directly to the underlying code	1. Limited to fundamental algorithms, lacking coverage of more complex or advanced algorithms. 2. Lacks voice-over functionality

In summary, slow rendering times are a common limitation across tools that utilize Manim which are EMA system, AAnim Library, and Community Manim Library, impacting their use in live environments and overall production speed. On the other hand, dsDraw and AAnim are designed to be more accessible and user-friendly, which requires less technical expertise to manipulate.

Meanwhile, Algorithm Visualizer excels in interactive, real-time visualizations of algorithms, making it highly customizable. However, its limitations to fundamental algorithms and lack of voice-over functionality restrict its scope for more complex educational content. MathMatrixMovies, while automating the creation of explainer videos and supporting multiple languages and educational levels, faces challenges related to multiple retries for error handling in the Manim code generation process. This increases the computational costs and slows down video production, making it difficult for users with limited hardware to use efficiently. Despite these drawbacks, it stands out for its interactive and customizable nature, tailored to various learning levels and languages.

2.4 Proposed Solution

To address the limitations observed in existing tools such as Manim-based systems, Algorithm Visualizer, and MathMatrixMovies, this project proposes a lightweight, browser-based animation generation system powered by GPT-4 and a custom TypeScript animation library. Unlike traditional Manim workflows that require slow Python-based rendering and high computational resources, this solution enables real-time animation directly on the web using HTML5 Canvas.

By leveraging GPT-4o for natural language prompt interpretation and script generation, the system lowers the technical barrier for content creation and reduces manual effort. This can be done through the concept of Animation-as-Code, where animations are defined and controlled through programming scripts rather than visual interfaces. Beyond precision and reusability, this paradigm offers a critical advantage for integration with LLMs. Since LLMs are naturally adept at generating structured code-like text, they can easily produce animation scripts following a predefined domain-specific language (DSL). This eliminates the barrier of GUI-based tools, enabling automated and scalable animation generation directly from user prompts.

Voice narration is integrated using Google Text-to-Speech (gTTS), offering an end-to-end solution for generating explainer videos with synchronized audio. The custom animation library supports core primitives (e.g., shapes, text, motion, transitions) and is optimized for live rendering without external dependencies or plugins. This design ensures accessibility, fast production, and compatibility across devices, making the system suitable for educators, students, and content creators working with limited hardware or technical skills. Details of the proposed method will be discussed in Chapter 3.

Chapter 3

System Methodology and Approach

This section provides an in-depth look into the tools involved, methods and technologies used in the project, covering design specifications, and the project workflow.

3.1 System Design Overview

This system is designed to transform static educational content, such as textbooks, into dynamic, web-based animated videos using a Retrieval-Augmented Generation (RAG) architecture. It consists of two main pipelines, the **Preprocessing Pipeline** and the **Query and Generation Pipeline** as shown in Figure 3.1.1 below.

Knowledge Base and Retrieval Pipeline

The knowledge base and retrieval pipeline is responsible for ingesting and preparing raw learning materials in PDF format. This process involves cleaning mathematical expressions, removing headers, footers, and discarding irrelevant sections such as the preface. The cleaned content is then segmented into smaller, manageable chunks, such as chapters or sections to accommodate the token limitations of large language models (LLMs). These chunks are embedded into vector representations using OpenAI's text-embedding-3-small model. The resulting embeddings are stored in a Pinecone vector database, which functions as the LLM's long-term memory. At runtime, the database is queried to retrieve the most semantically relevant chunks in response to a user prompt. These retrieved contexts are passed to the LLM, enabling it to generate a tailored animation script grounded in the original textbook content.

Query and Generation Pipeline

The query and generation pipeline begins when a user submits a prompt (e.g., "Explain the Fibonacci sequence") through the user interface. The prompt is first converted into a vector embedding using the same model previously applied to the learning materials. The resulting vector is used to query the Pinecone vector database, retrieving the top N most semantically relevant content chunks. These chunks provide contextual information for the LLM. The LLM processes this context along with the original prompt and uses a custom animation library API to generate an animation script.

The LLM processes this context along with the original prompt and produces two synchronized outputs:

1. An **animation script**, expressed in a custom domain-specific language (DSL). This DSL, implemented in TypeScript, abstracts low-level animation logic and enforces structural constraints such as unique object identifiers and strict syntax rules. The final script is rendered using a web-based canvas renderer, producing an animation that visually explains the concept described in the user's prompt.

2. A **narration script**, written as natural language text that complements the visual explanation. This narration script is passed to a text-to-speech (TTS) engine, which converts it into audio voice-over.

By generating both the animation and narration in parallel, the pipeline ensures that the final output is a cohesive, synchronized multimedia explanation.

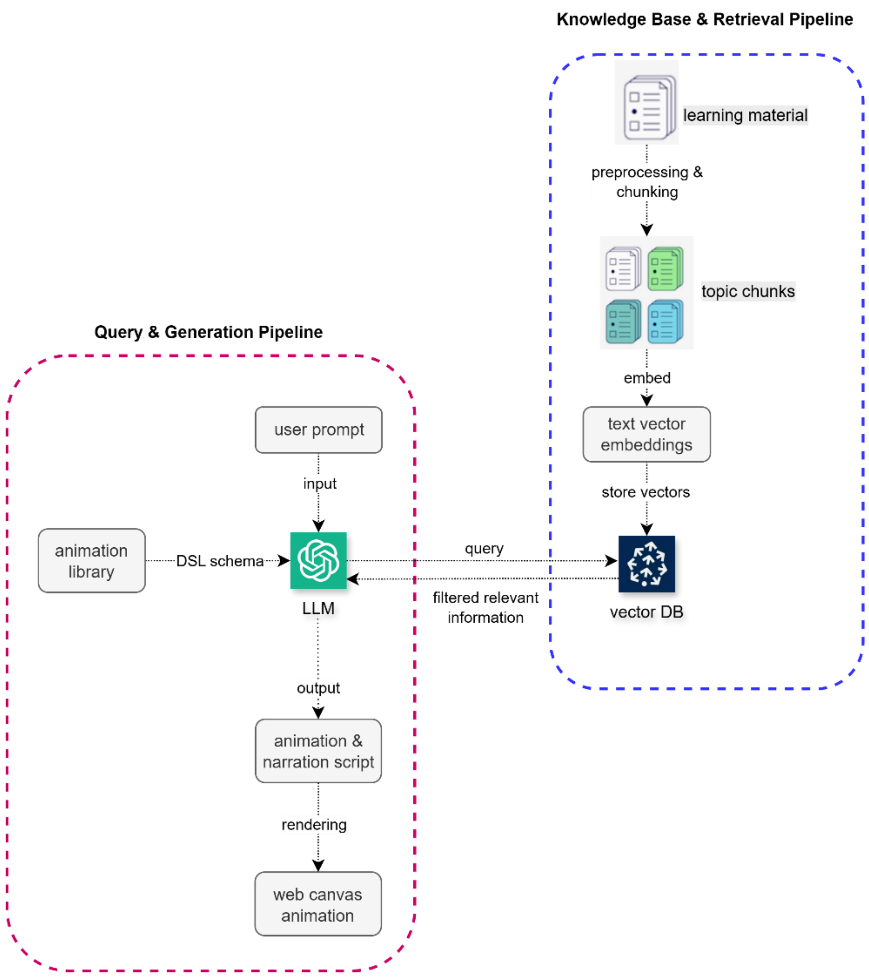


Figure 3.1.1 RAG-based Educational Animation Generation Pipeline

3.1.1 System Architecture Diagram

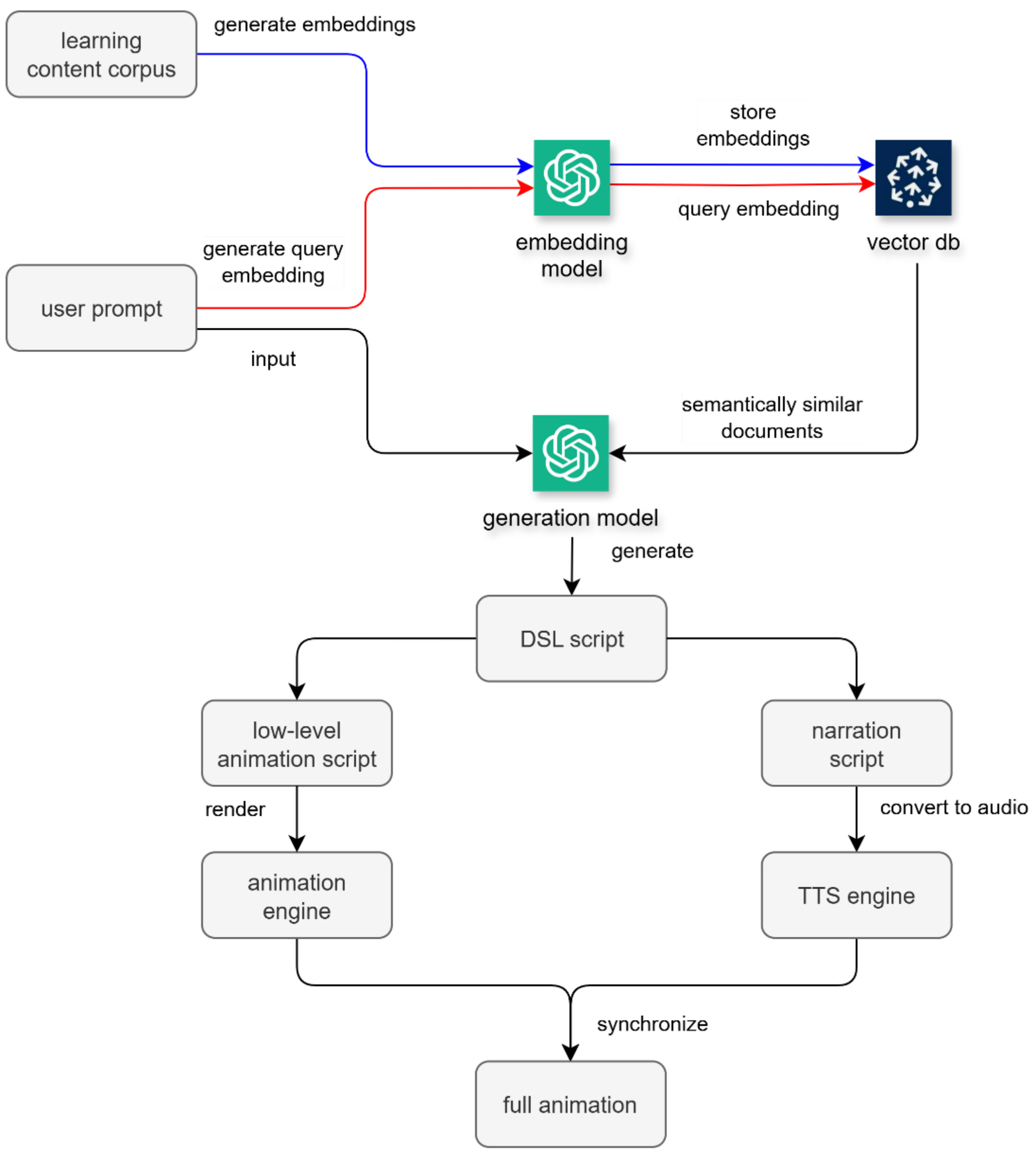


Figure 3.1.2 System Architecture Diagram

The proposed system architecture automatically generates animated educational videos from a high-level user prompt by integrating RAG, a domain-specific language (DSL), and text-to-speech (TTS) technologies.

The workflow begins when users input two things: learning material and a follow-up prompt. The learning material act as the learning content corpus and is pre-processed into vector embeddings, which are stored in a vector database. These embeddings enable efficient retrieval of semantically relevant content in response to user queries.

When a user submits a prompt, it is converted into a query embedding using the same embedding model and compared against the stored embeddings in the vector database. This process retrieves the top-N most semantically similar chunks of content from the corpus, ensuring that the generated output is grounded in accurate and contextually relevant knowledge. The retrieved context and the user's prompt are then inserted into a crafted prompt template then passed to the generation model.

The generation model produces a DSL script which contains both animation instructions and narration text. The DSL is specifically designed for educational content, allowing high-level concepts to be translated into precise visual and spoken explanations.

From here, the DSL script branches into two parallel processes. On one side, it is translated into a low-level TypeScript animation script, executed by the web-based animation engine to render shapes, movements, and transitions directly in the browser. On the other side, the narration text is converted into speech using a TTS engine. Finally, the animation and narration are synchronized to create a cohesive educational video.

By combining semantic retrieval, LLM-based generation, a purpose-built DSL, and a custom TypeScript rendering library, this architecture provides a complete pipeline for transforming textual prompts into engaging educational animations. Together, the DSL and the TypeScript animation library form the core innovations of the system: one bridges natural language and animation logic, while the other enables accessible, web-based execution of educational animations.

3.1.2 Methodologies and General Work Procedures

This project follows an **iterative and incremental development methodology**, allowing continuous improvement and refinement of the system components based on early feedback and testing outcomes. The development process was divided into distinct phases: requirements gathering, design specification, system implementation, testing, and refinement.

Firstly, initial requirements and objectives were defined through literature reviews and analysis of existing tools such as Manim, Algorithm Visualizer, and MathMatrixMovies. The system was then built incrementally, starting with the core animation engine in TypeScript and HTML5 Canvas, followed by integration with GPT-4 for prompt processing and gTTS for voice synthesis.

Each module was developed and tested individually before being integrated into the main application. Continuous testing was carried out after each feature implementation to ensure correctness, performance, and usability. Manual and functional testing methods were applied, with feedback incorporated throughout the development cycle.

Besides, version control was maintained using Git, and VS Code served as the primary local development environment. This methodology ensured the project stays aligned with its goals while remaining adaptable to evolving requirements or technical challenges.

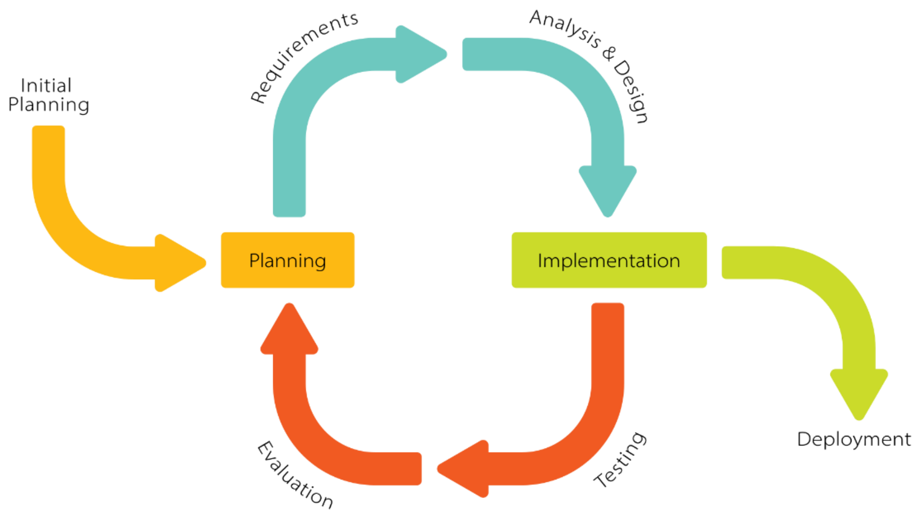


Figure 3.1.3 Iterative development model

3.1.3 Use Case Diagram

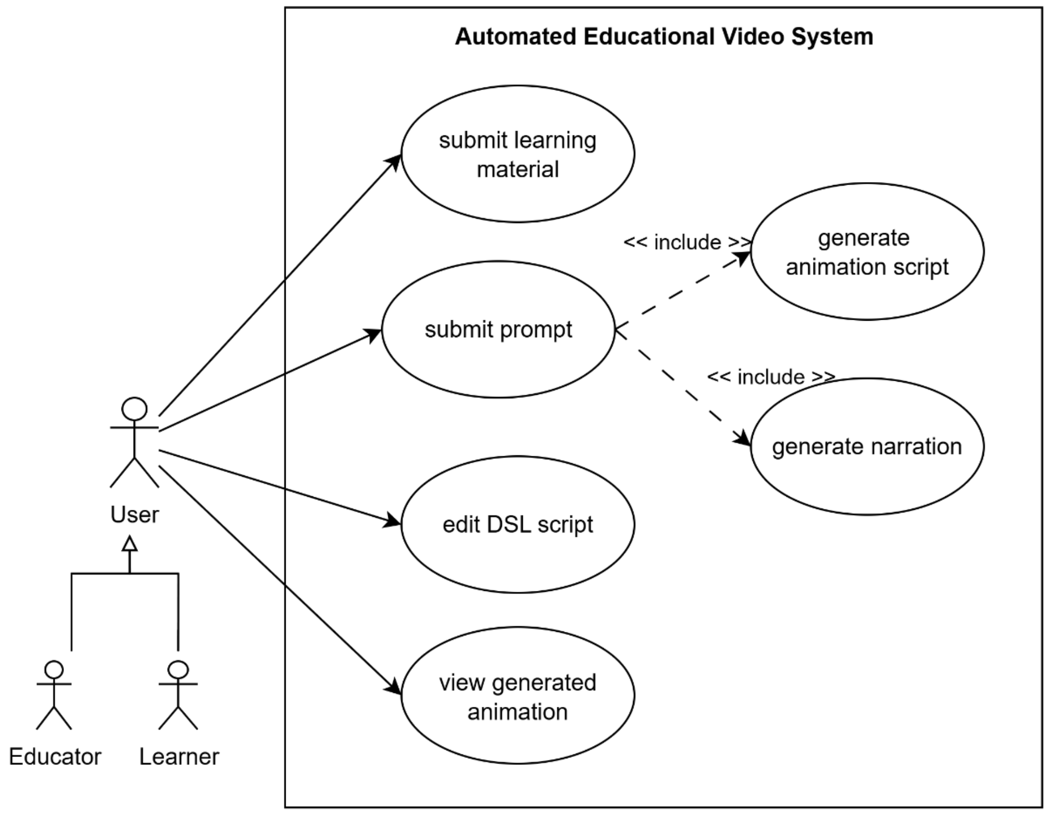


Figure 3.1.4 Overall Use Case Diagram of Automated Educational Video System

3.1.4 Use Case Description

3.1.3.1 Use Case Description for Submit Learning Material

Use Case ID	UC001	Version	1.0
Use Case	Submit Learning Material		
Purpose	To allow the user to provide a knowledge corpus that the system will preprocess and store for later retrieval when generating educational animations.		
Actor	User		
Trigger	The user uploads learning material into the system interface.		
Precondition	• The system is running and connected to the internet.		
Scenario Name	**Step**	**Action**	
Main Flow	1	The user selects the option to upload or input learning material (optional).	
	2	The system accepts the learning material in PDF format.	
	3	The system processes the material, splitting it into manageable chunks.	
	4	The system generates embeddings for each chunk using the embedding model.	
	5	The embeddings and their corresponding text segments are stored in the vector database.	
	6	The system confirms that the learning material has been successfully processed and stored.	
Alternate Flow A1 - *Empty Content*	1.1	The user uploads empty content.	
	1.2	The system notifies the user and cancels processing.	
Alternate Flow A2 – *Invalid Format*	2.1	The user upload material in an unsupported format.	
	2.2	The system prompts the user to re-upload in a valid format.	
Rules	• Only text-based documents are supported • Duplicate uploads overwrite existing stored content unless explicitly renamed.		
Author	Elaine Chung Hui Lin		

Table 3.1.1 Use Case Description for Submit Learning Material

3.1.3.2 Use Case Description for Submit Prompt

Use Case ID	UC002	Version	1.0
Use Case	Submit Prompt		
Purpose	To allow the user to request an educational animation.		
Actor	User		
Trigger	The user enters a prompt into the system interface and clicks Submit.		
Precondition	• The user has already submitted the learning material (corpus). • The system has generated and stored embeddings for the learning material.		
Scenario Name	**Step**	**Action**	
Main Flow	1	The user types a prompt.	
	2	The user clicks “Submit”.	
	3	The system accepts the prompt text.	
	4	The system generates a query embedding from the submitted prompt.	
	5	The query embedding is compared with the vector database.	
	6	The system retrieves semantically relevant content from the corpus.	
	7	The system passes the prompt and retrieved content to the generation model.	
	8	The system produces a DSL script containing animation instructions and narration text.	
Alternate Flow A1 – ***Invalid Prompt***	2.1	User submits an empty or invalid prompt.	
	2.2	The system requests the user to re-enter the prompt.	
Alternate Flow A2 – ***System Error***	5.1	The system fails to connect to the vector database.	
	5.2	The system displays error messages and logs the error.	
Alternate Flow A3 – ***LLM Error***	8.1	The LLM fails to generate a DSL script.	
	8.2	The system displays error messages and logs the error.	
Rules	• A prompt must contain at least one non-empty, valid textual input.		
Author	Elaine Chung Hui Lin		

Table 3.1.2 Use Case Description for Submit Prompt

3.1.3.3 Use Case Description for View Generated Animations

Use Case ID	UC003	Version	1.0
Use Case	View Generated Animations		
Purpose	To allow the user to view the final synchronized animation and narration generated by the system.		
Actor	User		
Trigger	The system has successfully generated both the animation and narration and makes them available for playback.		
Precondition	• The animation script does not contain parse error. • The video rendering module is available and functional. • The system has generated and synchronized the animation and narration.		
Scenario Name	**Step**	**Action**	
Main Flow	1	The system notifies the user that the generated animation is ready.	
	2	The user clicks the button “Play”.	
	3	The system loads the generated video and narration track into the playback interface.	
	4	The system synchronizes narration with the visual animation timeline.	
	5	The system renders and displays an animation playback interface.	
	6	The user views the generated animation with narration.	
Alternate Flow A1 – *Playback Error*	5.1	Playback fails due to rendering or network issue.	
	5.2	The system displays an error message.	
	5.3	Redirect to main flow step 2.	
Rules	• The output must only include content generated from the user-supplied prompt and corpus.		
Author	Elaine Chung Hui Lin		

Table 3.1.3 Use Case Description for View Generated Animations

3.1.3.4 Use Case Description for Edit DSL Script

Use Case ID	UC004	Version	1.0
Use Case	Edit DSL Script		
Purpose	To allow the user to review and correct the generated DSL script.		
Actor	User		
Trigger	The user chooses to manually refine the script.		
Precondition	• The Monaco editor is available and functional. • The system has generated a DSL script.		
Scenario Name	**Step**	**Action**	
Main Flow	1	The user navigates to the “Editor Page”.	
	2	The system automatically loads the generated DSL script into the Monaco editor.	
	3	The user reviews and edits the DSL script as needed.	
	4	The system validates the DSL syntax.	
	5	The system translates the DSL script into TypeScript code.	
	6	The system displays the corresponding TypeScript code.	
	7	The user proceeds to animation rendering (UC003).	
Sub Flow A1 – *Validate DSL*	4.1	If valid, the system translates the DSL script into it’s corresponding TypeScript code.	
	4.2	If invalid, the system displays an error message “Parse failed” with error line number.	
	4.3	The user modifies the DSL manually to fix the error.	
	4.4	Redirect to main flow step 4.	
Rules	• The translated TypeScript code is read-only and intended for reference only. • Only valid DSL syntax may be submitted for translation.		
Author	Elaine Chung Hui Lin		

Table 3.1.4 Use Case Description for Edit DSL Script

3.1.5 Activity Diagram

3.1.4.1 Activity Diagram for UC001

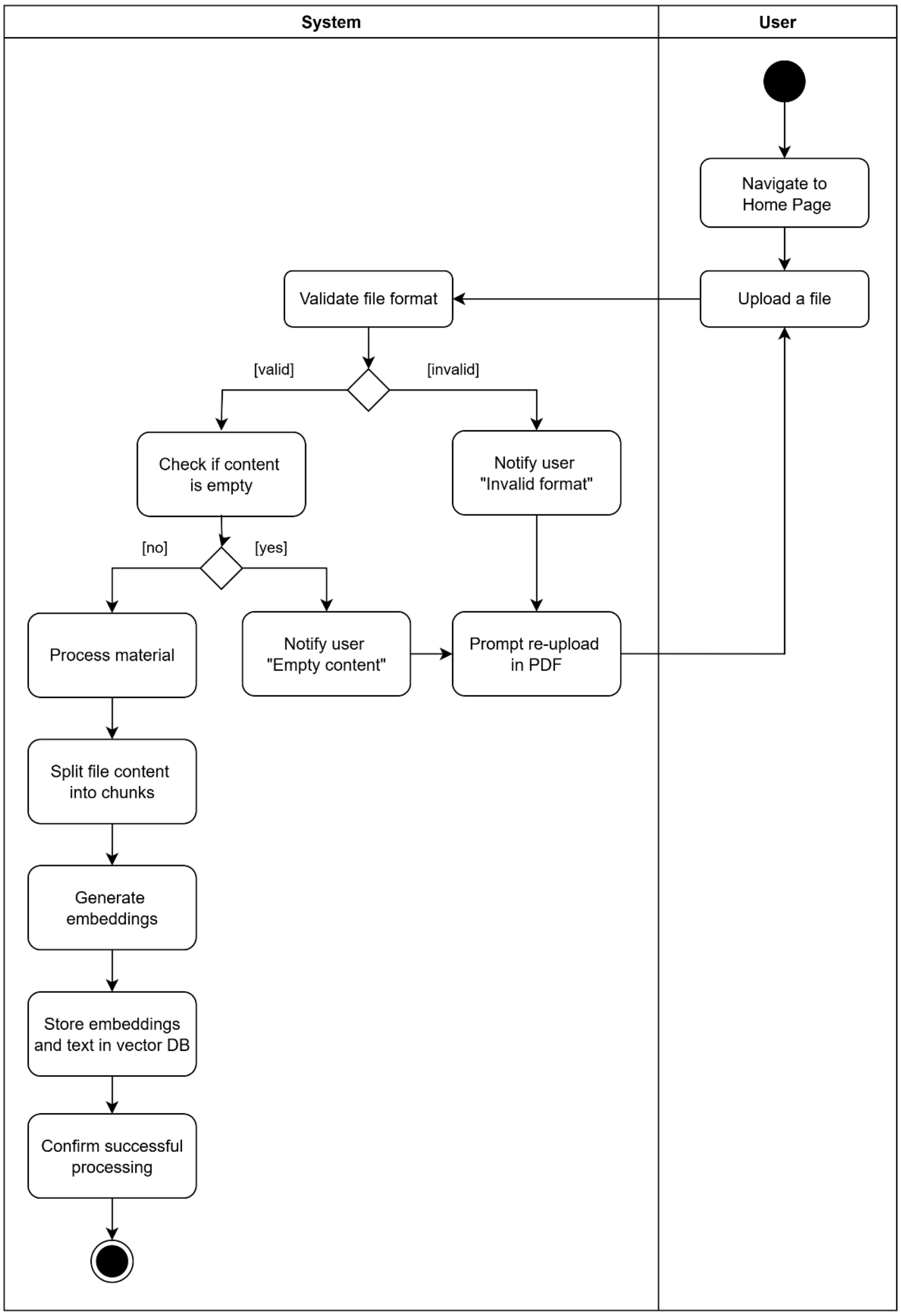

System
User
Navigate to Home Page
Upload a file
Validate file format
[valid]
[invalid]
Check if content is empty
Notify user "Invalid format"
[no]
[yes]
Process material
Notify user "Empty content"
Prompt re-upload in PDF
Split file content into chunks
Generate embeddings
Store embeddings and text in vector DB
Confirm successful processing

Figure 3.1.5 Activity Diagram for UC001: Submit Learning Material

3.1.4.2 Activity Diagram for UC002

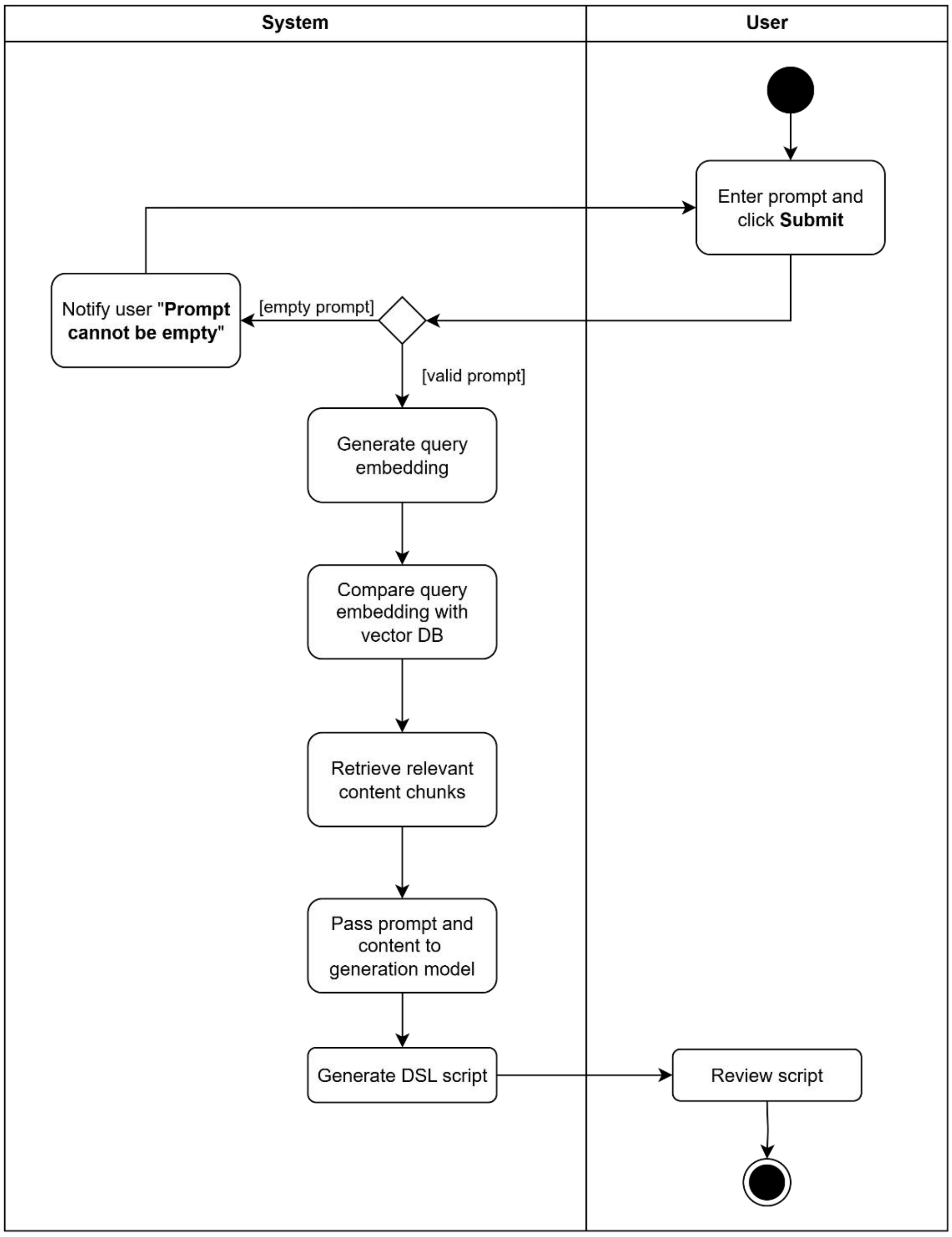

System
User
Enter prompt and click **Submit**
Notify user "**Prompt cannot be empty**"
[empty prompt]
[valid prompt]
Generate query embedding
Compare query embedding with vector DB
Retrieve relevant content chunks
Pass prompt and content to generation model
Generate DSL script
Review script

Figure 3.1.6 Activity Diagram for UC002: Submit Prompt

3.1.4.3 Activity Diagram for UC003

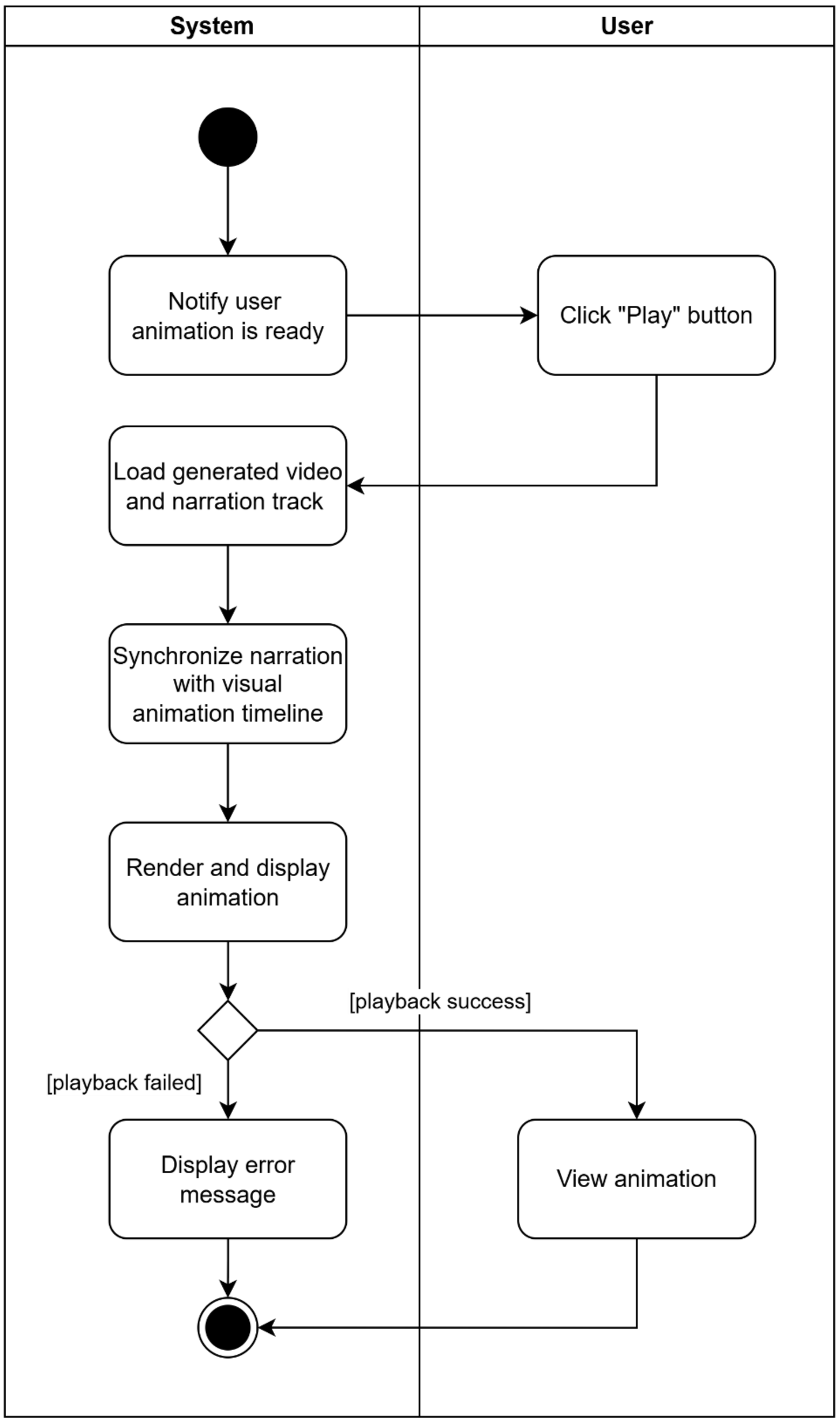

System
User
Notify user animation is ready
Click "Play" button
Load generated video and narration track
Synchronize narration with visual animation timeline
Render and display animation
[playback success]
[playback failed]
Display error message
View animation

Figure 3.1.7 Activity Diagram for UC003: View Generated Animations

-3.1.4.4 Activity Diagram for UC004

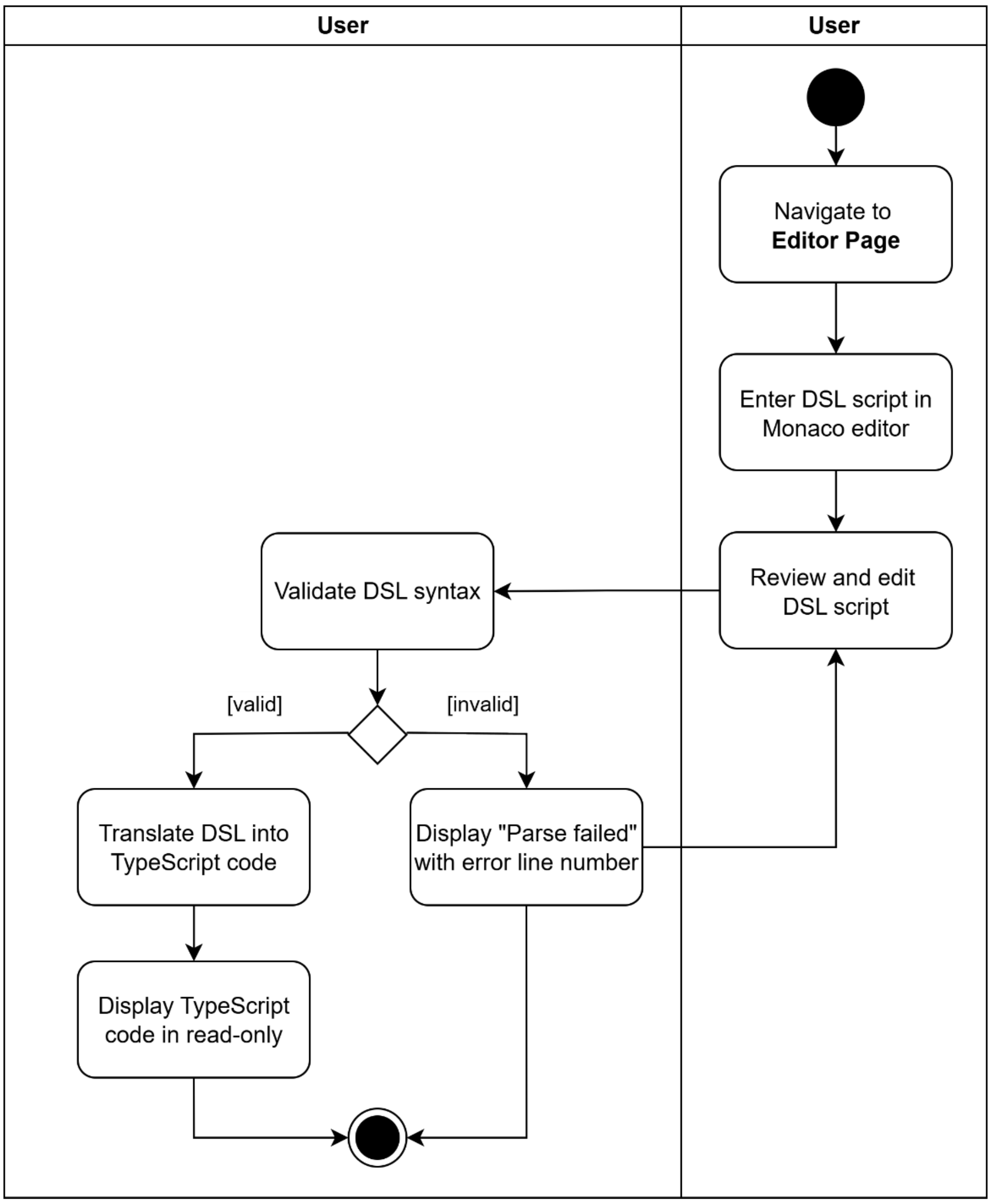

User
User
Navigate to Editor Page
Enter DSL script in Monaco editor
Review and edit DSL script
Validate DSL syntax
[valid]
[invalid]
Translate DSL into TypeScript code
Display "Parse failed" with error line number
Display TypeScript code in read-only

Figure 3.1.8 Activity Diagram for UC004: Edit DSL Script

3.1.6 Verification Plan

The accuracy of generated animations can be accessed by evaluating the degree to which generated animation scripts adhere to the syntax, semantics, and animation logic defined. Ideally, the system should achieve the shortest possible processing and response time while maintaining an elevated level of accuracy.

Another key performance indicator is system responsiveness, measured by the time taken from prompt submission to video playback on web browsers. Faster processing enhances the user experience, particularly for educational settings where quick feedback is valuable. This includes evaluating both the LLM’s response latency and the time taken for the script to be rendered into an animation on the canvas. Optimizing this metric ensures that the system remains viable for interactive use.

The system will be tested using a variety of prompt types, ranging from simple to complex. Simple prompts involve basic shape manipulations (e.g., “Draw a triangle”), while moderate prompts involve structured concepts (e.g., “Illustrate the area of a circle”), and complex prompts encapsulate abstract or multi-step topics (e.g., “Explain the Fibonacci sequence using recursion”). This variation allows for comprehensive assessment of the model's capabilities across different levels of instructional complexity.

Verification will focus on multiple aspects. First, syntax validation ensures that the generated code conforms to the expected structure and language rules of the custom animation library. Second, semantic validation checks whether the generated code logically represents the user’s intent and uses the correct properties, functions, and animation patterns. For instance, prompts requiring concurrent motion will be evaluated to confirm whether parallel animation calls are correctly implemented.

3.2 System Design

This system transforms traditional educational content such as textbooks into animated educational presentations using a pipeline that combines preprocessing, vector database, an LLM, and a custom animation rendering engine. A user provides a natural language prompt, and the system outputs an educational animation rendered on a web-based canvas.

Learning Material Processor

The raw learning material, in PDF format, undergoes preprocessing to clean and normalize the text. This includes removing noise such as headers, footers, inconsistent mathematical markup, and non-content sections like the prefaces. The cleaned content is then divided into topic-based segments or "chunks" to fit within the context window of LLMs. Each chunk is embedded into a dense vector representation using a pre-trained embedding model.

Vector Database

The system uses Pinecone, an open-source vector database, to store and manage these embeddings along with their associated metadata, including topic and section. At runtime, this database enables semantic retrieval of the most relevant content chunks in response to user queries.

Query and Retrieval

When a user submits a prompt, it is embedded into a vector using the same embedding model. This query vector is used to retrieve the top N semantically similar chunks from the vector database. These retrieved chunks provide the contextual grounding for the LLM's response.

LLM Reasoning Core

The LLM synthesizes an animation script using the retrieved contextual information. It generates structured animation instructions via a custom animation library API designed to be both semantically meaningful and programmatically valid.

Animation Library API

The animation library implemented in TypeScript, provides a set of programmable functions for visual elements (e.g., create, move, fadeIn, scale) for visual elements. A domain-specific language (DSL) acts as an abstraction layer between the LLM and the animation engine, allowing the LLM to produce precise and valid animation instructions. This DSL enforces constraints, such as requiring each visual object to have a unique ID, ensuring structured and predictable output behaviour and rendering consistency.

Animation DSL Syntax

The DSL follows a lexical note defined as follows:
Colors: red, green, blue, yellow, brown, black, white, orange, purple
IDs: must start with a letter/underscore, can contain letters, numbers, underscores
Numbers: integers or floats (e.g., 10, -3.5)

The DSL scripts consist of statements executed sequentially unless explicitly grouped for parallel execution. Statements are the fundamental units of the DSL, and each statement belongs to one of the following categories:

1. Shape Definition

Shapes are assigned an unique identifier (ID) upon creation just like how the naming of a variable is unique in a program. These identifiers are used later in group definitions and animation statements. The syntax is show as below:

Shape	**Syntax**
circle	<id> = circle at (<x>, <y>) radius <number> [color <color>]
dot	<id> = rectangle at (<x>, <y>) width <number> height <number> [color <color>]
square	<id> = square at (<x>, <y>) size <number> [color <color>]
triangle	<id> = triangle at (<x>, <y>) radius <number> [color <color>]
line	<id> = line (<x1>, <y1>) to (<x2>, <y2>) [color <color>]
text	<id> = text "<string>" at (<x>, <y>) [size <number>] [color <color>]

2. Group Definition

Groups allow multiple shapes to be animated and manipulated together. The syntax to define a group is as follows:

<id> = group { <id1>, <id2>, ... }

3. Animation Command

Animations define transformations or effects applied to shapes or groups.
Durations are optional. If omitted, default timing is used.

Command	**Syntax**
move	move <id> to (<x>, <y>) [over <seconds>s]
fade in	fadeIn <id> [over <seconds>s]
fade out	fadeOut <id> [over <seconds>s]
scale	scale <id> to <factor> [over <seconds>s]
rotate	rotate <id> by <degrees> [around (<x>, <y>)] [over <seconds>s]

4. Block Command (parallel execution)

Parallel execution of multiple statements:

```
parallel {
    animation command
}
```

5. Sleep Command

Pause execution for a specified number of seconds:

```
sleep <seconds>s
```

6. Narration Command (TTS)

Narration is expressed as inline comments beginning with "TTS:". They are parsed separately and converted into audio narration by the narration engine.

```
TTS: <text>
```

To enable reliable interpretation and execution of the DSL, the system incorporates a formal language processing pipeline composed of the following components. The parser gets tokens from the lexer and then returns the generated AST for the interpreter to traverse and interpret the input.

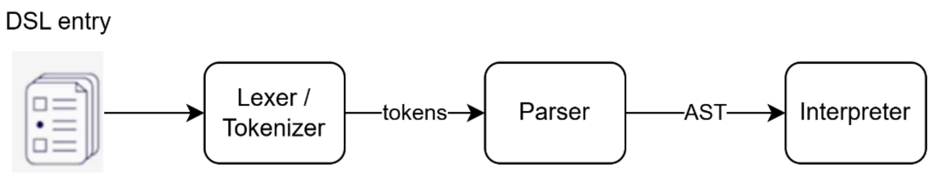


Figure 3.2.1 DSL Processing Pipeline

First, the DSL entry is broken into tokens like keywords, identifiers, coordinates, timestamps. Then, a lexer categorizes character streams into token classes, for example, identifier, number, string, symbol and timecode. The output is then passed to a parser which will then analyse the token stream based on a defined grammar, it verifies whether the input conforms to the DSL's rules and builds a parse tree (concrete syntax tree), which reflects the nested structure of each animation instruction. The parse tree is then transformed into an Abstract Syntax Tree (AST) which is a simplified, program-friendly representation of the instruction. This AST serves as the central structure for further semantic validation and execution. The AST is converted into an intermediate representation (IR) tailored to the animation engine. This IR abstracts higher-level commands into engine-specific API calls, decoupling the DSL structure from the runtime logic.

<u>Web Canvas Renderer</u>

This is a browser-based rendering engine that interprets the animation script and displays the resulting animation in real-time on an HTML canvas.

3.2.1 Sequence Diagram

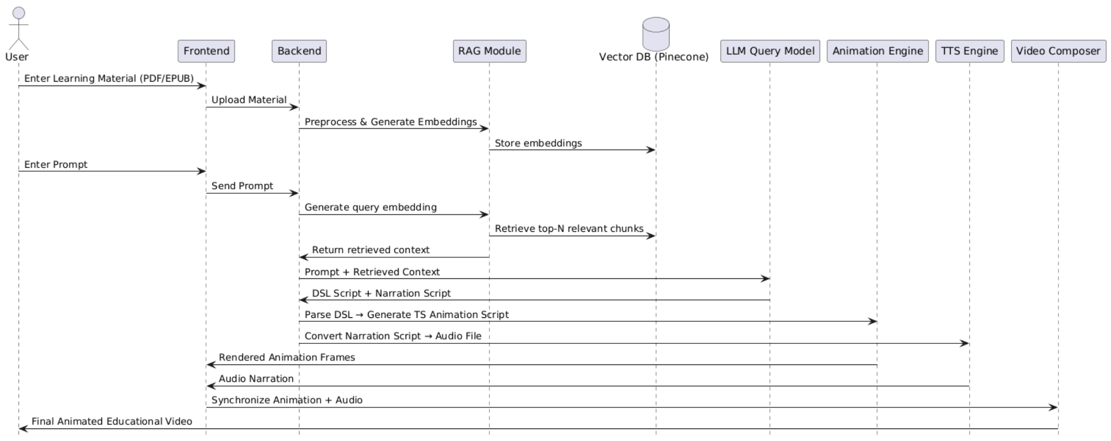

User
Frontend
Backend
RAG Module
Vector DB (Pinecone)
LLM Query Model
Animation Engine
TTS Engine
Video Composer
Enter Learning Material (PDF/EPUB)
Upload Material
Preprocess & Generate Embeddings
Store embeddings
Enter Prompt
Send Prompt
Generate query embedding
Retrieve top-N relevant chunks
Return retrieved context
Prompt + Retrieved Context
DSL Script + Narration Script
Parse DSL → Generate TS Animation Script
Convert Narration Script → Audio File
Rendered Animation Frames
Audio Narration
Synchronize Animation + Audio
Final Animated Educational Video

Figure 3.2.2 Sequence Diagram of End-to-End Educational Animation Pipeline

3.2.2 Class Diagram

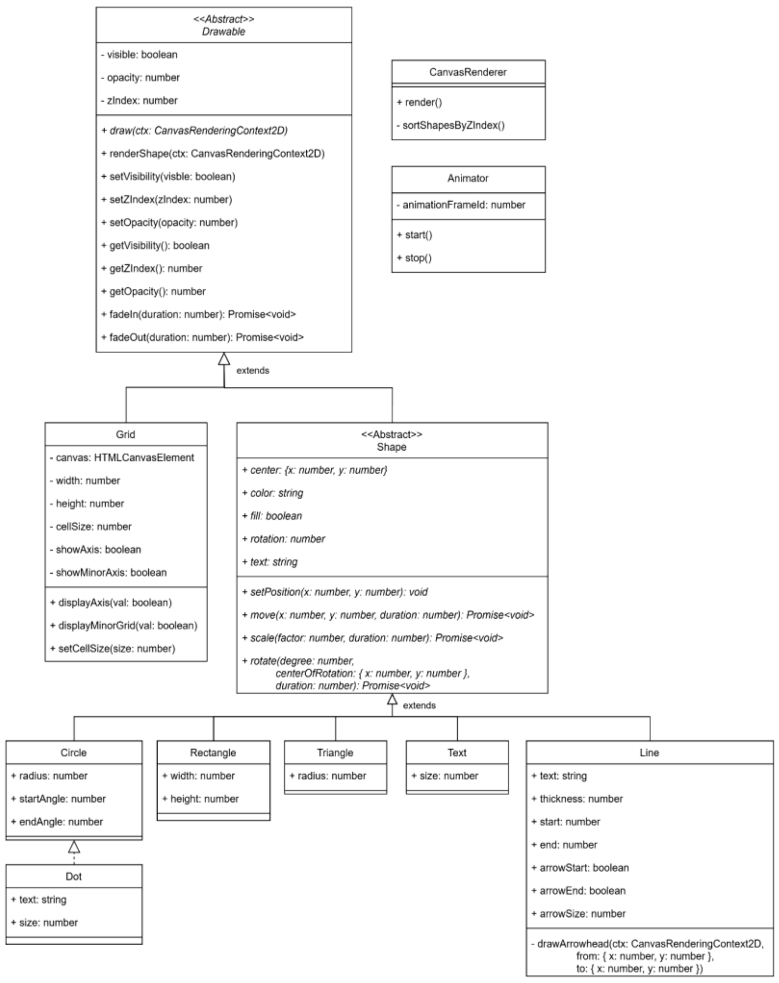

<<Abstract>> Drawable
- visible: boolean
- opacity: number
- zIndex: number
+ draw(ctx: CanvasRenderingContext2D)
+ renderShape(ctx: CanvasRenderingContext2D)
+ setVisibility(visble: boolean)
+ setZIndex(zIndex: number)
+ setOpacity(opacity: number)
+ getVisibility(): boolean
+ getZIndex(): number
+ getOpacity(): number
+ fadeIn(duration: number): Promise<void>
+ fadeOut(duration: number): Promise<void>
CanvasRenderer
+ render()
- sortShapesByZIndex()
Animator
- animationFrameId: number
+ start()
+ stop()
extends
Grid
- canvas: HTMLCanvasElement
- width: number
- height: number
- cellSize: number
- showAxis: boolean
- showMinorAxis: boolean
+ displayAxis(val: boolean)
+ displayMinorGrid(val: boolean)
+ setCellSize(size: number)
<<Abstract>> Shape
+ center: {x: number, y: number}
+ color: string
+ fill: boolean
+ rotation: number
+ text: string
+ setPosition(x: number, y: number): void
+ move(x: number, y: number, duration: number): Promise<void>
+ scale(factor: number, duration: number): Promise<void>
+ rotate(degree: number, centerOfRotation: { x: number, y: number }, duration: number): Promise<void>
extends
Circle
+ radius: number
+ startAngle: number
+ endAngle: number
Rectangle
+ width: number
+ height: number
Triangle
+ radius: number
Text
+ size: number
Line
+ text: string
+ thickness: number
+ start: number
+ end: number
+ arrowStart: boolean
+ arrowEnd: boolean
+ arrowSize: number
- drawArrowhead(ctx: CanvasRenderingContext2D, from: { x: number, y: number }, to: { x: number, y: number })
Dot
+ text: string
+ size: number

Figure 3.2.3 Class diagram of custom TypeScript animation library

From the above class diagram, the core of the design is the Drawable abstract class, which defines common properties such as visibility, opacity, and z-index, along with methods for rendering and applying fade in and out effects. The Shape abstract class extends Drawable and introduces geometric attributes such as position, rotation, color, and fill state, along with transformation methods (move, scale, rotate, and setPosition). This layered abstraction enforces consistency while allowing specialization for different graphical primitives.

Concrete shape classes include Circle, Rectangle, Triangle, Line, and Text. Each class introduces parameters specific to its geometry, such as radius for circles, width and height for rectangles, or endpoints and thickness for lines. Specialized variations like Dot extend Circle with text labeling, demonstrating the extensibility of the framework. These shapes can be independently animated or grouped together to form composite objects, enabling complex and coordinated animations.

Supporting infrastructure is provided by the CanvasRenderer, which is responsible for sorting shapes by z-index and rendering them on the HTML canvas. The Animator class manages the animation frame loop, starting and stopping playback to ensure smooth execution of time-based transformations. Additionally, the Grid class provides a structured background, configurable by width, height, cell size, and axis display options, supporting visual clarity for mathematical and geometric animations.

3.3 Challenges in LLM-Based Animation

3.3.1 Ensuring Accuracy

One of the key implementation challenges in this project lies in the reliance on LLMs to interpret and generate educational content. While LLMs are highly effective at producing coherent and relevant explanations, they may overlook nuanced mathematical details or struggle with advanced topics that require precision and formal rigor. This limitation introduces a risk of generating content that is either oversimplified or inaccurate, especially when dealing with spatial reasoning, complex formulas, mathematical proofs, or algorithmic processes. Therefore, ensuring math correctness and logic requires careful prompt engineering which adds additional layers of complexity to the system. Expert oversight is necessary, with educators and subject-matter experts validating the content for both pedagogical and factual accuracy.

3.3.2 High Computational Demand

Another significant challenge is the cost and performance trade-off when using paid APIs, particularly for processing large volumes of text such as an entire mathematics textbook. Parsing and summarizing hundreds of pages using GPT-4o can incur high API costs, contradicting the project's aim of delivering a cost-effective solution for educational content creation. Rendering complex animations requires significant processing power, further increasing computational load. Moreover, API rate limits and long processing times make it impractical to rely solely on external services for bulk content processing. These constraints necessitate optimization strategies, such as extracting only relevant sections on demand or exploring lightweight local alternatives. In short, balancing functionality, speed, and affordability remains a core challenge in implementing the system efficiently.

Chapter 4

System Design

This section outlines the preliminary work done and results according to the proposed method in Chapter 3. The feasibility of the project is highlighted with implementation details. At this stage of the project, the system implements the workflow starting from a user prompt to the generation of web-based canvas animation, omitting the preprocessing and vector retrieval components.

4.1 System Block Diagram

The system block diagram illustrates the overall structure and interaction between the key components within the application. It outlines the flow of data and the connection between the user interface and logic processing layer.

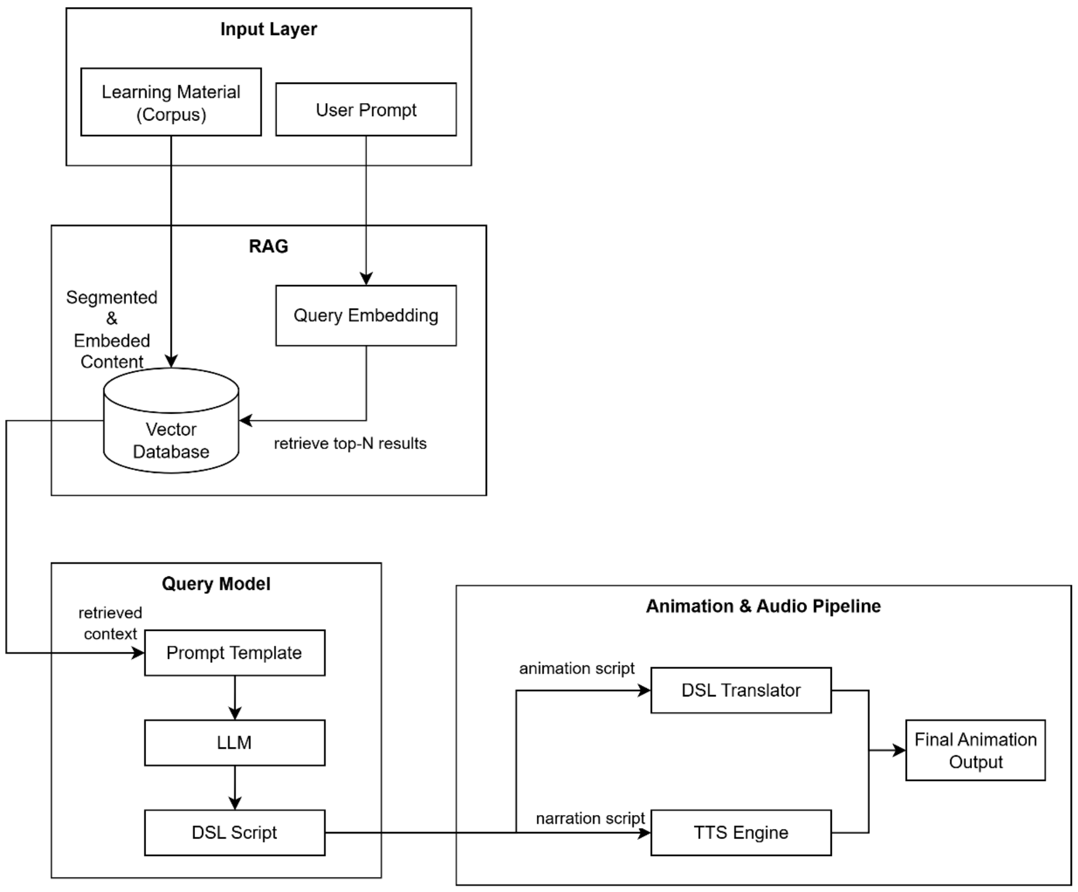


Figure 4.1.1 System Block Diagram

4.2 System Components Specifications

4.2.1 Input Handling

The Input Handling component manages all user-provided data, including the learning material and follow-up prompt. Learning material serves as the knowledge corpus and is pre-processed into vector embeddings for semantic retrieval, while the user prompt specifies the topic or concept to be explained. This component ensures that both inputs are properly validated and formatted before being passed to RAG module. By efficiently managing and organizing these inputs, the system guarantees that subsequent processes, including context retrieval and LLM-based script generation, receive accurate and structured data for producing educational videos.

4.2.2 RAG Module

The RAG module integrates external knowledge from the learning material into the LLM to ensure that generated content is accurate, syllabus-aligned, and contextually relevant. LLMs alone have limitations, including knowledge cut-off dates, lack of access to domain-specific textbooks, and a finite context window that prevents processing large learning materials in a single prompt. Without grounding, LLMs may hallucinate or produce explanations that deviate from the intended curriculum. The RAG approach overcomes these challenges by indexing the learning material as vector embeddings, retrieving the top-N semantically relevant segments in response to a user query, and conditioning the LLM's output on these segments. This ensures that the generated animations and narration are factually correct, aligned with the syllabus, and grounded in authoritative sources.

4.2.3 LLM Query Model

The LLM Query Model is responsible for generating both the visual and narration components of the educational video. It receives a structured prompt containing the user query and context retrieved by the RAG module. In FYP1, multiple LLMs were experimented with, including OpenAI GPT-4o, GPT-4.1, Meta-Llama-4-17B-16E-Instruct, and DeepSeek models, to study output quality and behavior. In the current phase (FYP2), the system primarily relies on **OpenAI GPT-4o** due to its superior reasoning capabilities, large context window, reliable adherence to structured prompts, and high-quality code generation, which are essential for producing accurate and executable TypeScript DSL scripts.

A **system prompt** is used to define the model's behaviour, responsibilities, and output format. It specifies the animation library's available shapes, creation syntax, supported animation methods, and sequencing rules, ensuring that the LLM produces coherent, executable scripts compatible with the Animation Engine. The LLM outputs two components: a **DSL script** describing the animation sequence and a **narration script** for spoken explanations. By combining GPT-4o's capabilities with a structured system prompt, the module ensures consistent, accurate, and syllabus-aligned educational content suitable for automated video generation.

4.2.4 Animation Engine

The Animation Engine is implemented as a TypeScript library and serves as the core module responsible for rendering the visual content of the educational video. It adopts an object-oriented programming (OOP) design, encapsulating graphical and animation logic into a hierarchy of reusable classes. The class diagram of the library can be found in Section 3.2.3. The DSL script generated by the LLM is parsed and interpreted into concrete operations that act on these objects, translating them from high-level animation instructions to low-level rendering commands.

A parser interprets the DSL script generated by the LLM, converting high-level animation commands into executable operations on shapes, text, and groups. The library supports fundamental shapes such as circles, rectangles, triangles, and text objects, as well as animation methods including drawing, moving, scaling, fading, and grouping. Grouping mechanisms allow multiple shapes to be manipulated together, enabling complex coordinated animations. Figure 3.2.3 shows the class structure of this library, illustrating the main shapes, animator classes, and grouping mechanisms. By providing a structured and modular API, the animation engine ensures that visual content is rendered accurately and efficiently according to the specifications defined in the DSL script.

4.2.5 TTS Engine

The system uses the Google Text-to-Speech (gTTS) SDK for Node.js. The TTS Engine converts the narration script generated by the LLM into audio files, providing a spoken explanation that complements the animated visuals. This allows the final video output to include synchronized narration. The TTS Engine is invoked in the dslController immediately after DSL processing. The returned audio file list is passed to the Video Composition module, where it is synchronized with the rendered animation to produce the final educational video.

- **Input**: An array of narration strings extracted from the DSL output
- **Output**: Corresponding .mp3 audio files, each representing a segment of narration

4.2.6 Video Composition and Playback

The Video Composition and Playback component integrates the outputs of the Animation Engine and the TTS Engine to produce the final educational animation on HTML5 Canvas. It synchronizes animation frames with audio narration, ensuring smooth playback and coherent timing. To control sequential execution of scenes, generated animation logic is wrapped in async Immediately Invoked Function Expressions (IIFE), allowing the use of await and short pauses to create smooth scenic transitions. On the other hand, canvas rendering performance is optimized using requestAnimationFrame, maintaining approximately 60 frames per second. Supporting utility functions, such as timing and interpolation helpers, ensure consistent animation behavior, while the coordinate system uses a top-left origin (0,0) with pixel-based measurements, optionally scalable for different screen sizes. A debug mode is also provided to visualize shape boundaries and coordinate grids during development, facilitating efficient testing and verification of animation sequences. By combining performance optimization, timing utilities, and careful synchronization, the Video Composition component delivers smooth, high-quality educational animations with synchronized narration.

4.3 System Data Flow

This section presents the flow of data and interactions between system components in the automated educational video generation pipeline. The system processes inputs, retrieves relevant content, generates animations and narration, and composes the final video. The main data flow can be described as follows:

Input Stage:

- Users provide learning material (PDFs, text) and a follow-up query.
- Learning material is pre-processed into vector embeddings for semantic search.

RAG Module:

- The user query is converted into an embedding and compared against the vector database.
- Top-N relevant content segments are retrieved, providing context for the LLM.

LLM Query Model:

- Receives the user query and retrieved context.
- Generates two outputs: a **DSL script** for the animation engine and a **narration script** for the TTS engine.

Animation and TTS Engines:

- The DSL script is interpreted by the Animation Engine to render shapes, movements, and transitions.
- The narration script is processed by the TTS engine to generate audio files.

Video Composition:

- Animation frames and audio files are synchronized.
- The final output is a cohesive educational video that integrates visuals and narration.

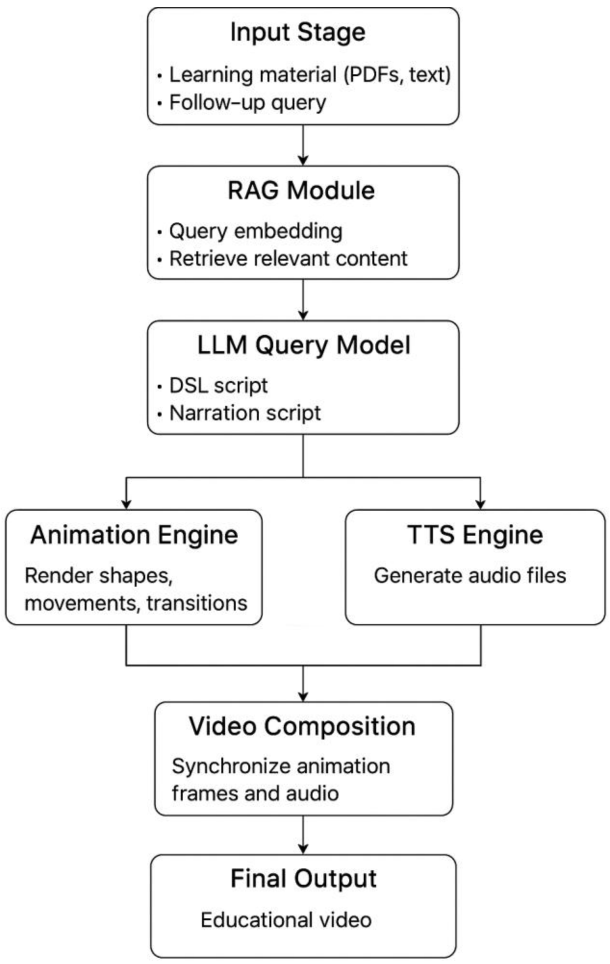


Figure 4.3.1 Data Flow for Automated Educational Video Generation Pipeline

Chapter 5

System Implementation

This chapter provides an overview of how the automated educational video generation system was configured and developed. It outlines the setup of the project environment, the integration of core components which are RAG, the DSL, the TypeScript animation engine, and TTS, and their combination into a working pipeline. Challenges encountered during implementation and the solutions applied are also discussed, along with reflections on the system's performance and areas for potential improvement.

5.1 Hardware Setup

The hardware required for this project includes a laptop for development and testing.

Table 5.1.1 Hardware specifications

Aspect	Specification
Model	BOD-WXX9
Processor	11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz (8CPUs)
OS	Windows 11 Home Single Language 64-bit (10.0, build 22631)
RAM	16GB
GPU	Intel(R) Iris(R) Xe Graphics

5.2 Software Setup

The tooling and library installation are listed below: --

Table 5.2.1 Development Tools

Software	Version	Purpose
Visual Studio Code	2025 1.103.2	IDE
git	2.45.2.windows.1	Version Control
Postman	11.59.6	API testing

Table 5.2.2 Core Software Stack

Software / Framework	Version	Purpose
TypeScript	5.8.3	Main programming language
React.js	19.1.0	Frontend library
TailwindCSS	3.4.17	Frontend styling
Node.js	20.14.0	Backend runtime
ANTLR4	0.5.0-alpha.4	Parser generator
gTTS	-	Text to speech
Pinecone	-	Vector store
OpenAI GPT-4o	-	Natural language processing
OpenAI text-embedding-3-small	-	Text to embeddings

5.3 Setting and Configuration

This section outlines the setup and configuration of the development environment, and project settings for the development and testing of the system.

5.3.1 Development Environment Configuration

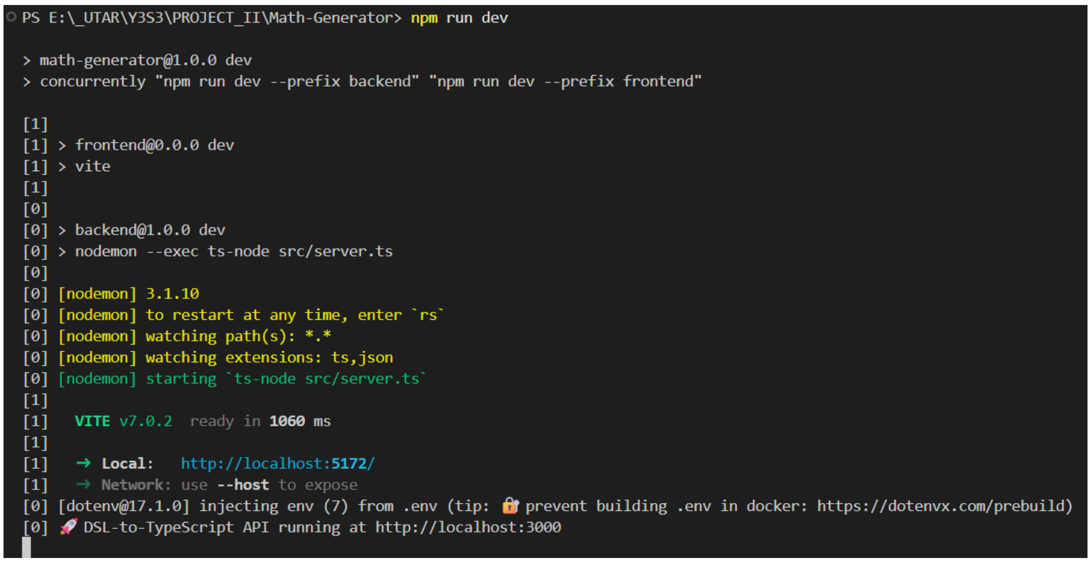
The system was developed using the hardware and software environments described in Sections 5.1 and 5.2.

Node.js served as the runtime environment for backend operations and managing the development of the ecosystem. To connect backend to frontend, Axios is used as the client for sending HTTP requests. To prepare the development environment, a React application was created using the following commands, which generate the boilerplate files for the application:

```
npm create-react-app math-generator
cd math-generator
npm start
```

For subsequent start-ups, the following commands are run in the VS Code Terminal:

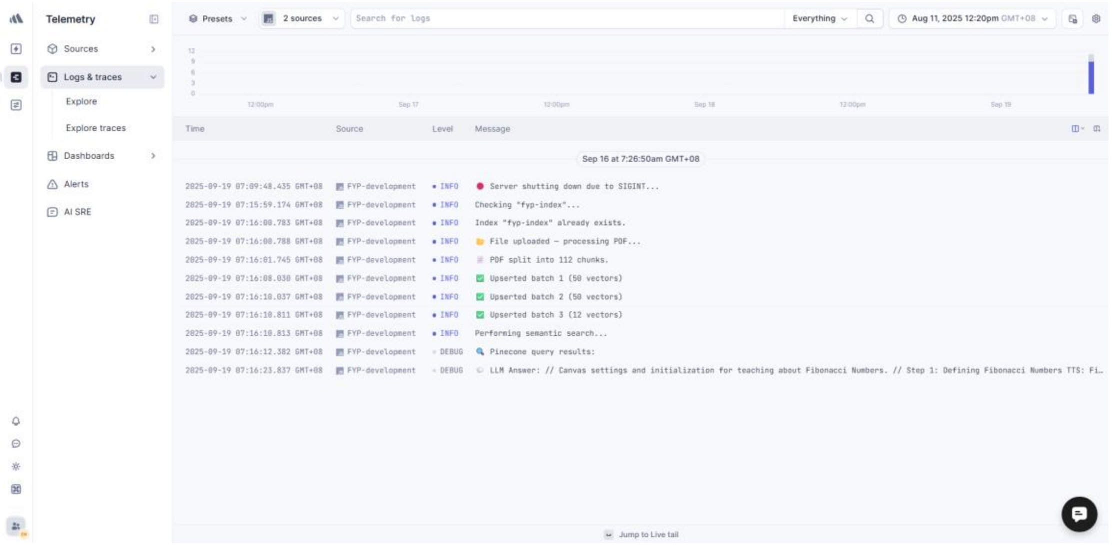
```
npm run dev
```



```
PS E:\_UTAR\Y3S3\PROJECT_II\Math-Generator> npm run dev

> math-generator@1.0.0 dev
> concurrently "npm run dev --prefix backend" "npm run dev --prefix frontend"

[1]
[1] > frontend@0.0.0 dev
[1] > vite
[1]
[0]
[0] > backend@1.0.0 dev
[0] > nodemon --exec ts-node src/server.ts
[0]
[0] [nodemon] 3.1.10
[0] [nodemon] to restart at any time, enter `rs`
[0] [nodemon] watching path(s): *.*
[0] [nodemon] watching extensions: ts,json
[0] [nodemon] starting `ts-node src/server.ts`
[1]
[1]   VITE v7.0.2  ready in 1060 ms
[1]
[1]   ➜  Local:   http://localhost:5172/
[1]   ➜  Network: use --host to expose
[0] [dotenv@17.1.0] injecting env (7) from .env (tip: 🔐 prevent building .env in docker: https://dotenvx.com/prebuild)
[0] 🚀 DSL-to-TypeScript API running at http://localhost:3000
```

Figure 5.3.1 Starting Local Development Server

The system also uses Logtail for centralized logging. The logging configuration is defined in the backend, where API requests, parsing errors, and runtime events are captured and forwarded to Logtail for monitoring.



```
Time                         Source            Level   Message
                                  Sep 16 at 7:26:50am GMT+08
2025-09-19 07:09:48.435 GMT+08  FYP-development  • INFO   ● Server shutting down due to SIGINT...
2025-09-19 07:15:59.174 GMT+08  FYP-development  • INFO   Checking "fyp-index"...
2025-09-19 07:16:00.783 GMT+08  FYP-development  • INFO   Index "fyp-index" already exists.
2025-09-19 07:16:00.788 GMT+08  FYP-development  • INFO   📁 File uploaded – processing PDF...
2025-09-19 07:16:01.745 GMT+08  FYP-development  • INFO   📄 PDF split into 112 chunks.
2025-09-19 07:16:08.030 GMT+08  FYP-development  • INFO   ✅ Upserted batch 1 (50 vectors)
2025-09-19 07:16:10.037 GMT+08  FYP-development  • INFO   ✅ Upserted batch 2 (50 vectors)
2025-09-19 07:16:10.811 GMT+08  FYP-development  • INFO   ✅ Upserted batch 3 (12 vectors)
2025-09-19 07:16:10.813 GMT+08  FYP-development  • INFO   Performing semantic search...
2025-09-19 07:16:12.382 GMT+08  FYP-development  • DEBUG  🔍 Pinecone query results:
2025-09-19 07:16:23.837 GMT+08  FYP-development  • DEBUG  LLM Answer: // Canvas settings and initialization for teaching about Fibonacci Numbers. // Step 1: Defining Fibonacci Numbers TTS: Fi…
```

Figure 5.3.2 Logtail for monitoring logs

5.3.1.1 Application Dependencies

The project was initialized using Node.js and TypeScript. Core dependencies were installed and managed via npm, with all required packages listed inside the package.json file. The pacakages are installed using the command npm install <package_name>.

Table 5.3.1 Application Dependencies

Package Name	Version	Purpose in the Application
@azure-rest/ai-inference	^1.0.0-beta.6	Azure AI inference client for LLM interactions.
@azure/core-auth	^1.10.0	Provides authentication support for Azure services.
@azure/core-sse	^2.3.0	Enables server-sent events (SSE) for streaming responses from Azure/OpenAI models.
@pinecone-database/pinecone	^6.1.2	Client library for interacting with Pinecone vector database for semantic retrieval (RAG).
antlr4ts	^0.5.0-alpha.4	TypeScript target of ANTLR, used to define and parse the custom DSL grammar.
cors	^2.8.5	Middleware to enable cross-origin requests for the backend API.
dotenv	^17.1.0	Loads environment variables from .env into process.env.
express	^5.1.0	Backend web framework for defining routes and APIs.
expresss	^0.0.0	(Likely a typo or unused package — “express” should be used).
gtts	^0.2.1	Google Text-to-Speech integration for narration generation.
langchain	^0.3.30	Framework for managing LLM calls, prompt templates, and retrieval augmentation (RAG).
multer	^2.0.2	Middleware for handling file uploads (e.g., PDF learning material).
openai	^5.11.0	Official OpenAI client library for API calls to GPT models.
pdf-parse	^1.1.1	Extracts text from uploaded PDF learning material for preprocessing.

5.3.1.2 Development Dependencies

Table 5.3.2 Project Development Dependencies

Package Name	**Version**	**Purpose in the Application**
@logtail/node	^0.5.5	Client for sending logs to Logtail (structured logging).
@logtail/winston	^0.5.5	Winston transport for integrating Logtail logging.
@types/cors	^2.8.19	Type definitions for cors.
@types/express	^5.0.3	Type definitions for express
@types/katex	^0.16.7	Type definitions for katex.
@types/multer	^2.0.0	Type definitions for multer.
@types/pdf-parse	^1.1.5	Type definitions for pdf-parse.
css-loader	^7.1.2	Loads and resolves CSS imports in frontend React project.
nodemon	^3.1.10	Automatically restarts the backend server on file changes during development.
style-loader	^4.0.0	Injects CSS into the DOM in the React frontend.
typescript	^5.8.3	Provides static typing and compilation for TypeScript code.
winston	^3.17.0	Logging library for structured and configurable log output.

5.3.2 Project Configuration

5.3.2.1 Project Directory Structure

The project is organized into **backend** and **frontend** repositories, each with a clear directory structure to support modular development and maintainability.

Backend

The backend handles DSL parsing, RAG retrieval, LLM integration, TTS generation, and API endpoints. Its main structure is as follows:

```
backend/
├── dist/                      # Compiled TypeScript output
│   └── lib/
│       ├── group/              # Grouping classes for animations
│       ├── interface/         # TypeScript interfaces
│       ├── shapes/            # Shape classes (Circle, Rectangle, etc.)
│       └── util/              # Utility functions
├── public/
│   └── audio/                 # Generated TTS audio files
└── src/
    ├── assets/                # Static assets
    ├── config/                # Configuration files and constants
    ├── controllers/           # API controllers, including DSL controller
    ├── DSL/                   # DSL grammar, parser, and visitor
    │   ├── .antlr/            # ANTLR grammar files
    │   └── generated/          # ANTLR-generated parser and visitor files
    ├── RAG/                    # Retrieval-Augmented Generation module
    ├── routes/                 # Express route definitions
    └── util/                   # Utility scripts (e.g., reusable helper function)
```

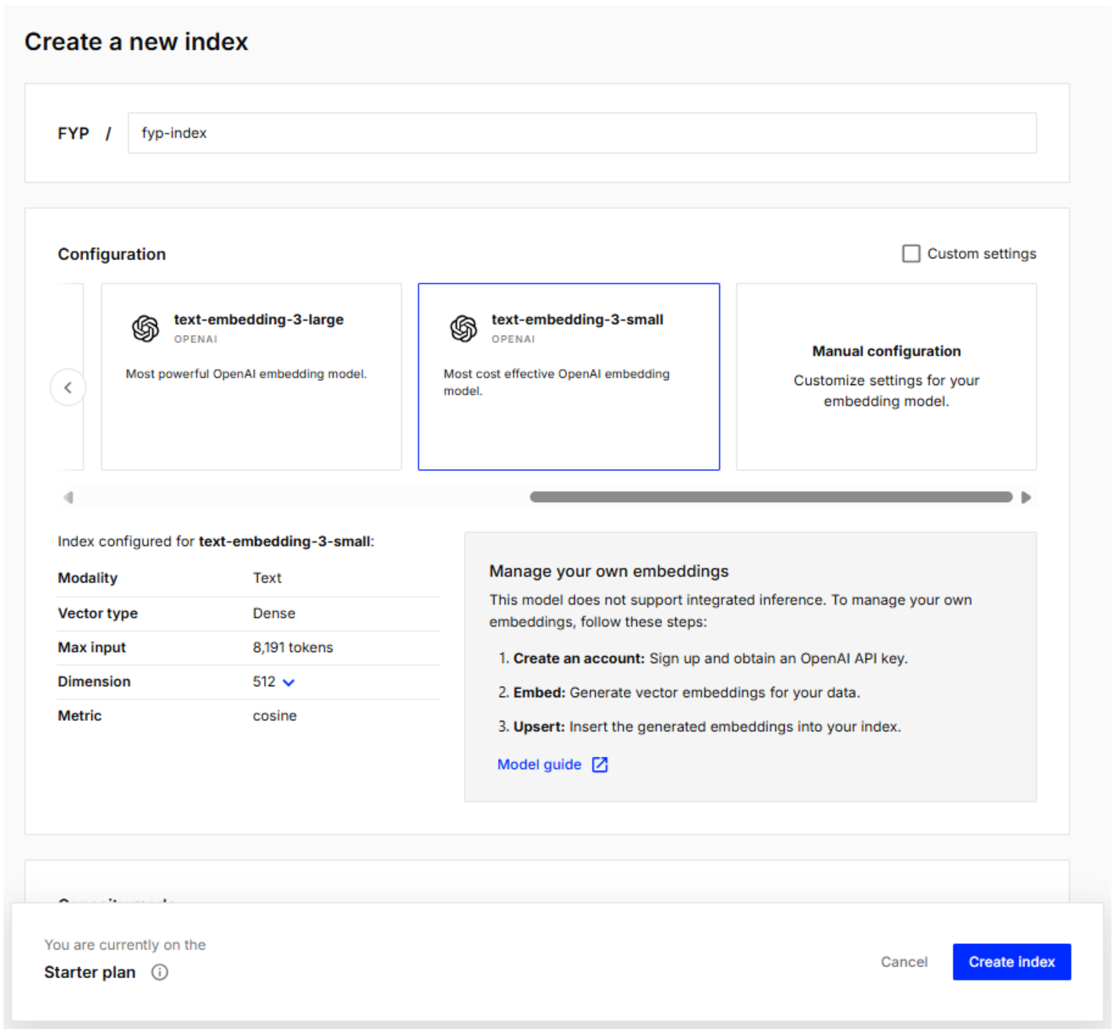
Frontend

The frontend handles the user interface, rendering the animations on canvas, and interacting with the backend via API calls:

```
frontend/
├── .vite/                 # Vite build cache and dependency files
│   └── deps/
├── public/                # Static files served by the frontend
└── src/
    ├── assets/            # Static assets (images, icons, etc.)
    ├── components/        # Reusable React/Vue components
    ├── lib/               # Core animation library
    │   ├── animation/     # Animation engine scripts
    │   ├── group/         # Grouping logic for shapes
    │   ├── interface/     # TypeScript interfaces
    │   └── shapes/        # Shape classes for canvas rendering
    └── pages/             # Application pages and routes
```

5.3.2.2 Pinecone Database Initialization

An index is first created in Pinecone, where **text-embedding-3-small** is selected as the embedding model due to its cost-effectiveness and suitability for semantic search tasks. AWS was chosen as the cloud provider, with us-east-1 as the deployment region.



Create a new index

FYP / fyp-index

Configuration

Custom settings

text-embedding-3-large
OPENAI
Most powerful OpenAI embedding model.

text-embedding-3-small
OPENAI
Most cost effective OpenAI embedding model.

Manual configuration
Customize settings for your embedding model.

Index configured for **text-embedding-3-small**:

Modality	Text
Vector type	Dense
Max input	8,191 tokens
Dimension	512
Metric	cosine

Manage your own embeddings

This model does not support integrated inference. To manage your own embeddings, follow these steps:

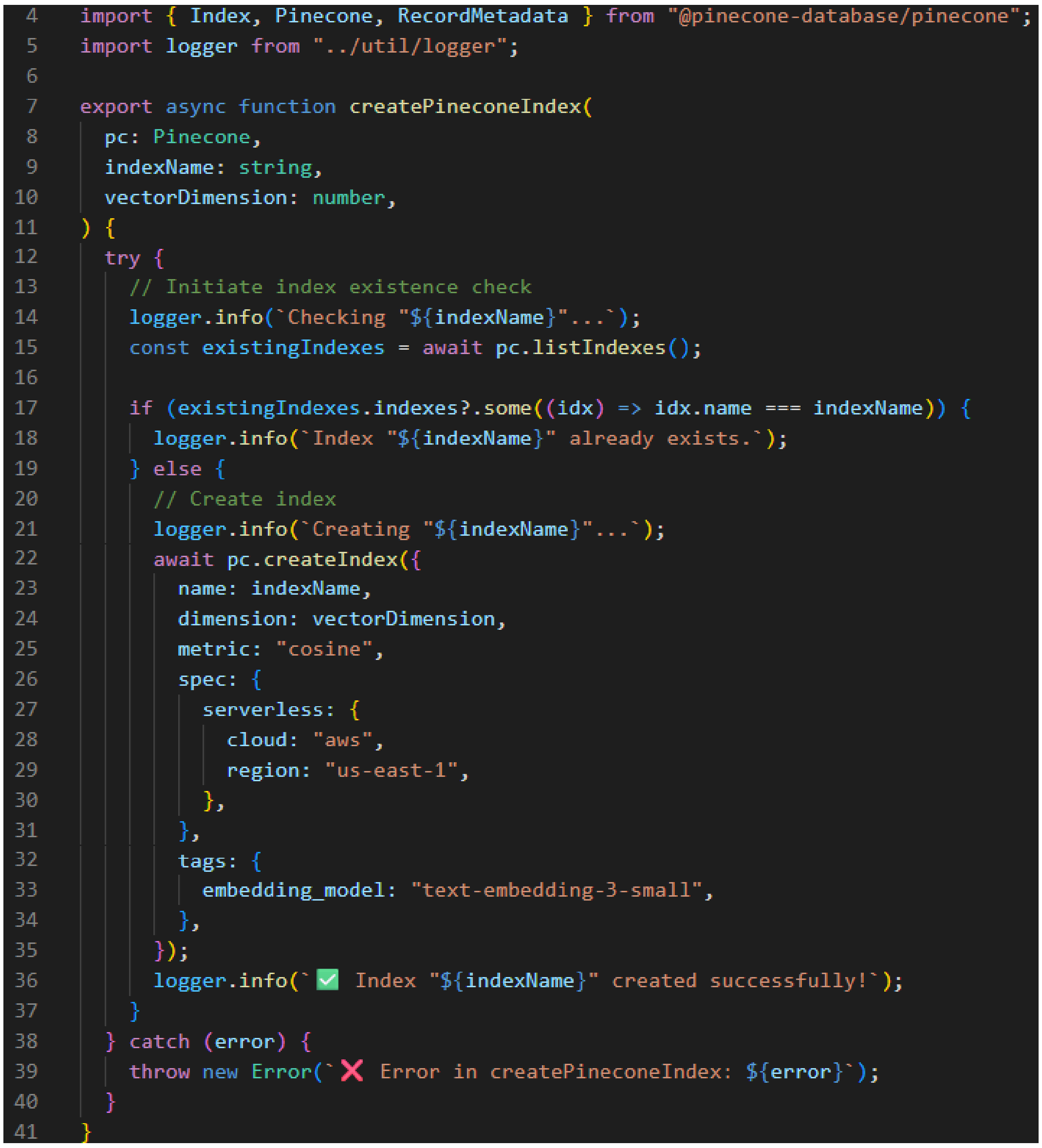
1. **Create an account:** Sign up and obtain an OpenAI API key.
2. **Embed:** Generate vector embeddings for your data.
3. **Upsert:** Insert the generated embeddings into your index.

Model guide

You are currently on the Starter plan

Figure 5.3.1 Pinecone index setup

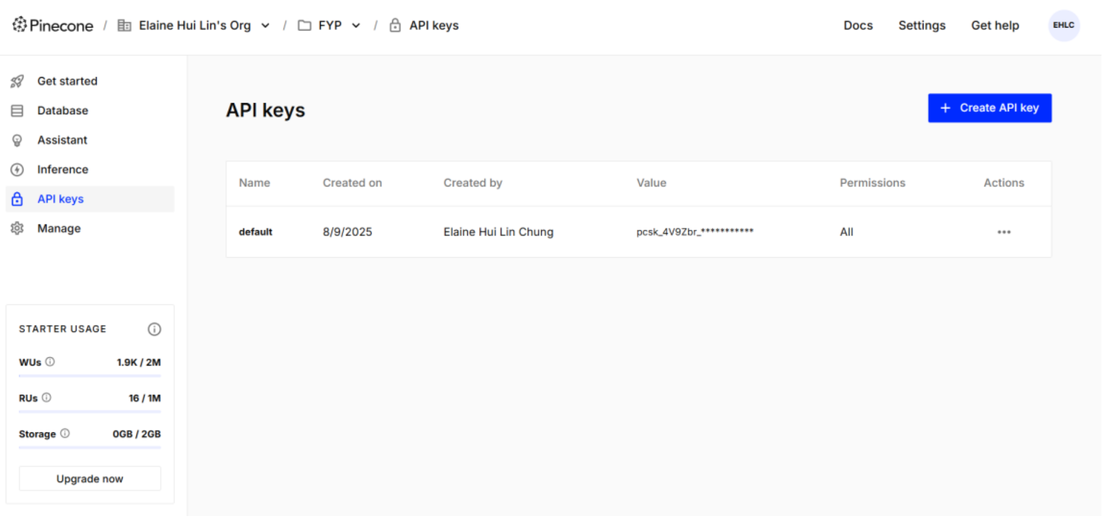
Alternatively, the system is also capable of creating the index programmatically. This is achieved through the *createPineconeIndex* function as shown below, which checks for the existence of the specified index and creates it if it does not exist. The programmatic configuration specifies the vector dimension, the cosine similarity metric, and tags the index with the embedding model used. This ensures that the index is always available for inserting and retrieving embeddings as part of the RAG pipeline.



```
import { Index, Pinecone, RecordMetadata } from "@pinecone-database/pinecone";
import logger from "../util/logger";

export async function createPineconeIndex(
  pc: Pinecone,
  indexName: string,
  vectorDimension: number,
) {
  try {
    // Initiate index existence check
    logger.info(`Checking "${indexName}"...`);
    const existingIndexes = await pc.listIndexes();

    if (existingIndexes.indexes?.some((idx) => idx.name === indexName)) {
      logger.info(`Index "${indexName}" already exists.`);
    } else {
      // Create index
      logger.info(`Creating "${indexName}"...`);
      await pc.createIndex({
        name: indexName,
        dimension: vectorDimension,
        metric: "cosine",
        spec: {
          serverless: {
            cloud: "aws",
            region: "us-east-1",
          },
        },
        tags: {
          embedding_model: "text-embedding-3-small",
        },
      });
      logger.info(`✅ Index "${indexName}" created successfully!`);
    }
  } catch (error) {
    throw new Error(`❌ Error in createPineconeIndex: ${error}`);
  }
}
```

Figure 5.3.3 createPineconeIndex function implementation

Then, an API key is created through Pinecone's web interface, copied and stored inside an .env file.



API keys

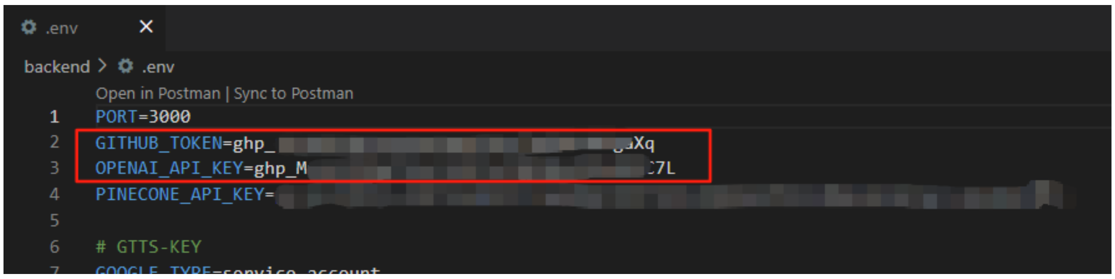
Name	Created on	Created by	Value	Permissions	Actions
default	8/9/2025	Elaine Hui Lin Chung	pcsk_4V9Zbr_***********	All	•••

Figure 5.3.4 Screenshot of Pinecone API key creation

5.3.2.3 LLM Access via GitHub Models

Free GitHub Models were used as the LLM provider. An API token is generated from GitHub and configured as the authentication method. The token was granted *models:read* permissions to authorize access to the hosted models.

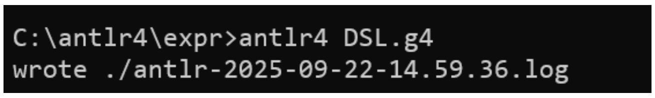
Dependencies for the client libraries are installed and saved to the package.json inside the project folder. Additionally, environment variables are created in the project to securely store the API credentials. The tokens are then referenced in the client code using the .env file, as shown below:



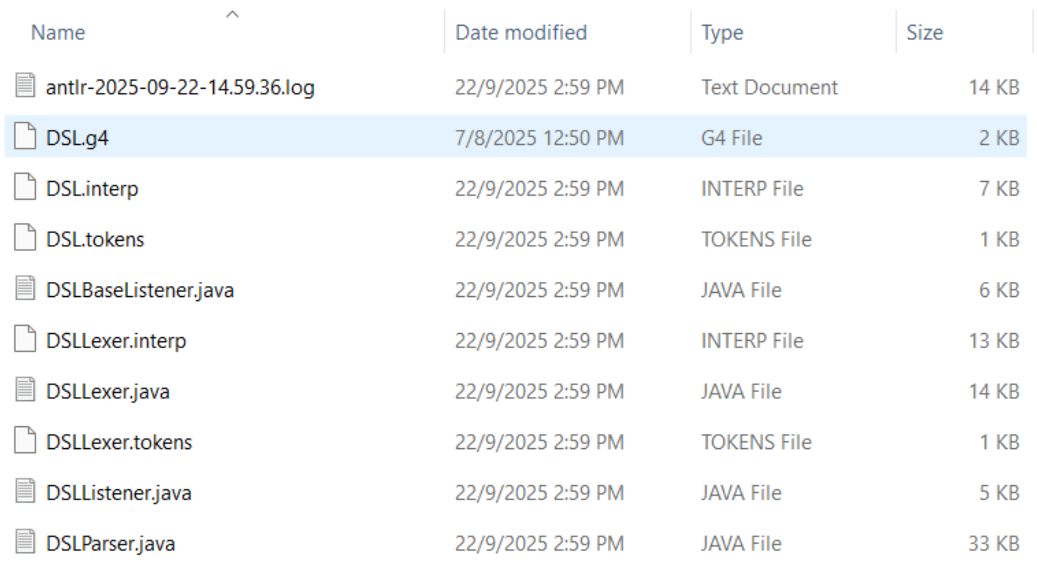
```
PORT=3000
GITHUB_TOKEN=ghp_[redacted]Xq
OPENAI_API_KEY=ghp_M[redacted]7L
PINECONE_API_KEY=[redacted]

# GTTS-KEY
GOOGLE_TYPE=service_account
```

5.3.2.4 ANTLR4 configuration

The grammar file is compiled using ANTLR4 script.

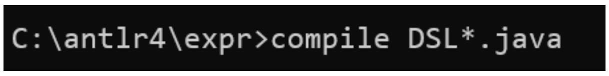


```
C:\antlr4\expr>antlr4 DSL.g4
wrote ./antlr-2025-09-22-14.59.36.log
```

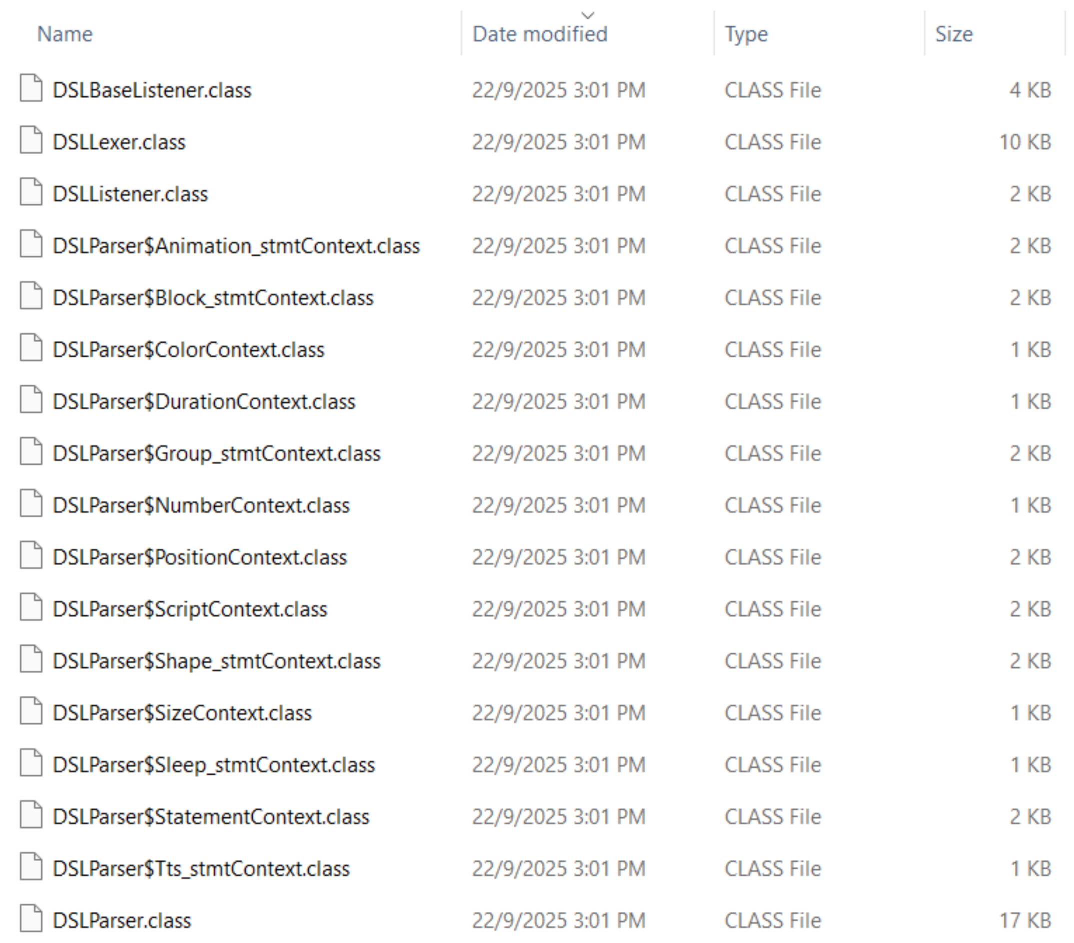


Name	Date modified	Type	Size
antlr-2025-09-22-14.59.36.log	22/9/2025 2:59 PM	Text Document	14 KB
DSL.g4	7/8/2025 12:50 PM	G4 File	2 KB
DSL.interp	22/9/2025 2:59 PM	INTERP File	7 KB
DSL.tokens	22/9/2025 2:59 PM	TOKENS File	1 KB
DSLBaseListener.java	22/9/2025 2:59 PM	JAVA File	6 KB
DSLLexer.interp	22/9/2025 2:59 PM	INTERP File	13 KB
DSLLexer.java	22/9/2025 2:59 PM	JAVA File	14 KB
DSLLexer.tokens	22/9/2025 2:59 PM	TOKENS File	1 KB
DSLListener.java	22/9/2025 2:59 PM	JAVA File	5 KB
DSLParser.java	22/9/2025 2:59 PM	JAVA File	33 KB

Next, the the parser and all java files are compiled to generate .class files to test the grammar. To test the lexer and parser of the grammar, grun command is used.

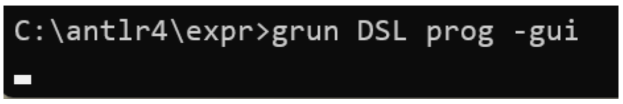


```
C:\antlr4\expr>compile DSL*.java
```

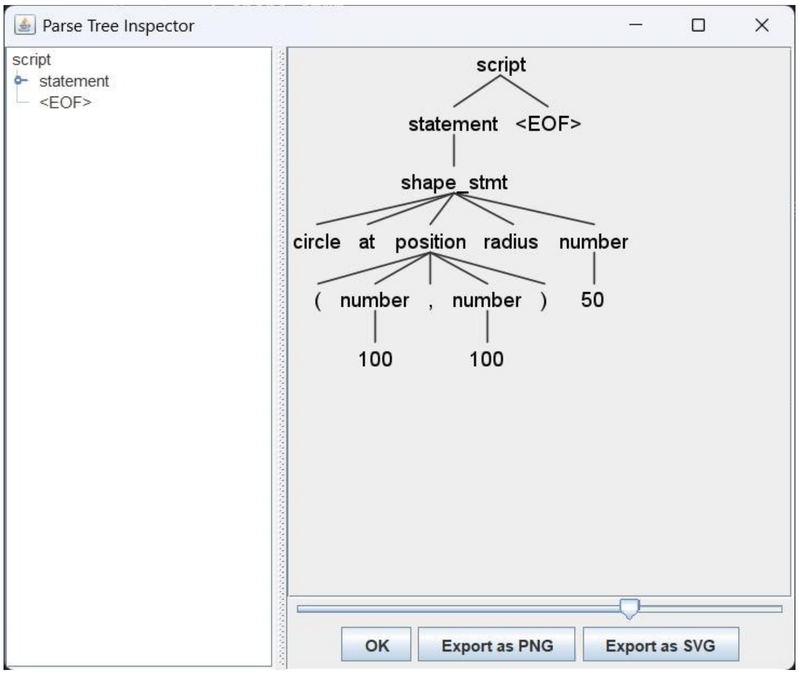


Name	Date modified	Type	Size
DSLBaseListener.class	22/9/2025 3:01 PM	CLASS File	4 KB
DSLLexer.class	22/9/2025 3:01 PM	CLASS File	10 KB
DSLListener.class	22/9/2025 3:01 PM	CLASS File	2 KB
DSLParser$Animation_stmtContext.class	22/9/2025 3:01 PM	CLASS File	2 KB
DSLParser$Block_stmtContext.class	22/9/2025 3:01 PM	CLASS File	2 KB
DSLParser$ColorContext.class	22/9/2025 3:01 PM	CLASS File	1 KB
DSLParser$DurationContext.class	22/9/2025 3:01 PM	CLASS File	1 KB
DSLParser$Group_stmtContext.class	22/9/2025 3:01 PM	CLASS File	2 KB
DSLParser$NumberContext.class	22/9/2025 3:01 PM	CLASS File	1 KB
DSLParser$PositionContext.class	22/9/2025 3:01 PM	CLASS File	2 KB
DSLParser$ScriptContext.class	22/9/2025 3:01 PM	CLASS File	2 KB
DSLParser$Shape_stmtContext.class	22/9/2025 3:01 PM	CLASS File	2 KB
DSLParser$SizeContext.class	22/9/2025 3:01 PM	CLASS File	1 KB
DSLParser$Sleep_stmtContext.class	22/9/2025 3:01 PM	CLASS File	1 KB
DSLParser$StatementContext.class	22/9/2025 3:01 PM	CLASS File	2 KB
DSLParser$Tts_stmtContext.class	22/9/2025 3:01 PM	CLASS File	1 KB
DSLParser.class	22/9/2025 3:01 PM	CLASS File	17 KB

Upon success, a graphical user interface (GUI) is shown with the corresponding parse tree of the DSL.



```
C:\antlr4\expr>grun DSL prog -gui
```



5.3.3 Environment Variables

A plain text file (.env) is used to store environment variables for the application. This ensures that sensitive information, such as API keys and tokens, is not hardcoded in the source code or pushed into version control. By externalizing these secrets, the system improves security, reduces the risk of public exposure, and simplifies configuration across different environments

The .env file includes the following entries:-

```
PORT=3000
GITHUB_TOKEN=<your_github_token>
OPENAI_API_KEY=<your_openai_api_key>
PINECONE_API_KEY=<your_pinecone_api_key>

# logging
LOG_LEVEL=info # error | warn | info | http | verbose | debug | silly
LOGTAIL_SOURCE_TOKEN=<your_logtail_source_token>
LOGTAIL_ENDPOINT=<your_logtail_endpoint>
```

To make these variables accessible within the application, the **dotenv** library is used. When the server starts, dotenv automatically loads the contents of the .env file and injects the variables into process.env.

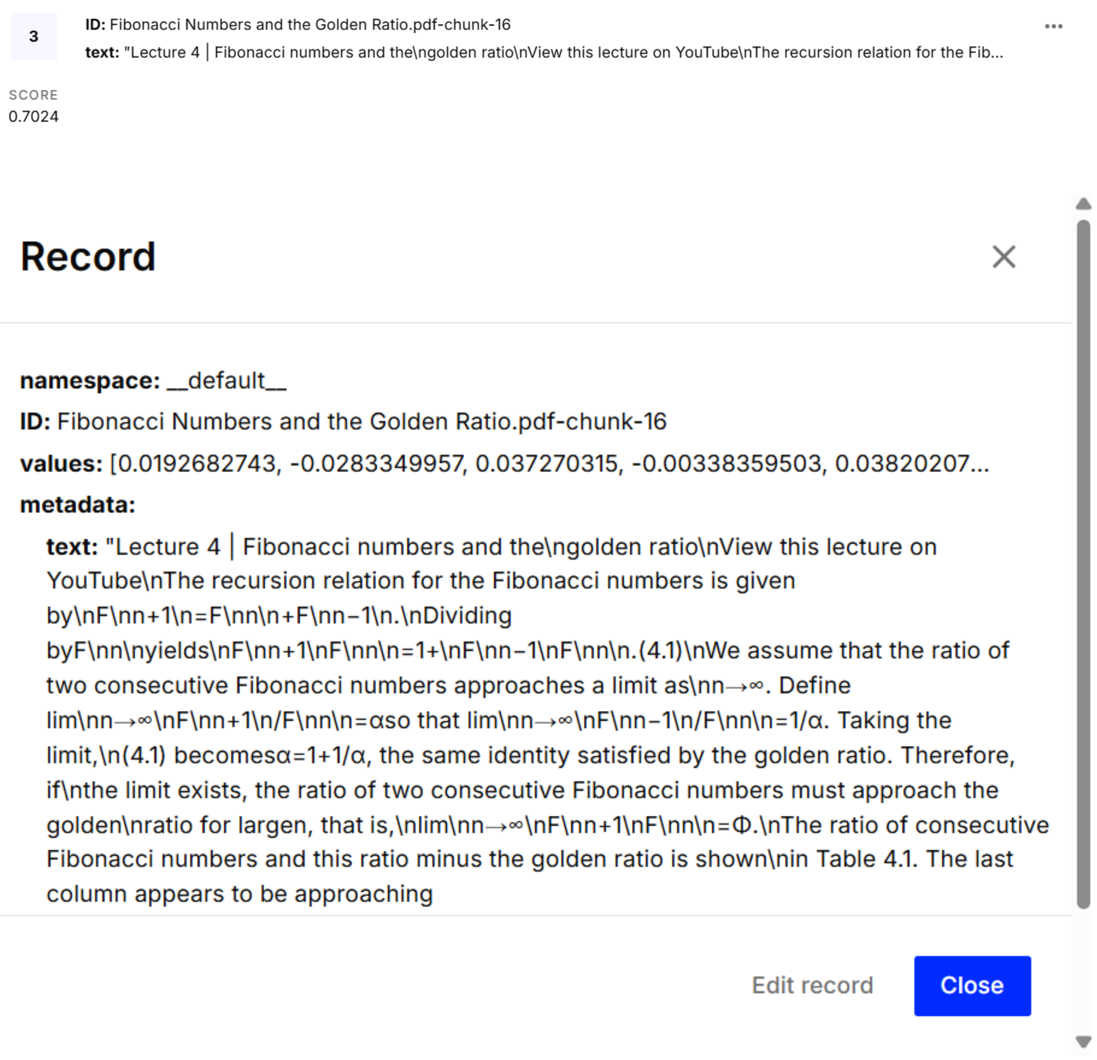
5.4 System Operation

This section illustrates how the system operates from the user's perspective, showing the interaction flow between different components. The process begins on the Home Page, where the user provides input, and continues on the Editor Page, where generated scripts can be refined and executed to produce the final educational animation.

5.4.1 Home Page

The Home Page serves as the primary entry point of the system. Users are presented with a simple interface where they can submit a natural language prompt describing the educational concept they wish to visualize. Optionally, users may also upload supporting learning material in PDF format, which the system processes into embeddings for semantic retrieval.

Once the user submits the input, the system's backend retrieves relevant content from the uploaded corpus if available, combines it with the prompt, and forwards it to the language model. Figure 5.4.1 shows the sample record fetch from Pinecone Vector Database based on the highest score. The LLM then generates a DSL script that encodes both the animation steps and narration instructions.


3
SCORE
0.7024

ID: Fibonacci Numbers and the Golden Ratio.pdf-chunk-16
text: "Lecture 4 | Fibonacci numbers and the\ngolden ratio\nView this lecture on YouTube\nThe recursion relation for the Fib...

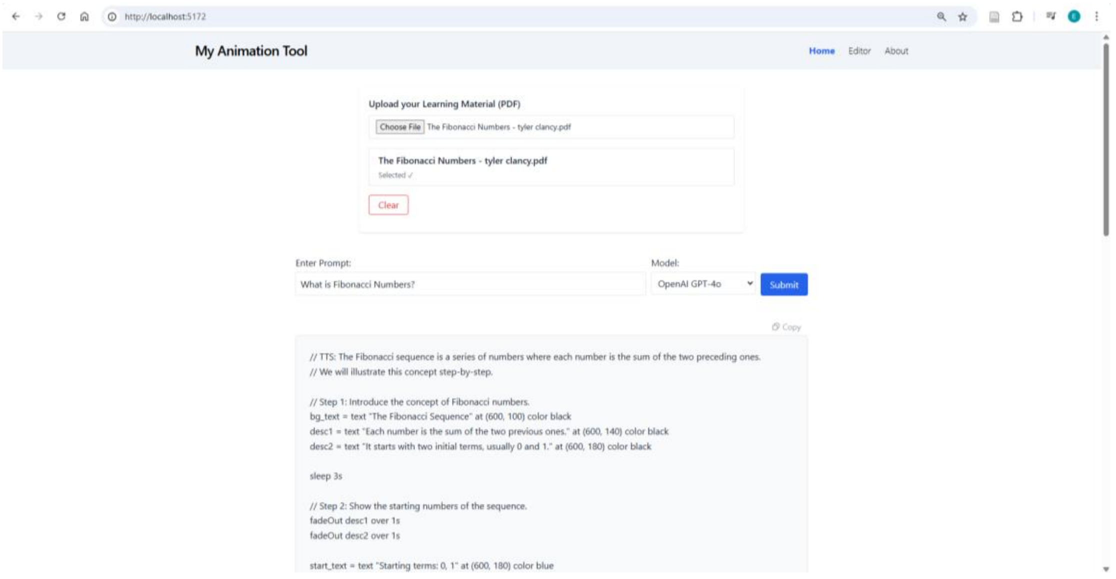
Record

namespace: __default__
ID: Fibonacci Numbers and the Golden Ratio.pdf-chunk-16
values: [0.0192682743, -0.0283349957, 0.037270315, -0.00338359503, 0.03820207...
metadata:
text: "Lecture 4 | Fibonacci numbers and the\ngolden ratio\nView this lecture on YouTube\nThe recursion relation for the Fibonacci numbers is given by\nF\nn+1\n=F\nn\n+F\nn−1\n.\nDividing byF\nn\nyields\nF\nn+1\nF\nn\n=1+\nF\nn−1\nF\nn\n.(4.1)\nWe assume that the ratio of two consecutive Fibonacci numbers approaches a limit as\nn→∞. Define lim\nn→∞\nF\nn+1\n/F\nn\n=αso that lim\nn→∞\nF\nn−1\n/F\nn\n=1/α. Taking the limit,\n(4.1) becomesα=1+1/α, the same identity satisfied by the golden ratio. Therefore, if\nthe limit exists, the ratio of two consecutive Fibonacci numbers must approach the golden\nratio for largen, that is,\nlim\nn→∞\nF\nn+1\nF\nn\n=Φ.\nThe ratio of consecutive Fibonacci numbers and this ratio minus the golden ratio is shown\nin Table 4.1. The last column appears to be approaching

Edit record
Close

Figure 5.4.1 Sample record stored in PineconeDB

This page emphasizes ease of use by hiding the technical details from the user, ensuring that even users with minimal technical background can interact with the system effectively.


http://localhost:5172
My Animation Tool
Home
Editor
About
Upload your Learning Material (PDF)
Choose File
The Fibonacci Numbers - tyler clancy.pdf
The Fibonacci Numbers - tyler clancy.pdf
Selected ✓
Clear
Enter Prompt:
What is Fibonacci Numbers?
Model:
OpenAI GPT-4o
Submit
Copy

```
// TTS: The Fibonacci sequence is a series of numbers where each number is the sum of the two preceding ones.
// We will illustrate this concept step-by-step.

// Step 1: Introduce the concept of Fibonacci numbers.
bg_text = text "The Fibonacci Sequence" at (600, 100) color black
desc1 = text "Each number is the sum of the two previous ones." at (600, 140) color black
desc2 = text "It starts with two initial terms, usually 0 and 1." at (600, 180) color black

sleep 3s

// Step 2: Show the starting numbers of the sequence.
fadeOut desc1 over 1s
fadeOut desc2 over 1s

start_text = text "Starting terms: 0, 1" at (600, 180) color blue
```

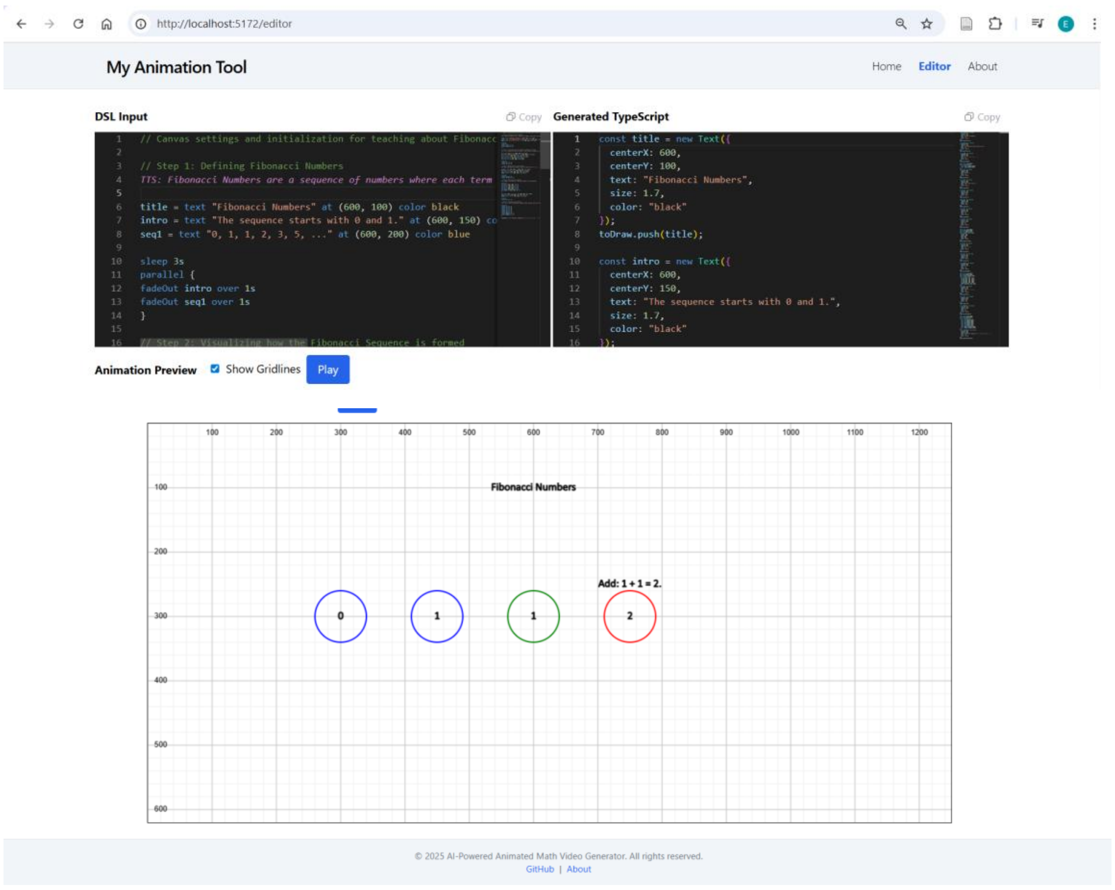
Figure 5.4.2 Home Page User Interface

5.4.2 Editor Page

The Editor Page provides an environment for reviewing and refining the generated DSL script. At the center of this interface is the **Monaco editor**, which allows the user to view and optionally edit the script. Users can correct errors, adjust animations, or fine-tune the sequence before rendering.

The DSL script is continuously monitored for changes. When a change is detected, the system validates the script before proceeding. The script is then parsed using ANTLR4 and translated into TypeScript objects through a custom visitor (DSLToTSVisitor). The generated TypeScript code is displayed in a read-only panel for reference. On the backend, this code is compiled and executed to produce synchronized animations alongside voice narration. Finally, a playback interface is provided to preview the output.

The design of the DSL and its web-based editor was conceptually inspired by existing DSL frameworks, particularly, PlantUML and its online web editor [20]. The interface of PlantUML editor can be found in the Appendix A.2. However, unlike PlantUML which focuses on live UML diagram generation, the proposed DSL is purpose-built for educational animations and integrates with a TypeScript-based animation engine. Tthe animation editor in this system renders the corresponding TypeScript code and animation in real-time as the user inputs DSL commands. This approach was adopted to provide immediate visual feedback and improve user interaction with the educational content generation process.



```
// Canvas settings and initialization for teaching about Fibonacc

// Step 1: Defining Fibonacci Numbers
TTS: Fibonacci Numbers are a sequence of numbers where each term

title = text "Fibonacci Numbers" at (600, 100) color black
intro = text "The sequence starts with 0 and 1." at (600, 150) co
seq1 = text "0, 1, 1, 2, 3, 5, ..." at (600, 200) color blue

sleep 3s
parallel {
fadeOut intro over 1s
fadeOut seq1 over 1s
}

// Step 2: Visualizing how the Fibonacci Sequence is formed
```

```
const title = new Text({
  centerX: 600,
  centerY: 100,
  text: "Fibonacci Numbers",
  size: 1.7,
  color: "black"
});
toDraw.push(title);

const intro = new Text({
  centerX: 600,
  centerY: 150,
  text: "The sequence starts with 0 and 1.",
  size: 1.7,
  color: "black"
});
```

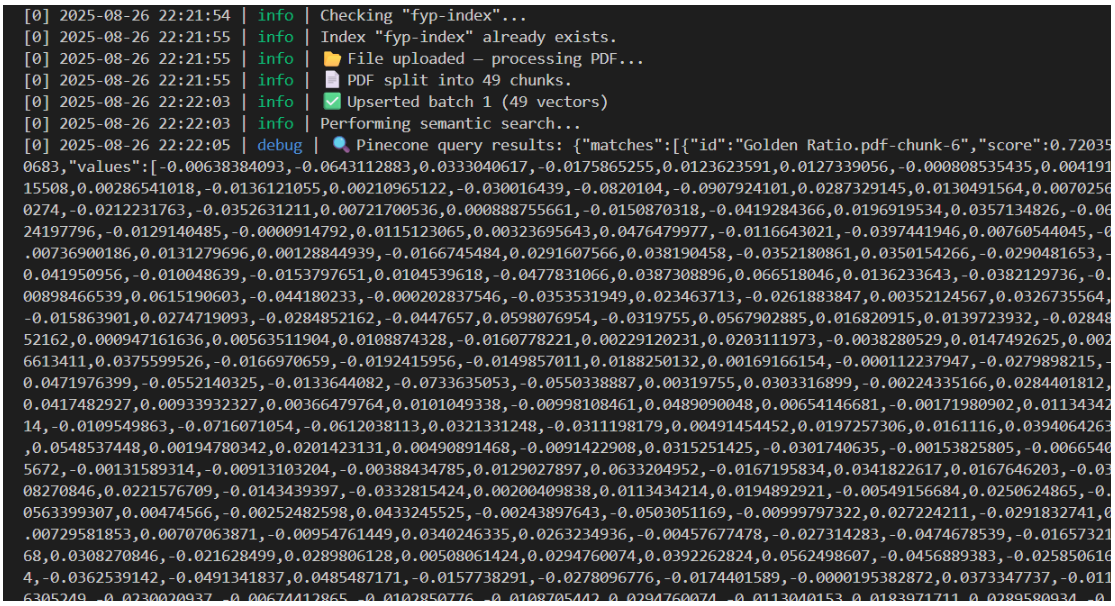
Figure 5.4.3 Editor Page User Interface

5.4.2.1 Animation Execution and Output

Once the DSL script has been validated and translated into TypeScript code, the system proceeds with executing the animation pipeline. The CanvasRenderer and Animator components work together to render the defined shapes such as circles, rectangles, triangles, and text onto the HTML Canvas. Animation instructions, including operations like *draw*, *move*, *scale*, *fadeIn*, and *fadeOut*, are then applied sequentially to create smooth, continuous visual effects.

If narration is enabled, the script's text lines are sent to the Google TTS API, which generates the corresponding audio files. These audio segments are synchronized with the animation timeline, ensuring that the spoken narration aligns with the visual presentation.

The final output is an animated educational video that plays directly within the browser. To keep the system lightweight and cost-efficient, it is designed to be single-use, requiring no database or persistent storage. This design ensures low overhead while still producing engaging audio-visual content for learners.



```
[0] 2025-08-26 22:21:54 | info | Checking "fyp-index"...
[0] 2025-08-26 22:21:55 | info | Index "fyp-index" already exists.
[0] 2025-08-26 22:21:55 | info | 📂 File uploaded – processing PDF...
[0] 2025-08-26 22:21:55 | info | 📄 PDF split into 49 chunks.
[0] 2025-08-26 22:22:03 | info | ✅ Upserted batch 1 (49 vectors)
[0] 2025-08-26 22:22:03 | info | Performing semantic search...
[0] 2025-08-26 22:22:05 | debug | 🔍 Pinecone query results: {"matches":[{"id":"Golden Ratio.pdf-chunk-6","score":0.72035
0683,"values":[-0.00638384093,-0.0643112883,0.0333040617,-0.0175865255,0.0123623591,0.0127339056,-0.000808535435,0.004191
15508,0.00286541018,-0.0136121055,0.00210965122,-0.030016439,-0.0820104,-0.0907924101,0.0287329145,0.0130491564,0.0070256
0274,-0.0212231763,-0.0352631211,0.00721700536,0.000888755661,-0.0150870318,-0.0419284366,0.0196919534,0.0357134826,-0.06
24197796,-0.0129140485,-0.0000914792,0.0115123065,0.00323695643,0.0476479977,-0.0116643021,-0.0397441946,0.00760544045,-0
.00736900186,0.0131279696,0.00128844939,-0.0166745484,0.0291607566,0.038190458,-0.0352180861,0.0350154266,-0.0290481653,-
0.041950956,-0.010048639,-0.0153797651,0.0104539618,-0.0477831066,0.0387308896,0.066518046,0.0136233643,-0.0382129736,-0.
00898466539,0.0615190603,-0.044180233,-0.000202837546,-0.0353531949,0.023463713,-0.0261883847,0.00352124567,0.0326735564,
-0.015863901,0.0274719093,-0.0284852162,-0.0447657,0.0598076954,-0.0319755,0.0567902885,0.016820915,0.0139723932,-0.02848
52162,0.000947161636,0.00563511904,0.0108874328,-0.0160778221,0.00229120231,0.0203111973,-0.0038280529,0.0147492625,0.002
6613411,0.0375599526,-0.0166970659,-0.0192415956,-0.0149857011,0.0188250132,0.00169166154,-0.000112237947,-0.0279898215,-
0.0471976399,-0.0552140325,-0.0133644082,-0.0733635053,-0.0550338887,0.00319755,0.0303316899,-0.00224335166,0.0284401812,
0.0417482927,0.00933932327,0.00366479764,0.0101049338,-0.00998108461,0.0489090048,0.00654146681,-0.00171980902,0.01134342
14,-0.0109549863,-0.0716071054,-0.0612038113,0.0321331248,-0.0311198179,0.00491454452,0.0197257306,0.0161116,0.0394064263
,0.0548537448,0.00194780342,0.0201423131,0.00490891468,-0.0091422908,0.0315251425,-0.0301740635,-0.00153825805,-0.0066540
5672,-0.00131589314,-0.00913103204,-0.00388434785,0.0129027897,0.0633204952,-0.0167195834,0.0341822617,0.0167646203,-0.03
08270846,0.0221576709,-0.0143439397,-0.0332815424,0.00200409838,0.0113434214,0.0194892921,-0.00549156684,0.0250624865,-0.
0563399307,0.00474566,-0.00252482598,0.0433245525,-0.00243897643,-0.0503051169,-0.00999797322,0.027224211,-0.0291832741,0
.00729581853,0.00707063871,-0.00954761449,0.0340246335,0.0263234936,-0.00457677478,-0.027314283,-0.0474678539,-0.01657321
68,0.0308270846,-0.021628499,0.0289806128,0.00508061424,0.0294760074,0.0392262824,0.0562498607,-0.0456889383,-0.025850616
4,-0.0362539142,-0.0491341837,0.0485487171,-0.0157738291,-0.0278096776,-0.0174401589,-0.0000195382872,0.0373347737,-0.011
6305249,-0.0230020937,-0.00674412865,-0.0102850776,-0.0108705442,0.0294760074,-0.0113040153,0.0183971711,0.0289580934,-0.
```

Figure 5.4.4 Full Log in One Animation Generation Pipeline Lifecycle

A sample DSL script and its corresponding translated TypeScript code are provided in Appendix A.3. This example demonstrates an additional benefit of the DSL, in which it significantly reduces code verbosity and code length by more than 50%. Specifically, 88 lines of DSL code were sufficient to express the same functionality that required 181 lines of TypeScript. This reduction illustrates the conciseness and expressive power of the DSL, potentially improving readability, maintainability, and efficiency in authoring educational animations.

5.5 Implementation Issues and Challenges

During the implementation phase, several technical and practical challenges emerged. This section discusses the technical and non-technical challenges faced throughout the development of the project and its solution.

One of the earliest issues was related to the **design of the DSL**. The initial grammar definitions created ambiguities that caused parsing conflicts in ANTLR, particularly when interpreting nested instructions or handling optional parameters. This led to difficulties in generating consistent parse trees. The issue was mitigated by refining the grammar rules and adopting a stricter definition of allowable syntax. In addition, a visitor-based approach was implemented to systematically translate DSL commands into TypeScript objects, which improved reliability and consistency in execution.

Another significant challenge was associated with the **grouping functions in animation engine**. While basic animations such as draw, move, and fadeIn were straightforward, grouped animations such as scaleAndMove or scaleAndRotate required careful handling. Initially, these animations were incorrectly applied to individual objects, leading to runtime errors and unexpected behavior. To address this, validation logic was introduced to ensure that grouped transformations were only applied to group objects. This enforced type safety at the DSL execution level and preserved the intended structure of animations.

Furthermore, a key challenge was the **unpredictability of LLM outputs**. The generation model occasionally produced inconsistent DSL instructions, ranging from minor syntax errors to invalid animation commands. Although it could produce high-level instructions, it sometimes generated commands outside the supported grammar or introduced formatting errors, which caused failures during script translation and execution. To address this issue, model configuration parameters such as temperature and top-P were adjusted to reduce randomness in the LLM's output. Lowering these values constrained the model to generate more deterministic and consistent scripts, which helped mitigate errors in DSL syntax and unsupported commands. However, these adjustments alone were insufficient; therefore, system prompt and few-shot prompting were used to guide the model with explicit grammar rules and working examples, which improved adherence to the DSL structure.

Performance constraints also became evident when processing large input prompts, such as entire textbook chapters. These large queries caused significant latency and memory usage, which was unsuitable for real-time animation generation. To overcome this, a chunking mechanism was introduced in combination with embeddings and semantic retrieval. This allowed the system to break down large inputs into smaller, manageable segments that could be processed incrementally while still preserving context.

Another challenge was **synchronizing the generated animations with voice-over narration** from Google Text-to-Speech. Early implementations resulted in mismatches, where the narration either lagged behind or advanced ahead of the visuals. To resolve this, timestamp markers were introduced within the animation pipeline, enabling tighter alignment between audio playback and animation scene transitions. This ensured that the final video output provided a coherent and engaging learning experience.

A further limitation was the **truncation of model responses** when outputs exceeded the maximum token limit, which is 65535 tokens. This occurred most frequently when generating lengthy DSL scripts or detailed explanations, resulting in incomplete outputs. To mitigate this, the system was adjusted to split requests into smaller sub-tasks where possible, ensuring more reliable responses.

In general, the development of the project was a continuous process of adaptation and debugging. Overcoming the challenges encountered during implementation provided valuable insights that influenced the system's architecture and functionality. Each issue ultimately contributed to a more refined, resilient, and practical solution that remained aligned with the overall objectives of the project.

5.6 Concluding Remark

This chapter has outlined the configuration and implementation of the automated educational video generation system, covering the setup of the development environment, integration of core components, and execution of the system pipeline. The implementation process highlighted both the technical achievements and the challenges faced, such as managing environment dependencies, integrating large language models, and ensuring reliable translation of DSL scripts into animations. Through systematic problem-solving and iterative refinement, the system evolved into a functional prototype capable of generating synchronized educational animations with narration.

The insights gained during this phase not only ensured a working implementation but also revealed opportunities for future enhancements, including optimization of rendering efficiency, improved model guidance, and expanded user interaction features.

Chapter 6

System Evaluation and Discussion

In this chapter, the evaluation of the system was conducted to verify its correctness, performance, and overall reliability in generating educational animations from high-level user prompts. Testing was performed based on the core use cases identified in Chapter 3, with emphasis on both functional accuracy and system efficiency.

6.1 System Testing and Performance Metrics

To evaluate the system, a series of tests were conducted focusing on the correctness, accuracy, and usability of the LLM-driven animation pipeline. The evaluation employed both quantitative metrics and qualitative assessments.

Table 6.1.1 System Performance

Metric	**Description**	**Measurement Approach**	**Results and Observation**
Syntax correctness	Percentage of generated scripts that complied with the custom animation library	5 test prompts evaluated	80% valid on first attempt
Conceptual accuracy	Extent to which animations correctly represented the intended mathematical idea	Manual inspection and peer review	Generally accurate, but errors in concurrent motions
Sequential vs. concurrent	Correct use of parallel vs. sequential animations	6 prompts requiring concurrency	66.67% correct, 33.33% sequential misinterpretation
Average response latency	Time from user prompt submission to rendered animation	Timed over 6 test runs	22.95s average
Usability	Ease of use and clarity of the application interface	Informal peer testing with 10 participants	Rated 4.5/5 overall

The results indicate that the system performs reliably in generating animations, though issues remain with concurrency handling and occasional hallucinated properties. Responsiveness was acceptable for prototype use, and users found the interface simple and intuitive.

6.2 Testing Setup and Result

Each use case are tested individually using unit testing. The results are as shown in Table 6.2.1.

Table 6.2.1 Testing of Individual Use Cases

Use Case	Input	Expected Output	Actual Output	Result
UC001	Upload valid PDF	Embeddings stored	Success	Pass
UC002	Submit empty prompt	Error message	Error displayed	Pass
UC003	Valid DSL script	Smooth animation	Worked with narration sync	Pass
UC004	Invalid DSL	Parse failed + line number	Handled correctly	Pass

6.2.1 Testing Setup

Test Environment:

The environment used for testing is as shown in the table below.

Table 6.2.2 Testing Environment

Hardware	Laptop with Intel i7 CPU, 16GB RAM, Windows 11 Version 24H2
Software	Node.js, React (frontend), Express (backend), GitHub-hosted LLM API
Browser	Chrome 139.0.7258.139 (Official Build) (64-bit) (cohort: Stable)

Test Dataset:

During testing, 2 simple geometry prompts, 2 algorithmic and 2 complex mathematics prompts are used.

Table 6.2.3 Test Data

Category of prompts	Test input data
geometry	What is symmetry in a square?
	What is a vector?
algorithmic	Explain Kruskal's algorithm with animations.
	Demonstrate topological sorting with a DAG.
complex mathematics	Visualize the Fibonacci sequence.
	Visualize Depth-First Search (DFS) on a tree.

Evaluation Method:

- Scripts checked for syntax compliance against the custom animation library
- Animations inspected for conceptual accuracy and correctness of sequential/concurrent motions
- Response time measured using browser developer tools in Chrome
- Informal peer review with 10 participants to gather usability feedback

6.2.2 Results and Analysis

Category of prompts	Test input data	Syntax Correctness on first attempt	LLM Response Time (s)	Translation Time (s)	Total response time (s)
geometry	What is symmetry in a square?	Valid	12.29	3.03	15.32
	What is a vector?	Valid	17.86	8.47	26.33
algorithmic	Explain Kruskal's algorithm with animations.	Invalid	18.41	4.98	23.39
	Demonstrate topological sorting with a DAG	Invalid	23.99	3.29	27.28
complex mathematics	Visualize the Fibonacci sequence.	Valid	17.81	1.81	19.62
	Visualize Depth-First Search (DFS) on a tree.	Valid	20.58	5.16	25.74
		Average	**18.49**	**4.46**	**22.95**

Correctness of Generated Scripts:

- Out of 6 prompts tested, 4 produced runnable scripts without modification.
- 2 required minor manual corrections (i.e., property naming mismatches and incorrect identifier assignment).
- 1 failed due to hallucinated or unsupported properties.

Accuracy of Animations:

- Most animations successfully conveyed the intended concept.
- Misuse of sequential instead of concurrent animations occurred in 4/10 concurrency-based tests.
- Motion smoothness was generally acceptable, though timing inconsistencies were noted in fast transitions.

Performance:

- Average time to generate and render animations: 8–12 seconds.
- Rendering pipeline itself was efficient (<2 seconds); most latency was due to LLM response time.

Usability Feedback:

- Testers found the web application intuitive and easy to use.
- Average usability rating: 4.2/5.
- Suggested improvements: progress indicators during rendering, and support for multi-turn prompting.

Observation

The testing confirmed that the system is capable of generating animations from natural language prompts with a reasonably high success rate. While the modular pipeline and interface functioned as intended, limitations remain in terms of handling parallel animations and error correction. Nonetheless, the system demonstrates the feasibility of LLM-driven educational animation, fulfilling the majority of the project objectives.

6.3 Project Challenges

i. Model Output Inconsistencies with Library Syntax and Usage

The primary challenge encountered during development was ensuring the **correctness and accuracy** of the generated animation code. Despite the use of few-shot examples and system prompts, the LLM occasionally produced outputs that violated the syntax and design rules of the custom animation library. A recurring issue was hallucinated properties, where the model referenced attributes that were not defined in the library's shape classes. For instance, the model sometimes attempted to apply a non-existent linewidth property to Line objects. Similarly, incorrect property naming was observed; the model occasionally used parameters such as x1, y1, x2, y2 instead of the required startX, startY, endX, endY for line coordinates, despite explicit demonstrations in the prompt. Another major source of inaccuracy was the misuse of sequential and concurrent animations. In cases where parallel execution was required (e.g., rotating clock hands simultaneously), the model frequently generated sequential animations, resulting in unintended and unnatural behaviour. To address these issues, several strategies were employed. A Retrieval-Augmented Generation (RAG) mechanism was integrated to provide the model with accurate library references during code generation. This was combined with manual correction, targeted prompt engineering, and model parameter tuning, all of which helped guide the model toward producing more consistent and syntactically correct outputs. Furthermore, the introduction of a well-defined DSL played a critical role in enforcing syntax, semantics, and animation rules, thereby reducing the likelihood of invalid or misaligned code.

ii. API Rate Limits on GitHub-Hosted LLM Models

Another challenge was the **API usage limits** imposed by the GitHub Marketplace, where the project accessed high-tier LLMs. Free-tier models were restricted to **50 requests per day per model**, which significantly constrained the testing and iteration phases. Since development often required frequent prompting, debugging, and fine-tuning, these limits slowed progress and necessitated careful planning of work sessions. In future iterations, this bottleneck could be addressed by **migrating to paid API plans** or hosting **private instances** of the models to ensure consistent availability.

iii. Lack of Feedback Mechanism for Model Refinement

A further limitation was the absence of a **feedback mechanism** to improve the quality of generated scripts. At the current stage, the system only supports single-turn prompting, where the model generates an output once per request without opportunities for iterative refinement. This restricts the ability to correct mistakes or guide the model toward better outputs within the same interaction. As a result, when the generated scripts contained issues such as syntax violations, hallucinated properties, misused library functions, or inappropriate animation timing, refinement relied solely on manual prompt engineering and human intervention. In future iterations, this challenge could be mitigated by enabling multi-turn prompting, which would allow interactive refinement of outputs, or by establishing a more systematic feedback loop through methods such as reinforcement learning from human feedback (RLHF). These improvements would help the model adapt and gradually produce more accurate and reliable results.

6.4 Objectives Evaluation

This section assesses how well the project met its initial objectives, which were specified during the project proposal phase. The primary objective of this project was to develop and integrate animation tools controlled by LLMs to generate educational animations for Mathematics. Overall, this objective was achieved. The system successfully demonstrated the ability of LLMs to interpret prompts and generate animations, but certain challenges like correctness of outputs constrained the level of automation and accuracy.

i. Development of a Modular Animation Pipeline

This objective was achieved. A modular pipeline was implemented, consisting of distinct modules for shape transformations, motion generation, and scene composition. Each module exposed well-defined APIs, enabling structured control by the LLM. The modular design proved effective in ensuring reusability and scalability, and additional features such as grouping, grid system were successfully incorporated without disrupting the existing framework.

ii. Implementation of a Motion Generation System

This objective was partially achieved. A library of predefined animations such as creation, transformation, scaling, fading, and interpolation was successfully developed and integrated into the system. These animations provided the foundation for LLM-driven generation. However, while the animations produced were visually functional, the system occasionally struggled with ensuring smoothness and naturalness of motion, especially when parallel animations were required. Further refinement of motion algorithms would be needed to fully realize this objective.

iii. Development of a Demonstration Application

This objective was achieved. A functional application was created to integrate the modular pipeline and enable user interaction. The application allowed users to input prompts and generate corresponding animations, thereby validating the feasibility of LLM-controlled animation. Additionally, a simple interface was implemented to present results, making the tool accessible for educators and learners. Nevertheless, the system currently operates on a single-turn prompting model without feedback loops, which limits the extent of error correction and iterative improvement.

6.5 Concluding Remark

Based on the results, the proposed method, which involves using an LLM to generate animations and visual explanations from educational prompts, is technically feasible. Functional elements like shape rendering, text transitions, and basic sequencing are present in the animation system, which can comprehend structured scripts. The LLM produces outputs that provide clear step-by-step guidance for less complex ideas. While some challenging topics were impeded by limitations in both LLM understanding and system rendering abilities, these concerns are predominantly engineering issues that can be resolved through improved prompt design, system refinement over time and eventually with better iterative development.

Chapter 7

Conclusion and Recommendation

This chapter presents the conclusion of the project and provides a final reflection on its outcomes. It revisits the main objectives set at the beginning, evaluates how effectively they were achieved, and highlights the contributions made to the field of automated educational content generation. In addition, the chapter outlines the limitations encountered during development and proposes future directions for extending the system's capabilities. Together, these elements provide closure to the research while emphasizing its significance and potential impact.

7.1 Conclusion

To conclude, the main problem addressed by this project was the need for an intuitive, automated way to create mathematical animations without requiring users to be well-versed in programming or specialized animation software. Motivated by the growing demand for accessible educational content, the project focused on leveraging LLMs to bridge the gap between high-level user instructions and low-level animation scripting.

A key contribution of this work is the development of a lightweight, LLM-driven pipeline capable of automatically generating educational animations. The proof-of-concept demonstrates the feasibility of combining LLMs with a custom DSL and animation library to produce meaningful and dynamic content, highlighting new opportunities in automated educational media production.

This project set out to design and implement an educational animation system that leverages LLMs to transform high-level prompts into corresponding visual animations with narrations. By combining a custom DSL with an animation engine and narration engine, the system enables the automated generation of mathematics-focused educational content on-the-fly, reducing the barrier for teachers and learners who may not have prior programming experience.

The integration of RAG for grounding ensured that explanations were grounded in textbook content, thereby enhancing factual accuracy and adaptability across different curricula. The DSL and animation library provided a structured medium for translating natural language responses into executable visualizations, while the overall system demonstrated the potential of LLM-driven automation in creating engaging, accessible educational materials.

Although the current implementation achieves its core objectives, several limitations remain. The system currently supports only a limited set of shapes and animations, relies primarily on single-turn interactions, and may occasionally produce ambiguous or malformed outputs when handling complex prompts. These challenges highlight opportunities for further development, such as adopting structured JSON response schemas, extending animation capabilities, incorporating hybrid grounding methods, and supporting multi-turn conversations for more interactive learning experiences.

The system currently supports only a limited set of shapes and animations, relies primarily on single-turn interactions, and may occasionally produce ambiguous or malformed outputs when handling complex prompts. These challenges highlight opportunities for further development, such as adopting structured JSON response schemas, extending animation capabilities, incorporating hybrid grounding methods, and supporting multi-turn conversations for more interactive learning experiences.

7.2 Recommendation

There are a few crucial areas for educational animation system's future development and enhancement that should be highlighted.

Firstly, a potential enhancement would be to adopt a **structured response schema in JSON** format. By enforcing a JSON schema, the LLM's outputs could be standardized into machine-readable structures that clearly separate instructional content (DSL commands) from explanatory text. This would provide several advantages, including clarity and consistency, easy validation and extensibility. Such a schema-driven approach would make the system more robust, maintainable, and extensible, while also supporting a clearer interface between the LLM, the DSL parser, and the animation engine.

Another direction for improvement is to introduce **support for more complex animations** within the DSL and TypeScript animation library. At present, the system supports basic shapes and transformations, but future work could extend this to include advanced effects such as curved motion paths, easing functions, layering (z-index), and conditional animations. This would allow the generation of richer, more engaging visualizations that better align with professional educational tools and enhance the learner's experience.

Furthermore, the system could benefit from **hybrid grounding methods** that combine Retrieval-Augmented Generation (RAG) with lightweight fine-tuning approaches. While RAG ensures flexibility and adaptability by retrieving content dynamically from textbooks, fine-tuning could help align the model more closely with DSL grammar and reduce hallucinations. A hybrid approach would therefore balance adaptability with precision, improving both the factual grounding of explanations and the syntactic accuracy of the generated animations.

Finally, another promising area of future improvement is the introduction of **multi-turn conversational support**. Currently, the system operates on a single-turn interaction model, where each user query results in a one-time generation of explanatory text and animation instructions. Incorporating multi-turn chat would enable learners to iteratively refine explanations and animations through follow-up questions or modifications. This would not only enhance personalization and adaptability but also provide a more natural and interactive learning experience, closer to the dynamics of a real classroom environment.

References

[1] J. Gerhard and I. Kotsireas, Eds., Maple in Mathematics Education and Research: Third Maple Conference, MC 2019, Waterloo, Ontario, Canada, October 15–17, 2019, Proceedings, vol. 1125. in Communications in Computer and Information Science, vol. 1125. Cham: Springer International Publishing, 2020. doi: 10.1007/978-3-030-41258-6.

[2] I. M.E., I. M. I., O. H., A. M.H., and A. A., "The use of animation video in teaching to enhance the imagination and visualization of student in engineering drawing," IOP Conf. Ser. Mater. Sci. Eng., vol. 203, p. 012023, May 2017, doi: 10.1088/1757-899X/203/1/012023.

[3] A. Barana, A. Conte, C. Fissore, F. Floris, M. Marchisio, and M. Sacchet, "The Creation of Animated Graphs to Develop Computational Thinking and Support STEM Education," in Maple in Mathematics Education and Research, vol. 1125, J. Gerhard and I. Kotsireas, Eds., in Communications in Computer and Information Science, vol. 1125. , Cham: Springer International Publishing, 2020, pp. 189–204. doi: 10.1007/978-3-030-41258-6_14.

[4] M. Marković and I. Kaštelan, "Demonstrating the Potential of Visualization in Education with the Manim Python Library: Examples from Algorithms and Data Structures," in 2024 47th MIPRO ICT and Electronics Convention (MIPRO), Opatija, Croatia: IEEE, May 2024, pp. 625–629. doi: 10.1109/MIPRO60963.2024.10569661.

[5] D. Jeffries, R. Mohan, and C. Norris, "dsDraw: Programmable Animations and Animated Programs," in Proceedings of the 2020 ACM Southeast Conference, Tampa FL USA: ACM, Apr. 2020, pp. 39–46. doi: 10.1145/3374135.3385292.

[6] A. Helbling and D. H. Chau, "ManimML: Communicating Machine Learning Architectures with Animation," 2023, arXiv. doi: 10.48550/ARXIV.2306.17108.

[7] "A pilot study into developing animations for electrical and electronic engineering curriculum." Accessed: Jul. 25, 2024. [Online]. Available:

https://research-repository.griffith.edu.au/items/6c6bfc06-5068-44e3-b9ea-daa5971712f7

[8] C. J. Brame, “Effective Educational Videos: Principles and Guidelines for Maximizing Student Learning from Video Content,” CBE—Life Sci. Educ., vol. 15, no. 4, p. es6, Dec. 2016, doi: 10.1187/cbe.16-03-0125.

[9] K. Dziedzic, M. Barszcz, M. Paśnikowska-Łukaszuk, and J. Jankowska, “THE ROLE OF COMPUTER ANIMATION IN TEACHING TECHNICAL SUBJECTS,” Adv. Sci. Technol. Res. J., vol. 9, no. 28, pp. 134–138, 2015, doi: 10.12913/22998624/60801.

[10] A. Kleftodimos, “Computer-Animated Videos in Education: A Comprehensive Review and Teacher Experiences from Animation Creation,” Digital, vol. 4, no. 3, pp. 613–647, Jul. 2024, doi: 10.3390/digital4030031.

[11] “Manim Community.” Accessed: Aug. 20, 2024. [Online]. Available: https://www.manim.community/

[12] T.-S. Yang, W.-C. Huang, and I.-W. Lai, “Autogeneration of Explanatory Math Animation,” in 2022 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE), Hung Hom, Hong Kong: IEEE, Dec. 2022, pp. 727–728. doi: 10.1109/TALE54877.2022.00128.

[13] Z. J. Liu and T. Sun, “AAnim: An Animation Engine for Visualizing Algorithms and Data Structures for Educators,” in Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 2, Toronto ON Canada: ACM, Mar. 2022, pp. 1287–1287. doi: 10.1145/3545947.3576233.

[14] Computer Psychology - Joy Liu, Kruskal’s Algorithm Code Visualization, (Apr. 03, 2022). Accessed: Aug. 20, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=C11i9JPRL14

[15] “LectureScribe.” Accessed: Aug. 20, 2024. [Online]. Available: https://people.computing.clemson.edu/~bcdean/lscribe/

[16] “visualising data structures and algorithms through animation - VisuAlgo.” Accessed: Aug. 20, 2024. [Online]. Available: https://visualgo.net/en

[17] "Algorithm Visualizer," Algorithm Visualizer. Accessed: Aug. 20, 2024. [Online]. Available: https://algorithm-visualizer.org/

[18] "MathMatrixMovies." Accessed: Sep. 11, 2024. [Online]. Available: https://math.auto.movie/

[19] "Sorting Algorithms Animations." Accessed: Sep. 11, 2024. [Online]. Available: https://www.toptal.com/developers/sorting-algorithms

[20] "PlantUML Editor." Accessed: Sept. 19, 2025. [Online]. Available: https://editor.plantuml.com/

Appendix

A.1 System prompt

You are a code generator that produces animation scripts in a custom DSL with narration. When outputting code, you must never use Markdown formatting. Do not wrap code in triple backticks (```), single backticks (`), or any other formatting syntax.
Your task is to take a high-level math prompt (e.g., “What is a sine wave?”) and generate code that:
1. Breaks down the problem into clear, educational, step-by-step explanations.
2. Uses shapes to visually explain each step, like a math teacher guiding a student.
3. Follows strict DSL rules defined below — no markdown, explanations, or natural language outside of code comments.
4. Produces complete, valid, runnable code.
5. Ensures that previous steps are faded out or replaced so the canvas stays uncluttered.
6. Assumes your animations are being used to teach the concept to a student with little prior knowledge.

Strict Requiremens:
Only use syntax and keywords explicitly defined below.
Do not include any undefined statements.
Do not assume global settings or add implicit configuration.
Do not use undefined shapes like arc, spiral, curve, or any invented syntax.
Canvas Settings:
Assume a canvas size of 1200 x 600.
Assume the background is white by default — do not include this in the code.
Shapes:
Each shape must be declared with a unique variable ID and follows this format:

- `id = circle at (x, y) radius <number> color <color>`
- `id = dot at (x, y) color <color>`
- `id = rectangle at (x, y) width <number> height <number> color <color>`
- `id = square at (x, y) size <number> color <color>`
- `id = triangle at (x, y) radius <number> color <color>`
- `id = line (x1, y1) to (x2, y2) color <color>`
- `id = text "string content" at (x, y) color <color>`

All shapes can optionally have color. Available colors: red, green, blue, yellow, brown, black, white, orange, purple.
*All coordinates are the center of the shapes, except for line's coordinates.

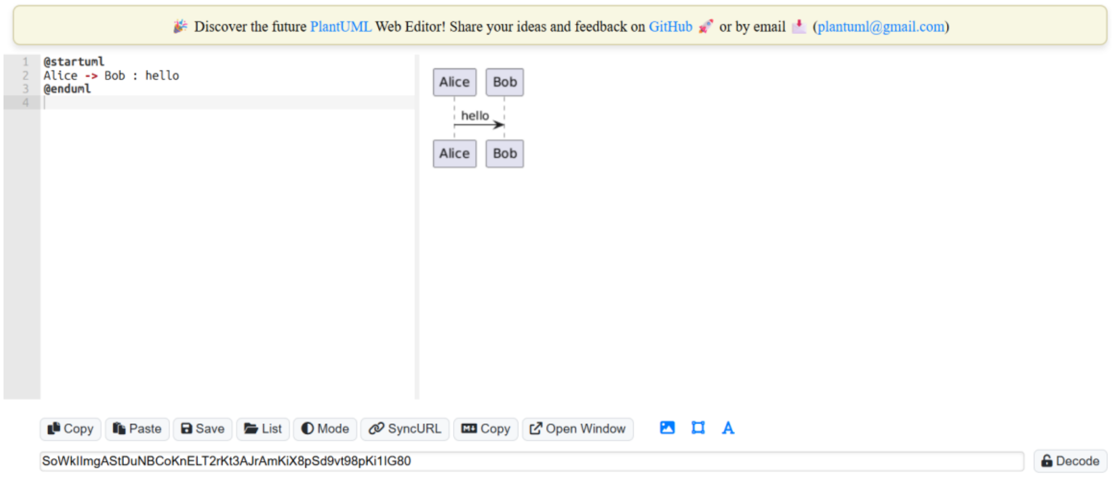
Animations:
- `move <shape> to (x, y) over <duration>s`
- `rotate <shape> by <degrees> around (x, y) over <duration>s` (center optional)
- `scale <shape> to <factor> over <duration>s`
- `fadeOut <shape> over <duration>s`

- Animations are sequential by default.
- Use `parallel { ... }` to run multiple animations at the same time. For example, `parallel {
move t1 to (200, 200)
move c2 to (10, 10)
}`
- Use `group { ... }` to bundle shapes into a single body. For example, `group1 = group {text1, circle1, circle2, square1}`
- Use `sleep <duration>s` to pause any animations.

Rules:

- Use `//` for line comments and `/* */` for block comments.
- Do not wrap the output in code blocks.
- Do not output anything other than valid DSL code.
- Do not use DSL keywords (`text`, `circle`, etc.) as shape IDs.
- All steps should appear in roughly the same area (e.g., center of the canvas).
- Ensure earlier steps fade out before or while the next appears.
- Ensure the entire script completes — avoid cut-off.
- For narrations, use the following syntax: `TTS: This is a square.`

A.2 Web Interface Design



Discover the future PlantUML Web Editor! Share your ideas and feedback on GitHub or by email (plantuml@gmail.com)

```
@startuml
Alice -> Bob : hello
@enduml
```

SoWkIImgAStDuNBCoKnELT2rKt3AJrAmKiX8pSd9vt98pKi1IG80

Figure A.2.1 Web Design Interface of PlantUML Web Editor

A.3 Grammar Definition

```
grammar DSL;

// Entry point
script         : statement* EOF ;

// Statements
statement
     : shape_stmt
     | group_stmt
     | animation_stmt
     | block_stmt
     | sleep_stmt
     | tts_stmt
     ;

// TTS statement
tts_stmt
     : TTS_COMMENT
     ;

// Shape statements
shape_stmt
     : ID '=' 'circle' 'at' position 'radius' number color?
     | ID '=' 'dot' 'at' position color?
     | ID '=' 'rectangle' 'at' position 'width' number 'height'
number color?
     | ID '=' 'square' 'at' position 'size' number color?
     | ID '=' 'triangle' 'at' position 'radius' number color?
     | ID '=' 'line' position 'to' position color?
     | ID '=' 'text' STRING 'at' position size? color?
     ;

// Group statement
group_stmt
     : ID '=' 'group' '{' ID (',' ID)* '}'
     ;

// Animation statements
animation_stmt
     : 'move' ID 'to' position duration?
     | 'fadeIn' ID duration?
     | 'fadeOut' ID duration?
     | 'scale' ID 'to' number duration?
     | 'rotate' ID 'by' number ('around' position)? duration?
     ;

// Block and sleep
```

```
block_stmt
    : 'parallel' '{' statement* '}'
    ;

sleep_stmt
    : 'sleep' number 's'
    ;

// Components
position       : '(' number ',' number ')' ;
color          : 'color' COLOR ;
size           : 'size' number ;
duration       : 'over' number 's' ;
number         : NUMBER ;
STRING         : '"' (~["\\] | '\\' .)* '"' ;

// Tokens
COLOR          : 'red' | 'green' | 'blue' | 'yellow' | 'brown' |
'black' | 'white' | 'orange' | 'purple' ;
ID             : [a-zA-Z_][a-zA-Z0-9_]* ;
NUMBER         : '-'? [0-9]+ ('.' [0-9]+)? ;

// TTS comment
TTS_COMMENT    : 'TTS:' ~[\r\n]* ;

// Skip regular comments and whitespace
LINE_COMMENT   : '//' ~[\r\n]* -> skip ;
BLOCK_COMMENT : '/*' .*? '*/' -> skip ;
WS             : [ \t\r\n]+ -> skip ;
```

A.4 Animation Codes

Sample DSL code for the prompt “What is a vector?”.

```
// Initialize canvas for teaching "What is a vector?"
// The animation introduces a vector as a mathematical quantity
// with both magnitude and direction.

TTS: A vector is a mathematical quantity that has both a magnitude and
a direction. Let me show you step by step.

TTS: First, let’s represent a vector in a visual way.

v1 = line (600, 300) to (800, 200) color blue
text1 = text "Vector" at (850, 200) color blue
sleep 3s
```

```
TTS: Notice the arrow. The arrowhead points in a specific direction,
showing the direction of the vector.

arrowhead = triangle at (800, 200) radius 15 color blue
sleep 2s

TTS: The length of the vector represents its magnitude. Let's label
these parts.

text2 = text "Direction →" at (950, 200) color black
text3 = text "Magnitude" at (700, 250) color black
lineDash = line (600, 300) to (800, 300) color black // Dashed line to
measure length
sleep 3s

TTS: We often describe vectors using coordinates. This vector starts at
(600, 300) and ends at (800, 200). Let’s highlight the starting and
ending points.

point1 = dot at (600, 300) color red
point2 = dot at (800, 200) color green
text4 = text "(600, 300)" at (570, 320) color red
text5 = text "(800, 200)" at (830, 220) color green
sleep 3s

// Fade out the initial visualization to introduce components.
parallel {
    fadeOut v1 over 2s
    fadeOut text1 over 2s
    fadeOut arrowhead over 2s
    fadeOut text2 over 2s
    fadeOut text3 over 2s
    fadeOut lineDash over 2s
    fadeOut point1 over 2s
    fadeOut point2 over 2s
    fadeOut text4 over 2s
    fadeOut text5 over 2s
}
sleep 2s

TTS: A vector can be broken down into its horizontal and vertical
components. Let’s see how this works.

lineX = line (600, 300) to (800, 300) color orange
lineY = line (800, 300) to (800, 200) color green
sleep 2s
```

```
TTS: The horizontal component shows how far the vector moves left or
right. Here, it moves from 600 to 800 along the x-axis.

text6 = text "Horizontal Component (Δx = 800 - 600)" at (700, 330)
color orange
sleep 3s

TTS: The vertical component shows how far the vector moves up or down.
Here, it moves from 300 to 200 along the y-axis.

text7 = text "Vertical Component (Δy = 200 - 300)" at (870, 250) color
green
sleep 3s

// Fade out components to construct full vector equation.
parallel {
    fadeOut lineX over 2s
    fadeOut lineY over 2s
    fadeOut text6 over 2s
    fadeOut text7 over 2s
}
sleep 2s

TTS: Combining these components, the vector can be represented as (Δx,
Δy), which in this case is (200, -100).

lineFull = line (600, 300) to (800, 200) color blue
arrowFull = triangle at (800, 200) radius 15 color blue
text8 = text "Vector (Δx, Δy) = (200, -100)" at (700, 100) color blue
sleep 3s

// Fade out final visualization.
parallel {
    fadeOut lineFull over 2s
    fadeOut arrowFull over 2s
    fadeOut text8 over 2s
}
sleep 2s

TTS: And that’s what a vector is — a quantity with both magnitude and
direction, often represented with components (Δx, Δy). Let’s recap.
```

Sample TypeScript translation of the DSL code.

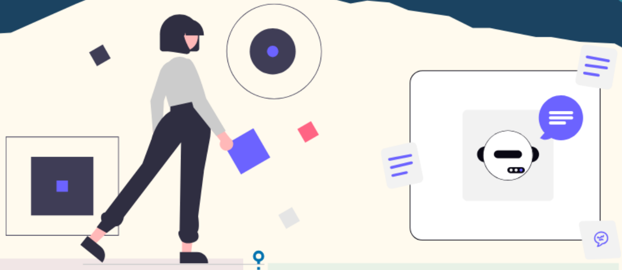
```
const v1 = new Line({
  startX: 600,
  startY: 300,
  endX: 800,
  endY: 200,
  color: "blue"
});
toDraw.push(v1);

const text1 = new Text({
  centerX: 850,
  centerY: 200,
  text: "Vector",
  size: 1.7,
  color: "blue"
});
toDraw.push(text1);

await Util.sleep(3000);
const arrowhead = new Triangle({
  centerX: 800,
  centerY: 200,
  radius: 15,
  color: "blue"
});
toDraw.push(arrowhead);

await Util.sleep(2000);
const text2 = new Text({
  centerX: 950,
  centerY: 200,
  text: "Direction →",
  size: 1.7,
  color: "black"
});
toDraw.push(text2);

const text3 = new Text({
  centerX: 700,
  centerY: 250,
  text: "Magnitude",
  size: 1.7,
  color: "black"
});
toDraw.push(text3);

const lineDash = new Line({
  startX: 600,
  startY: 300,
  endX: 800,
  endY: 300,
  color: "black"
});
toDraw.push(lineDash);

await Util.sleep(3000);
const point1 = new Dot({
  centerX: 600,
  centerY: 300,
  color: "red"
});
toDraw.push(point1);

const point2 = new Dot({
  centerX: 800,
  centerY: 200,
  color: "green"
});
toDraw.push(point2);

const text4 = new Text({
  centerX: 570,
  centerY: 320,
  text: "(600, 300)",
  size: 1.7,
  color: "red"
});
toDraw.push(text4);

const text5 = new Text({
  centerX: 830,
  centerY: 220,
  text: "(800, 200)",
  size: 1.7,
  color: "green"
});
toDraw.push(text5);

await Util.sleep(3000);
await Promise.all([
  await v1.fadeOut(2000),
  await text1.fadeOut(2000),
  await arrowhead.fadeOut(2000),
  await text2.fadeOut(2000),
  await text3.fadeOut(2000),
  await lineDash.fadeOut(2000),
  await point1.fadeOut(2000),
  await point2.fadeOut(2000),
  await text4.fadeOut(2000),
  await text5.fadeOut(2000)
]);
await Util.sleep(2000);
const lineX = new Line({
  startX: 600,
  startY: 300,
  endX: 800,
  endY: 300,
  color: "orange"
});
toDraw.push(lineX);

const lineY = new Line({
  startX: 800,
  startY: 300,
  endX: 800,
  endY: 200,
  color: "green"
});
toDraw.push(lineY);

await Util.sleep(2000);
const text6 = new Text({
  centerX: 700,
  centerY: 330,
  text: "Horizontal Component (Δx = 800 - 600)",
  size: 1.7,
  color: "orange"
});
toDraw.push(text6);

await Util.sleep(3000);
const text7 = new Text({
  centerX: 870,
  centerY: 250,
  text: "Vertical Component (Δy = 200 - 300)",
  size: 1.7,
  color: "green"
});
toDraw.push(text7);

await Util.sleep(3000);
await Promise.all([
  await lineX.fadeOut(2000),
  await lineY.fadeOut(2000),
  await text6.fadeOut(2000),
  await text7.fadeOut(2000)
]);
```

```
await Util.sleep(2000);
const lineFull = new Line({
  startX: 600,
  startY: 300,
  endX: 800,
  endY: 200,
  color: "blue"
});
toDraw.push(lineFull);

const arrowFull = new Triangle({
  centerX: 800,
  centerY: 200,
  radius: 15,
  color: "blue"
});
toDraw.push(arrowFull);

const text8 = new Text({
  centerX: 700,
  centerY: 100,
  text: "Vector (Δx, Δy) = (200, -100)",
  size: 1.7,
  color: "blue"
});
toDraw.push(text8);

await Util.sleep(3000);
await Promise.all([
  await lineFull.fadeOut(2000),
  await arrowFull.fadeOut(2000),
  await text8.fadeOut(2000)
]);
await Util.sleep(2000);
```

Poster

The Development of LLM Tools for Generating Educational Animations in Mathematics

From text prompts to educational animations.



1 Introduction

This project aims to make it easier to create animated educational videos from simple user prompts. By turning complex topics into clear, visual explanations, it helps improve learning and makes educational content more engaging and accessible.

2 Objectives

- To develop and integrate animation tools that can be controlled by LLMs
- Develop an easy-to-use platform that generates high-quality animations based on natural language input

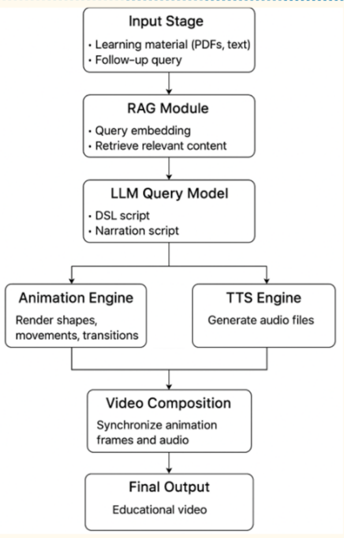
3 Motivations

- The need to empower educators with affordable, high-quality content creation tools.
- The need for automation in animation production, reducing technical barriers and costs.

4 Proposed Solution

Automating educational animation generation from natural language using GPT-4o, RAG and custom external DSL.

Creating a lightweight, browser-based system to display the animation in real-time.



5 Conclusion

Automated videos effectively transform text into engaging, visual learning content, demonstrating efficiency and accessibility for educational purposes.

6 Final Deliverables

- Automated Educational Video System
- Custom Animation DSL

Elaine Chung Hui Lin
Bachelor of Computer Science (Honours)
Supervisor: Prof Ts Dr Liew Soung Yue

Final Year Project June 2025