ALGO-Visualizer: Project Report

Generating Dynamic Algorithm Animations from Pseudocode using a DSPy-Inspired LLM Architecture

Course: Design Credit Course

Date: October 10, 2025

Professor: Mrs Pallavi Jain

Abstract

The "ALGO-Visualizer" project addresses a fundamental challenge in computer science education: the difficulty of understanding complex algorithms from static text. This report details the journey of creating a web-based tool that automatically transforms user-provided pseudocode for common sorting algorithms into dynamic, educational video animations. The project leverages a technology stack of Python, Streamlit, and the Manim animation library. The core intelligence is a Large Language Model (LLM) accessed via the OpenRouter API. This report documents the project's evolution, starting from a brittle, error-prone architecture that relied on specific prompts for each algorithm, which resulted in numerous rendering failures and visual glitches. The final, successful architecture is inspired by the DSPy framework, utilizing a single, robust "master prompt." This master prompt intelligently guides the LLM to generate correct and visually appealing Manim code for multiple algorithms from a single set of principles, resulting in a scalable and impressive educational tool.

1. Introduction

1.1. Problem Statement

Algorithms are the bedrock of computer science, but their abstract, step-by-step nature can be difficult for learners to grasp from textbooks and static diagrams. Visualizing the movement of data—the comparisons, swaps, and recursive calls—is crucial for building a deep, intuitive understanding. While many pre-made algorithm animations exist, there is a disconnect between the abstract pseudocode a student studies and the concrete visualization they watch.

1.2. Project Idea & Objectives

The central idea of ALGO-Visualizer is to bridge this gap. The project aimed to create a tool where a user could input the very pseudocode they are studying (either as text or an image) and a custom array of numbers, and in return, receive a custom-generated video that visually represents that algorithm's logic in action on their data.

The primary objectives were:

- To develop a simple, visually appealing, and intuitive web interface for user interaction.
- To accept user-provided pseudocode for multile algorithms like Bubble Sort, Merge Sort, and Quick Sort.
- To leverage an LLM to intelligently interpret the user's input and generate corresponding animation code.
- To use the Manim animation library to render this code into a high-quality, easy-to-understand video with correct logic and clean visuals.
- To evolve the system's architecture to be scalable, robust, and capable of handling variations in user input without failing.

2. Technology Stack

The project was built using a modern, Python-based technology stack, chosen for its power and rapid development capabilities.

- **Python:** The core programming language for the entire project.
- **Streamlit:** A Python library used to build the interactive web application frontend. It was chosen for its simplicity, speed, and ability to create a polished UI with custom styling.
- **Manim (Community Edition):** A powerful, open-source animation engine for creating precise, programmatic animations. It is the engine that renders the final videos.
- **Large Language Models (LLMs):** The intelligent core of the project. We utilized powerful models like Anthropic's Claude 3 Haiku, accessed via the **OpenRouter API**, to handle the complex task of converting abstract pseudocode into concrete Python/Manim code.
- **DSPy (Inspiration):** While the DSPy library itself was not used, its core philosophy was the breakthrough that made this project successful. DSPy champions the idea of programming with LLMs by using structured, constrained prompts (called "signatures") that teach the model *how to think*, rather than giving it brittle, step-by-step instructions. This "master prompt" approach was the key to making our project reliable.

3. Project Evolution: A Journey Through Failure to a Robust Solution

The development of ALGO-Visualizer was an iterative and challenging process that involved a significant architectural pivot after numerous failures.

3.1. The First Approach: Brittle, Algorithm-Specific Templates

The initial approach attempted to use a separate, hand-crafted Manim template for each algorithm.

- **Workflow:** The application would detect the algorithm from the user's pseudocode and then select a pre-written Python script (e.g., bubble_sort_template.py). It would then inject the user's array into this template and render it.
- **Challenges & Failures:** This method was a complete failure for anything beyond a single, simple algorithm.
    - **The Comparison Mode Disaster:** When attempting to create a "Comparison Mode" to run two animations side-by-side, this approach failed catastrophically. The animations would render in the center of the screen, overlapping each other and creating a confusing, unusable mess. The coordinate systems of the two templates conflicted, a problem that proved incredibly difficult to solve without a complete redesign.
    - **Lack of Scalability:** Adding a new algorithm meant writing an entire, complex Manim script from scratch.
    - **Zero Intelligence:** This approach did not use the LLM to *generate* the animation, defeating a core premise of the project. It was simply a script-runner.

3.2. Final Approach: The DSPy-Inspired Master Prompt

The complete failure of the template-based comparison mode forced a return to the project's original vision: having the LLM generate the code. To overcome the earlier issues of buggy code and rendering hangs, the architecture was redesigned around a single, universal "master prompt," inspired by the principles of the DSPy framework.

- **The Shift in Philosophy:** Instead of telling the LLM *exactly* how to animate Bubble Sort step-by-step, the new master prompt teaches the LLM the **general principles of creating a good algorithm animation**. It constrains the model's thinking process, leading to more reliable outputs.
- **Anatomy of the Master Prompt:** The final prompt is a highly structured template that instructs the LLM to:
    1. **Assume a Persona:** "You are an expert Manim programmer."
    2. **Identify the Algorithm:** Analyze the pseudocode and name the algorithm.
    3. **Use User Data:** Critically, it instructs the LLM to use the exact array of numbers provided by the user.

4. **Adhere to General Animation Principles:** It provides a set of universal rules for good visualizations: use VGroup of Square and Text for elements, use YELLOW for comparisons, animate swaps cleanly by moving Text objects, use vertical shifts for recursion to avoid overlap, and indicate completion with GREEN.
5. **Enforce a Strict Output Format:** This was the most critical change. The prompt demands that the LLM's entire output be a single, clean JSON object with two keys: "algorithm" and "manim_code". This makes the response programmatically predictable and eliminates parsing errors from conversational text.

- **Benefits of this Final Approach:**
    - **Generalization & Scalability:** The single prompt successfully generates high-quality animations for all supported algorithms. Adding a new algorithm (like Insertion Sort) is now as simple as adding its name to the list of allowed algorithms in the prompt.
    - **Robustness:** By enforcing a strict JSON output and providing clear guiding principles, the LLM is far less likely to generate buggy code that hangs the renderer.
    - **Maintainability:** The entire "brain" of the application is now consolidated into a single, well-defined prompt that is easy to read, debug, and refine.

4. Final Results

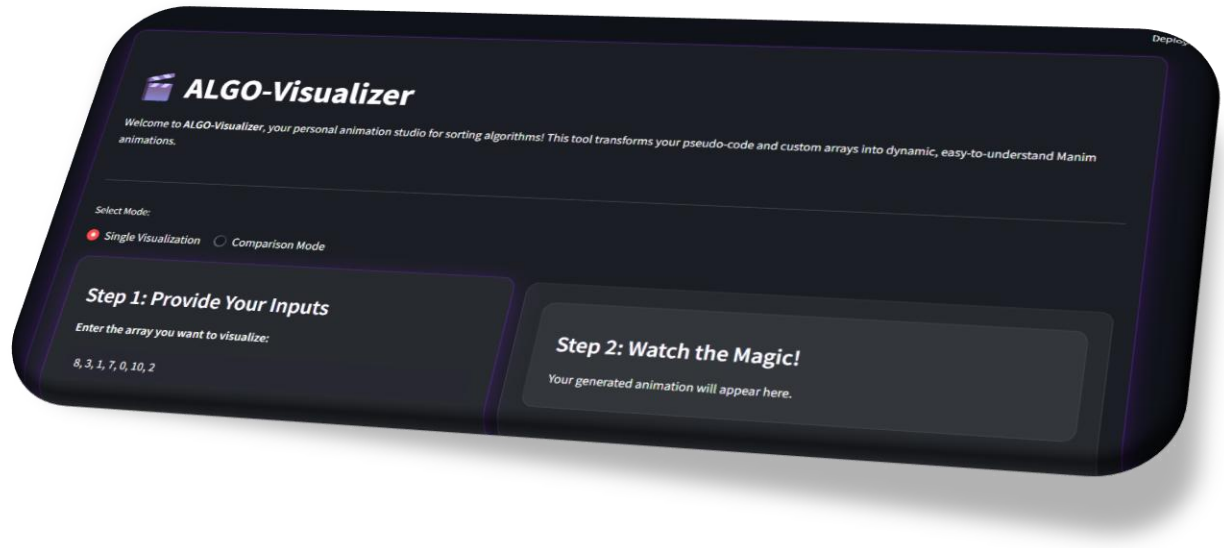
While the project is still under active development, the successful implementation of the DSPy-inspired architecture marks a major milestone and places it on a clear path toward completion. The current version of ALGO-Visualizer has overcome the significant roadblocks that plagued its early development.

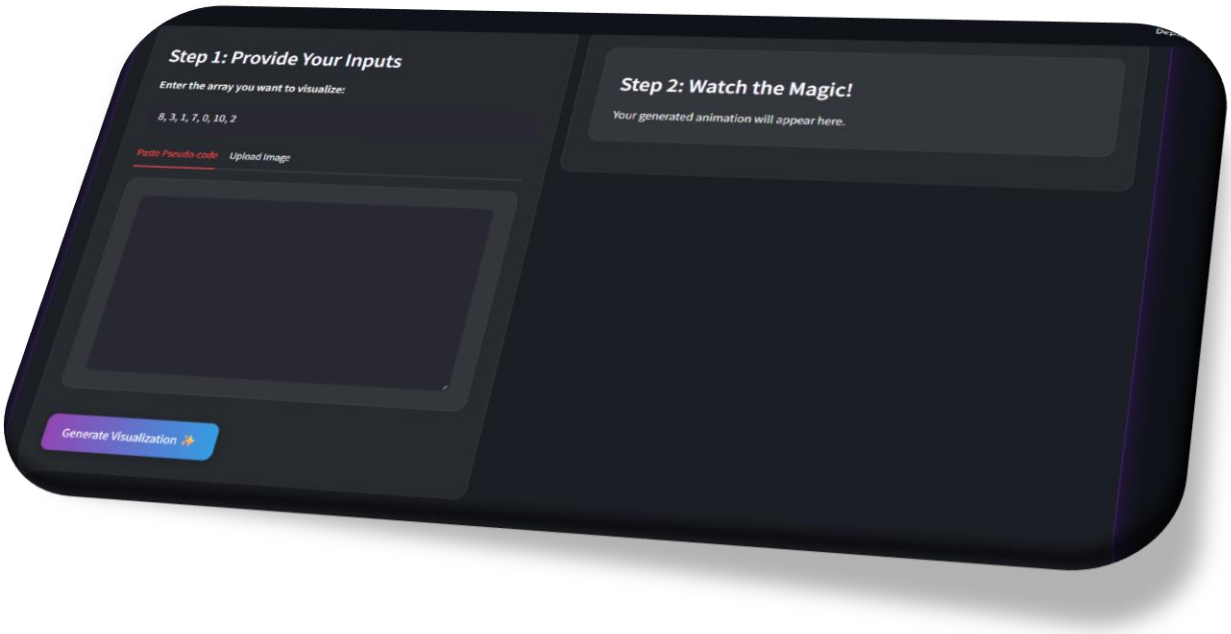
The previous approach, which relied on brittle, algorithm-specific prompts or templates, consistently failed. It produced Manim code with subtle but critical bugs, such as infinite loops or flawed logic, which caused the video renderer to hang indefinitely. This manifested as a frustrating "buffering" state with no clear error, making debugging nearly impossible. Furthermore, this old method completely broke down when trying to implement a "Comparison Mode," as the separate animation scripts could not coexist, resulting in a visual mess of overlapping animations.

The new architecture has definitively solved these problems. The application now provides a clean, professional, and interactive web interface that successfully generates high-quality, correct, and visually appealing animations for multiple algorithms from a single, intelligent prompt. The shift to the DSPy-inspired master prompt was the key that unlocked a reliable

and scalable solution, proving that this is the correct path forward. The project is now in a strong position to be finalized and expanded with additional features.

UI :





5. Acknowledgements

Contributors:

- Tushar Kant
- Akshay Bachuu

Special Thanks: A special thanks to our mentor Anshul Thakur, whose guidance and feedback were invaluable throughout this challenging project. Your suggestion to move towards a more generalized, prompt-based architecture was the critical insight that elevated this project from a collection of simple scripts to a truly intelligent and scalable system. Thank you for your support and for pushing us to find the right solution through an iterative process of failure and success.

----END----