Qualitative Analysis of LLM-generated and Human-written Code

Presented by:-

Avisha Goyal (21103057)
Abiha Naqvi (21103073)
Apeksha Jain (21103081)

Problem Statement

- Comparative study of AI-generated and human-written code.
- Evaluation of code quality, efficiency, and complexity.
- Development of tools to analyze and visualize results.

Objective

- Evaluate the effectiveness of Large Language Models (LLMs) in generating code.
- Compare LLM-generated code with human-written code in C++ and Python.
- Analyze on parameters like cyclomatic complexity, lines of code, time complexity, and space complexity.

Research Questions

- **RQ1** - How do correctness, structure, and performance differ between human-generated and LLM-generated code?
- **RQ2** - How does the specificity and quality of prompts influence the accuracy, structure, and relevance of LLM-generated code?
- **RQ3** - What factors drive user preference for human-generated versus LLM-generated code?

Phases of The Project

01 Research and Defining Project Objectives

02 Data Collection and Dataset Creation

03 Analyzing and Comparing LLM and Human Codes

04 Developing a Dashboard for Automated Analysis

Stages of The Project

Phase 1: Research and Defining Objectives

- **Extensive Research**: Conducted a thorough review of existing literature to understand the current landscape of AI-generated versus human-written code.
- **Identify Gaps:** Identified gaps in the current understanding of AI-generated code quality and efficiency.
- **Finalise Metrics After Reading Research Papers**: Finalized specific metrics and aspects to analyze, such as cyclomatic complexity, lines of code, time complexity, and space complexity.

Stages of The Project

Phase 2: Data Collection and Dataset Creation

- **Gather Human-Written Code**: Collected human-written code samples in C++ and Python.
- **Generate AI Code**: Used LLMs like ChatGPT 3.5 and Gemini 1.5 to generate code for similar problems or tasks.
- **Build a Diverse Dataset:** Created a comprehensive dataset with both human-written and AI-generated code samples covering various types of problems. Worked with almost 400 code samples.

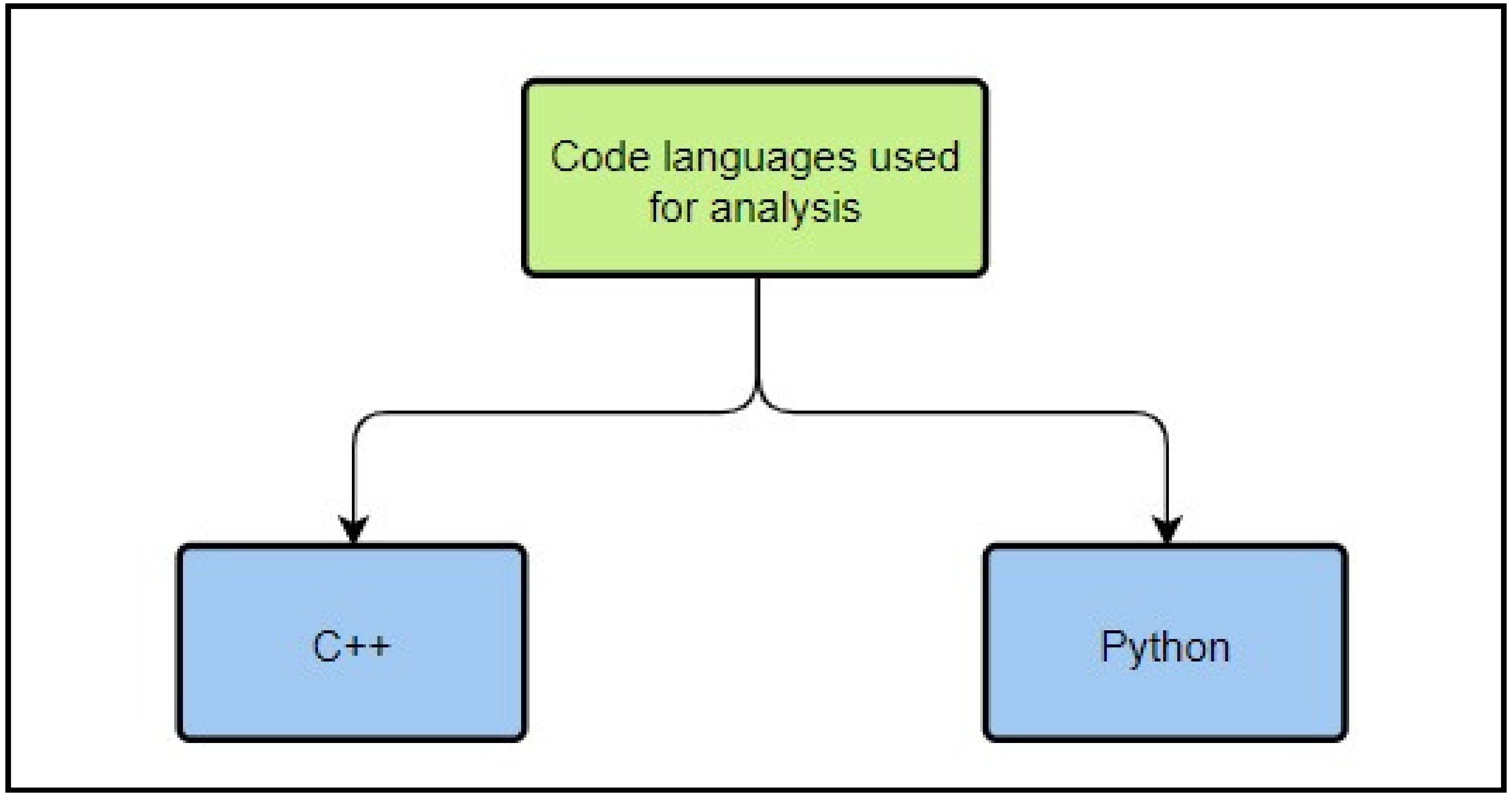


Fig 1. Coding languages selected for analysis

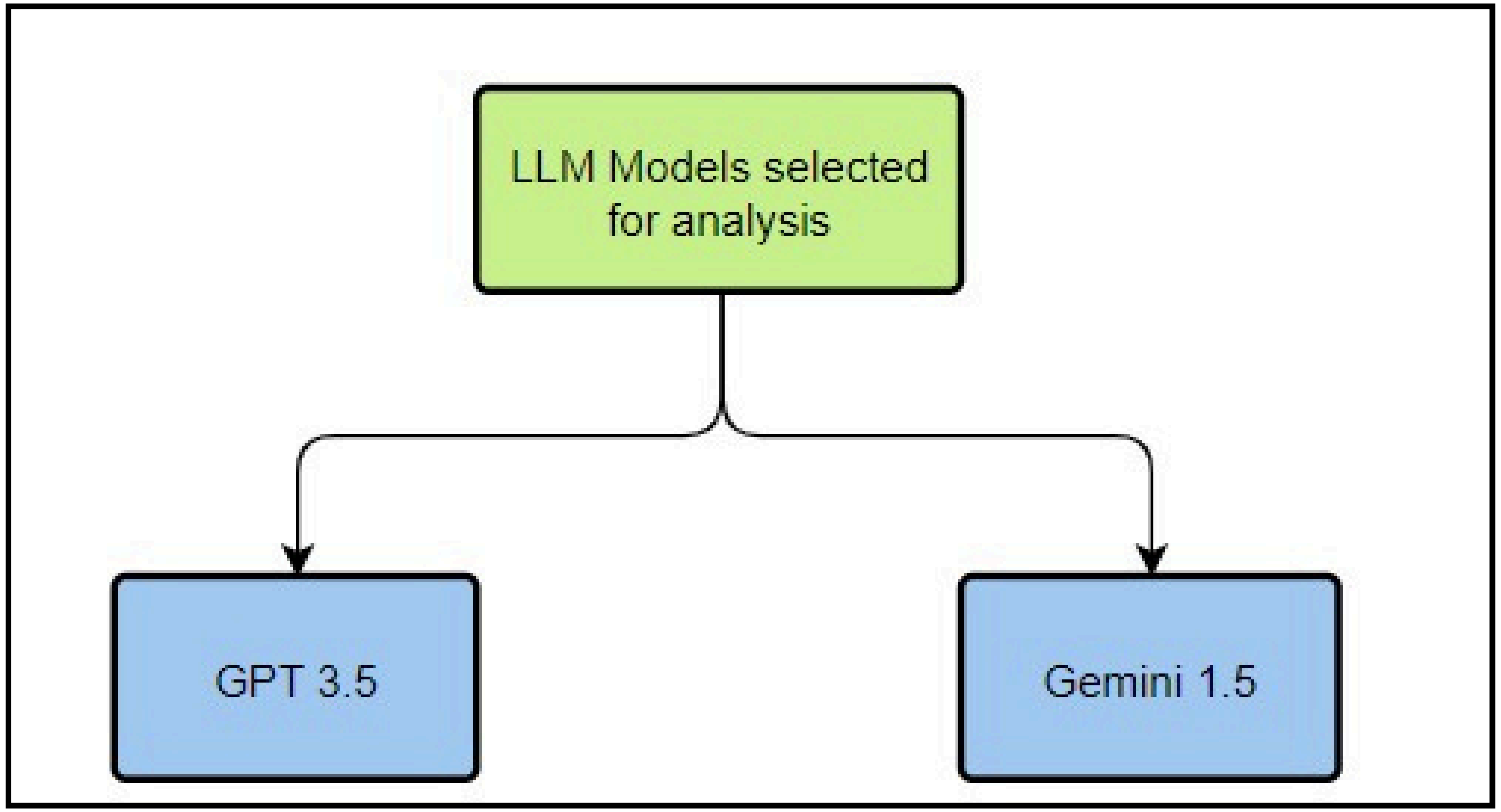


Fig 2. LLM models selected for analysis

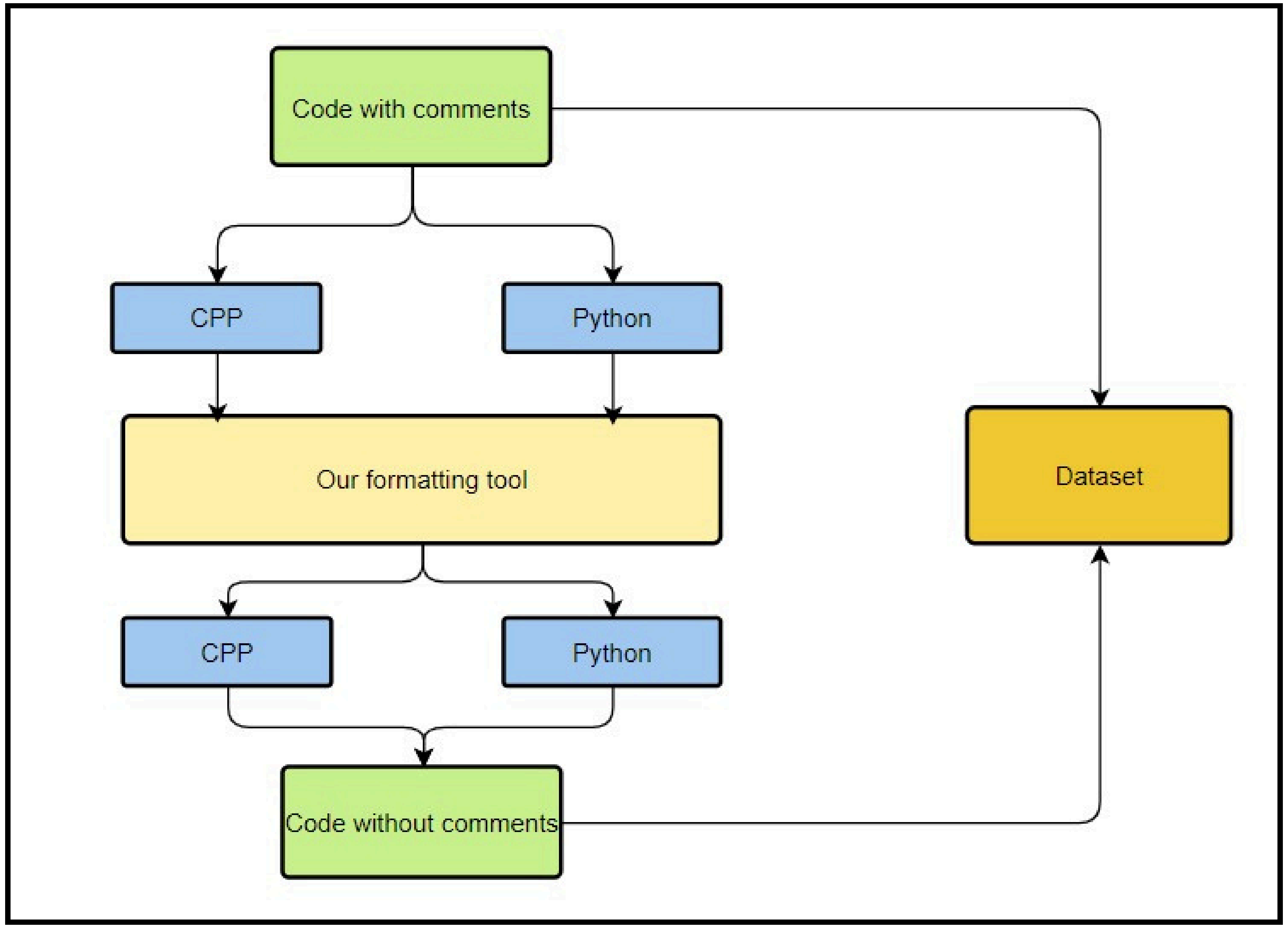


Fig 3. Flow of formatting of code snippets

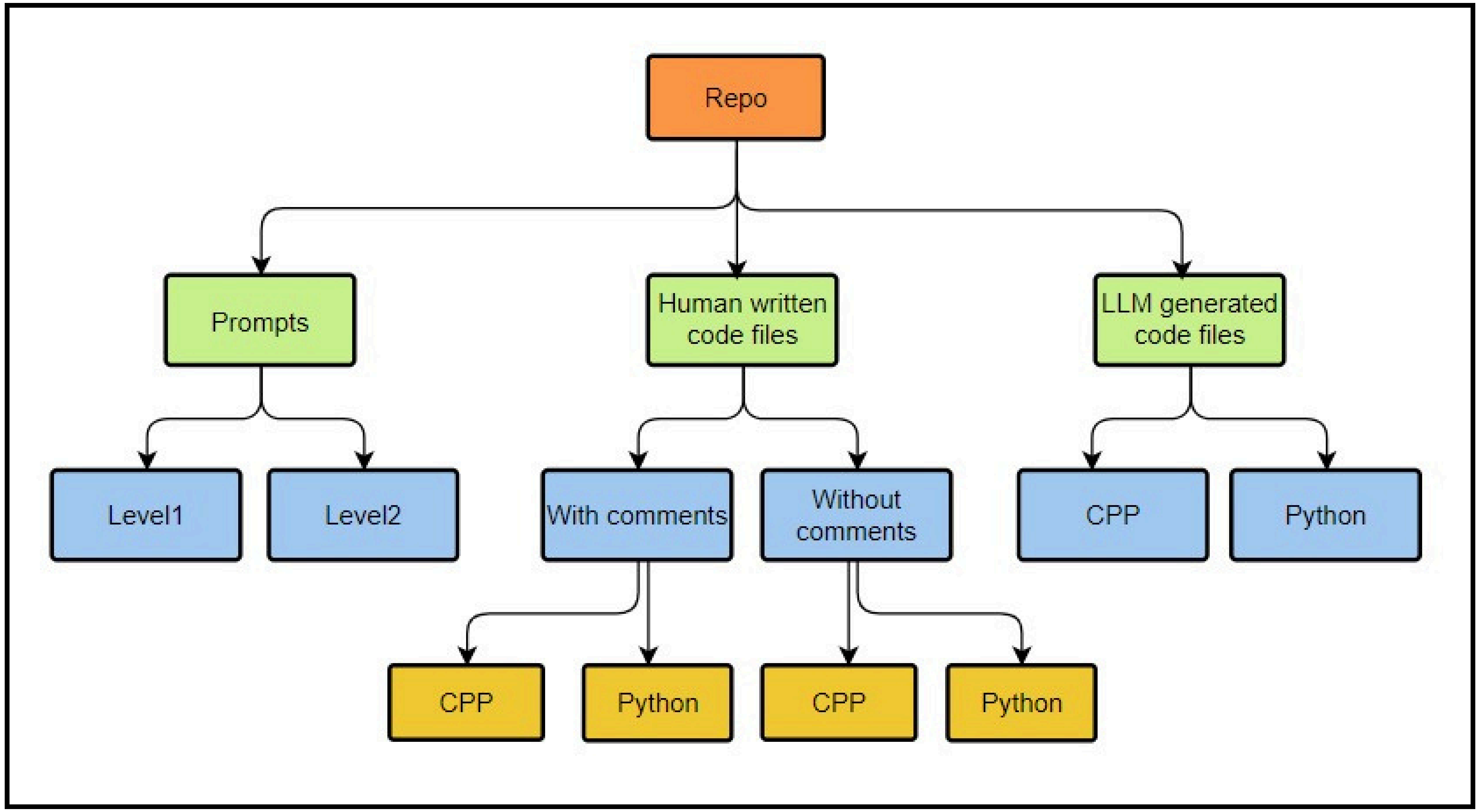


Fig 4. Dataset repo structure

Dataset Overview

- Balanced set of 40 programming questions from LeetCode and Codeforces.
- Equal distribution of simple and complex challenges.
- Data organized into four categories: ChatGPT - C++, ChatGPT - Python, Gemini - C++, Gemini - Python.
- Approximately 400 code samples across different levels of complexity.
- Metrics recorded for each sample: cyclomatic complexity, lines of code, time complexity, and space complexity.

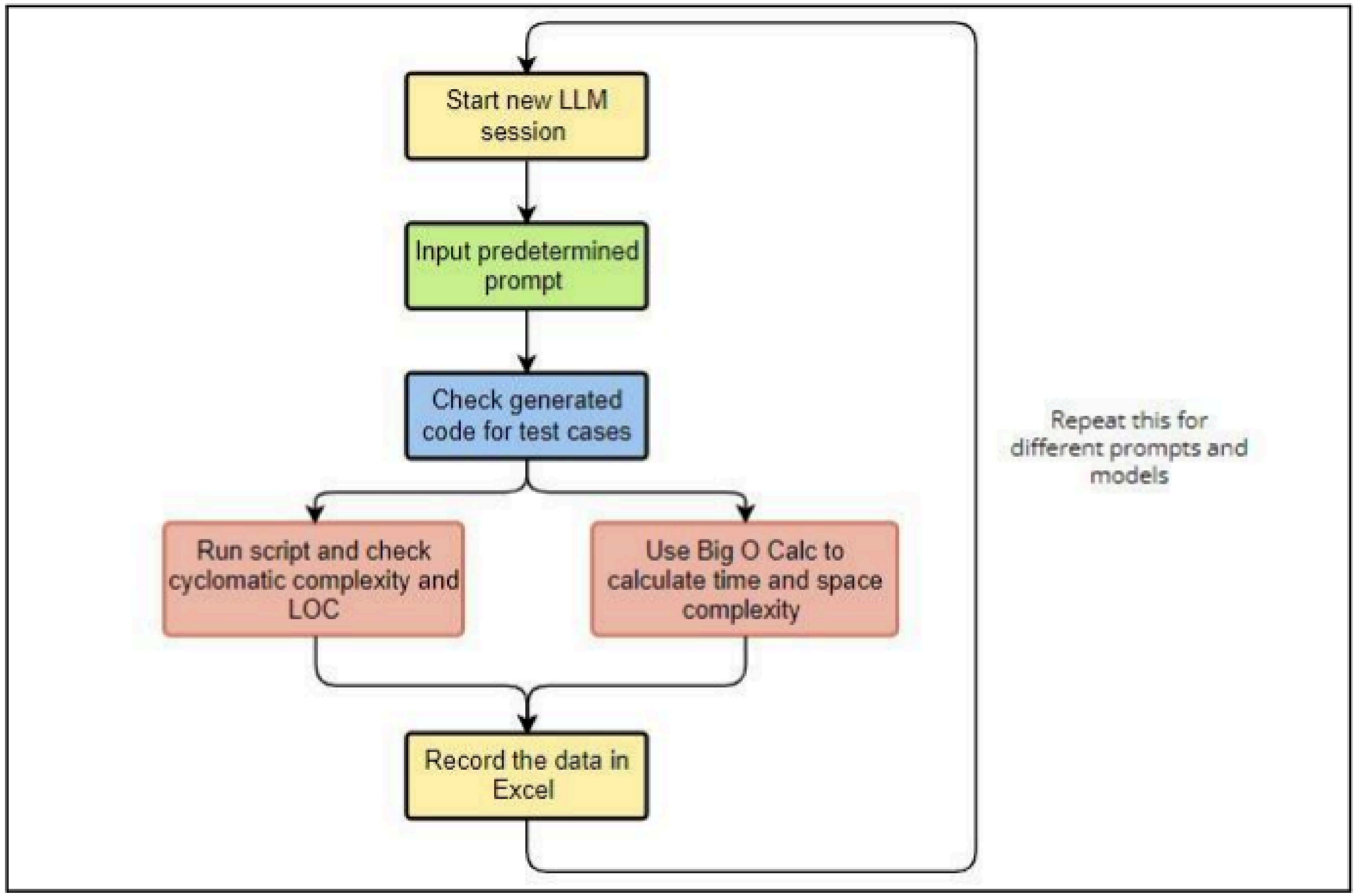

Start new LLM session
Input predetermined prompt
Check generated code for test cases
Run script and check cyclomatic complexity and LOC
Use Big O Calc to calculate time and space complexity
Record the data in Excel
Repeat this for different prompts and models

Fig 4. Experiment Procedure flow

Custom Scripts

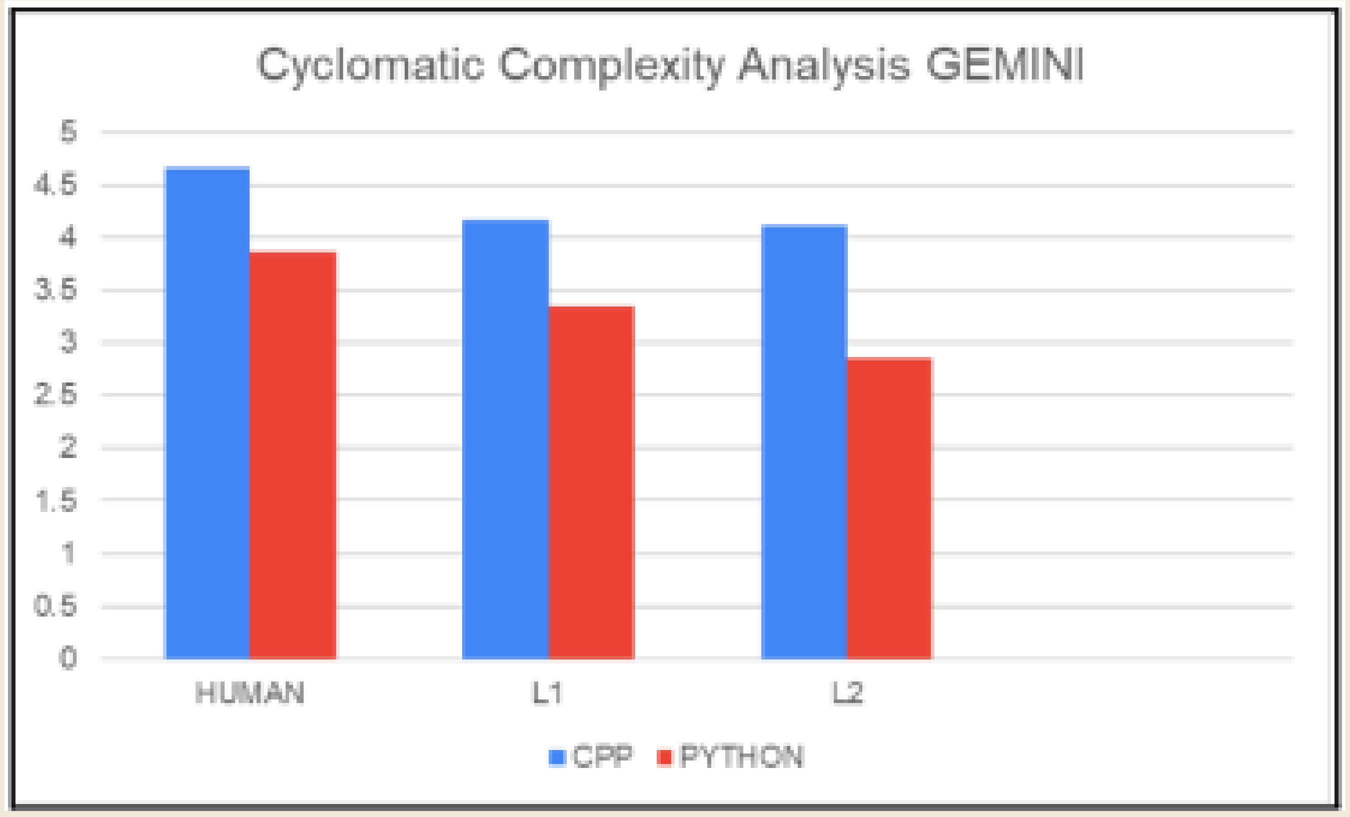
- **C++ Script:** Calculates Lines of Code (LOC) and Cyclomatic Complexity using regex to identify decision points like if, for, and case.
- **Python Script:** Evaluates cyclomatic complexity with flake8 McCabe plugin, providing detailed metrics for each function.
- **BigO Calculator:** Used for analyzing time and space complexity, offering insights into code scalability and efficiency.

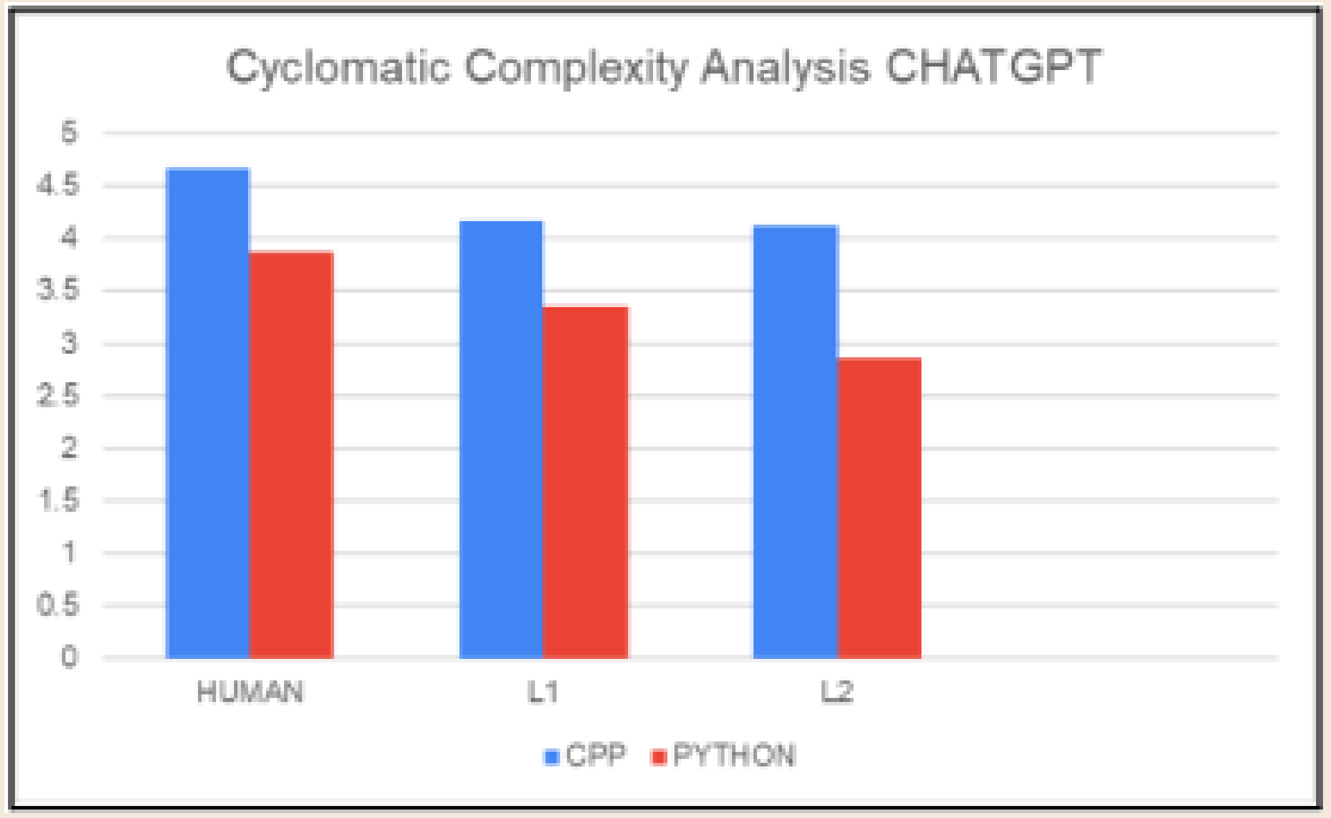
Phases of The Project

Phase 3: Analyzing and Comparing Codes

- **Detailed Analysis:** Performed a detailed analysis of the collected codes based on predefined software engineering metrics.
- **Comparison:** Compared human-written and AI-generated codes to draw meaningful conclusions about their strengths and weaknesses.
- **Insights:** Generated insights that will form the basis of our findings.

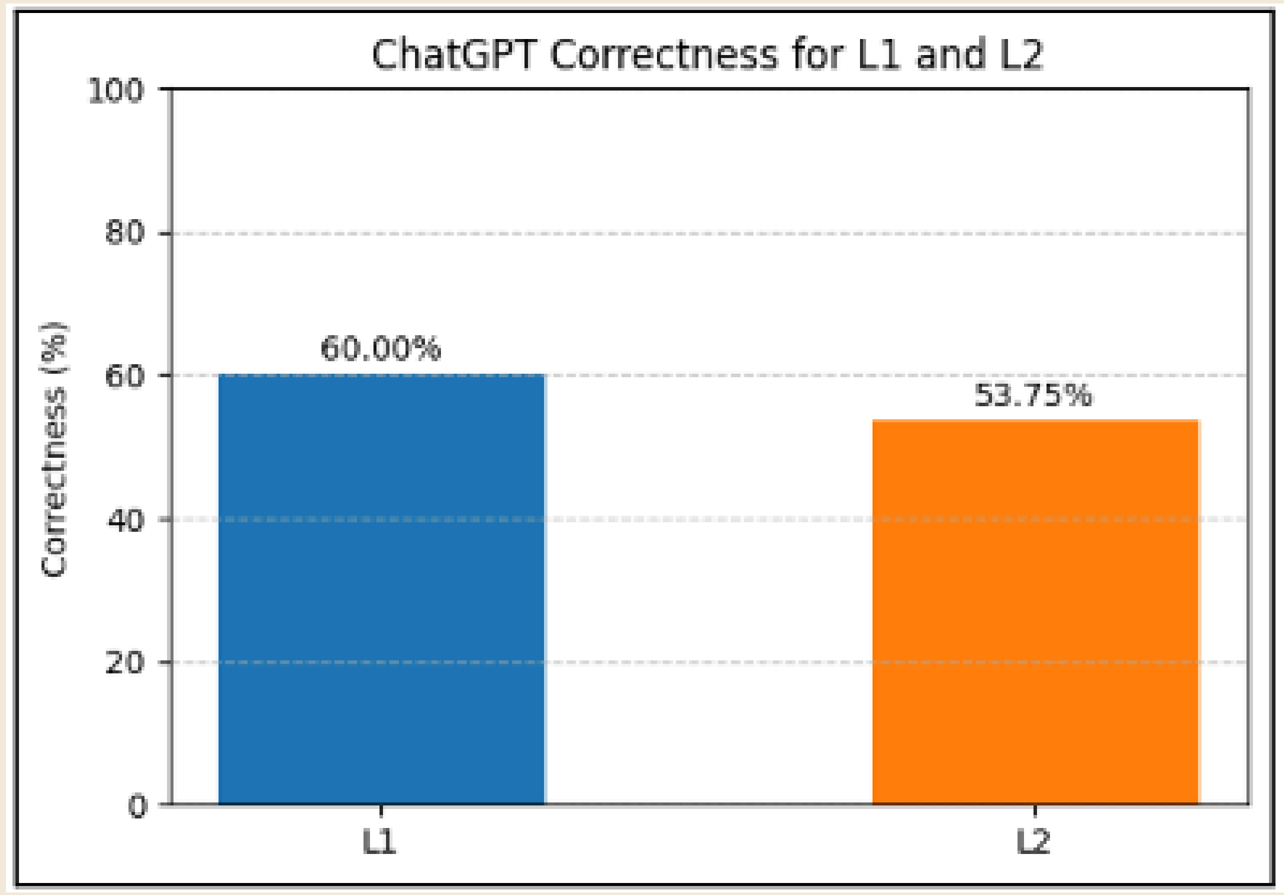
Some Insights

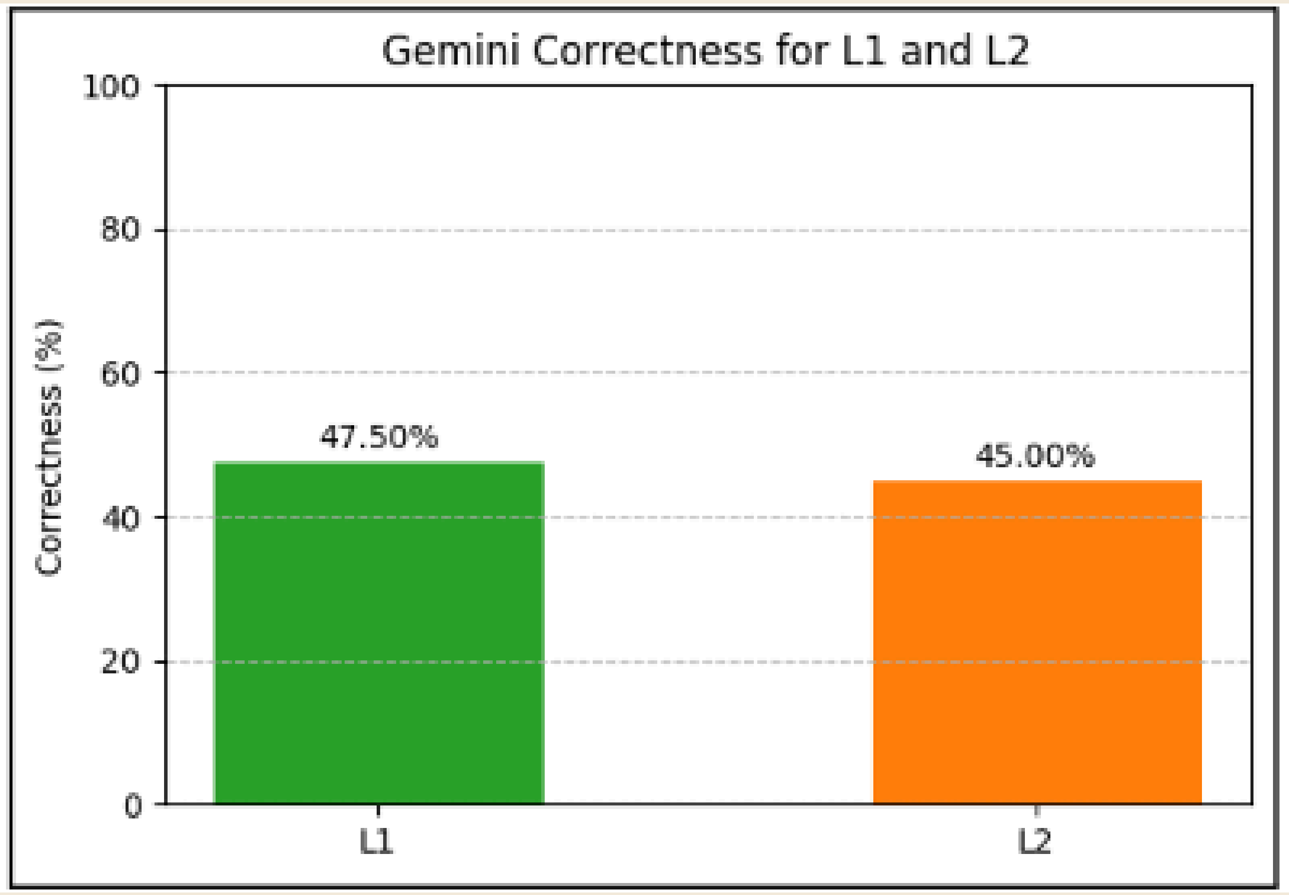




Finding: *The cyclomatic complexity of human - written code is generally more than LLM generated code for different models and prompt levels.*

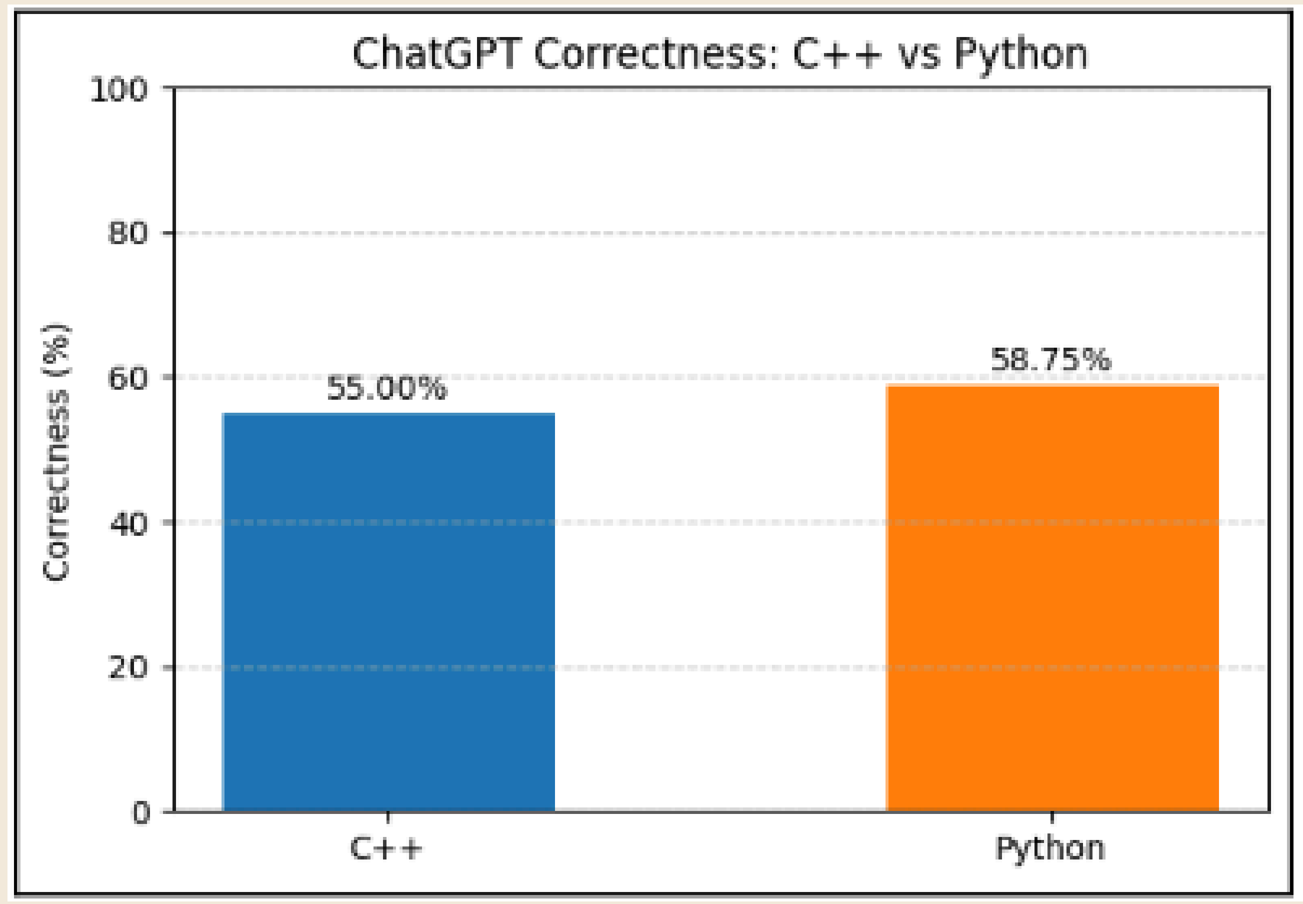
Some Insights

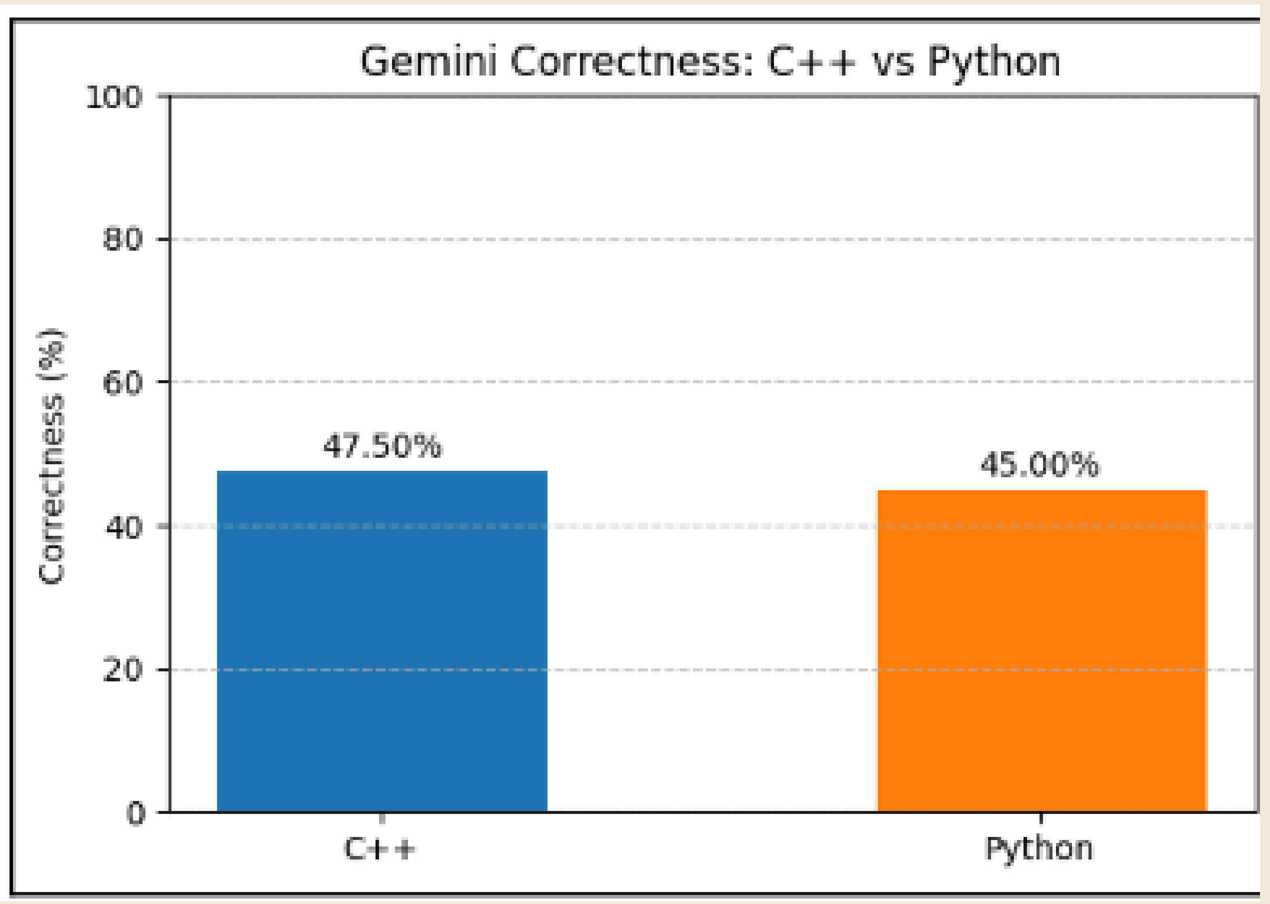




Finding: *Correctness decreases from L1 to L2 this shows LLM models work better with simple to the point prompts instead of complex prompts.*

Some Insights





Finding: *Through experiments found ChatGPT 3.5 works better for Python and Gemini works slightly better for CPP.*

Stages of The Project

Phase 4: Developing a Dashboard for Automated Analysis

- **Interactive Dashboard**: Develop a dashboard to automate the comparison and analysis process.
- **Visualize Differences**: Provide an interactive interface to visualize the differences and results across various metrics.
- **User-Friendly**: Make it easy to understand the outcomes of our study and explore the data in detail.

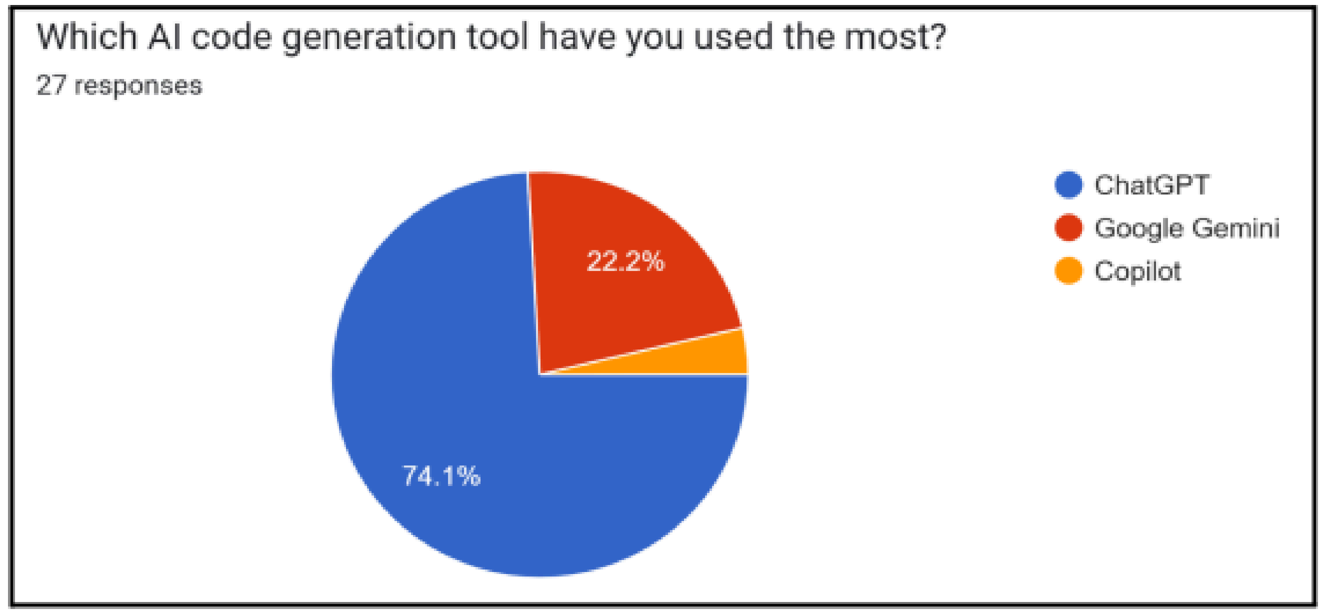
ToolChain

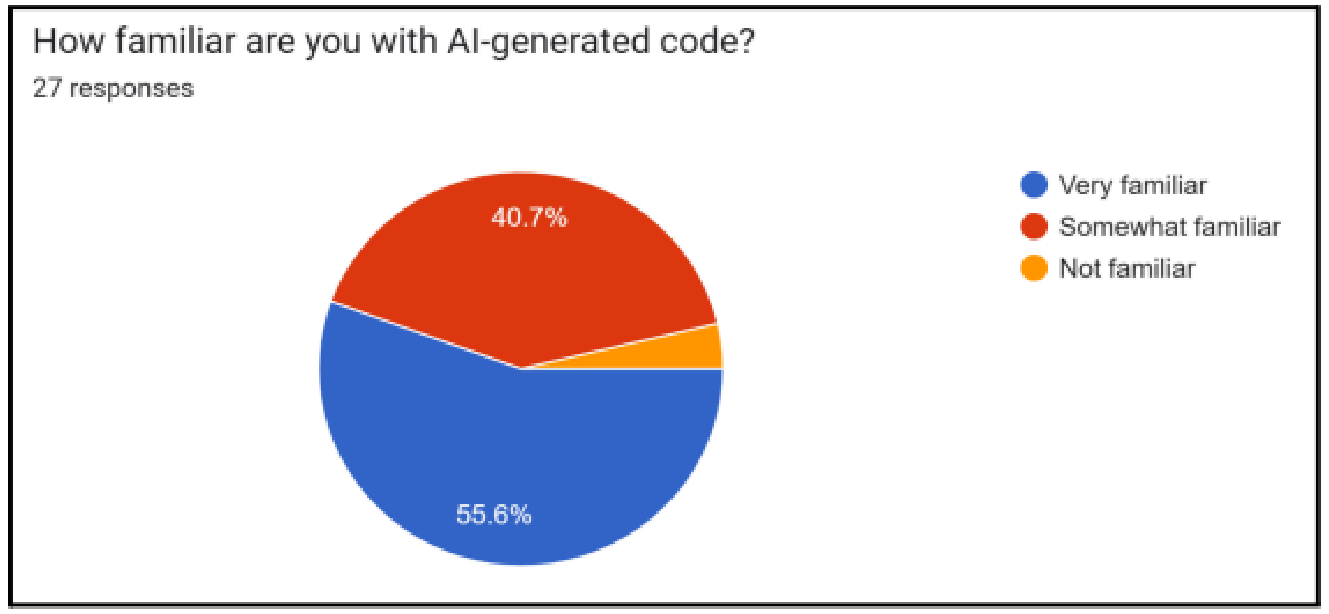
- **Frontend:** HTML, CSS, JavaScript
- **Backend:** Node.js, Express.js, MongoDB
- **Languages:** C++, Python
- **LLMs:** ChatGPT 3.5, Gemini 1.5
- **Analysis Tools:** BigO Calculator, Custom scripts for cyclomatic complexity

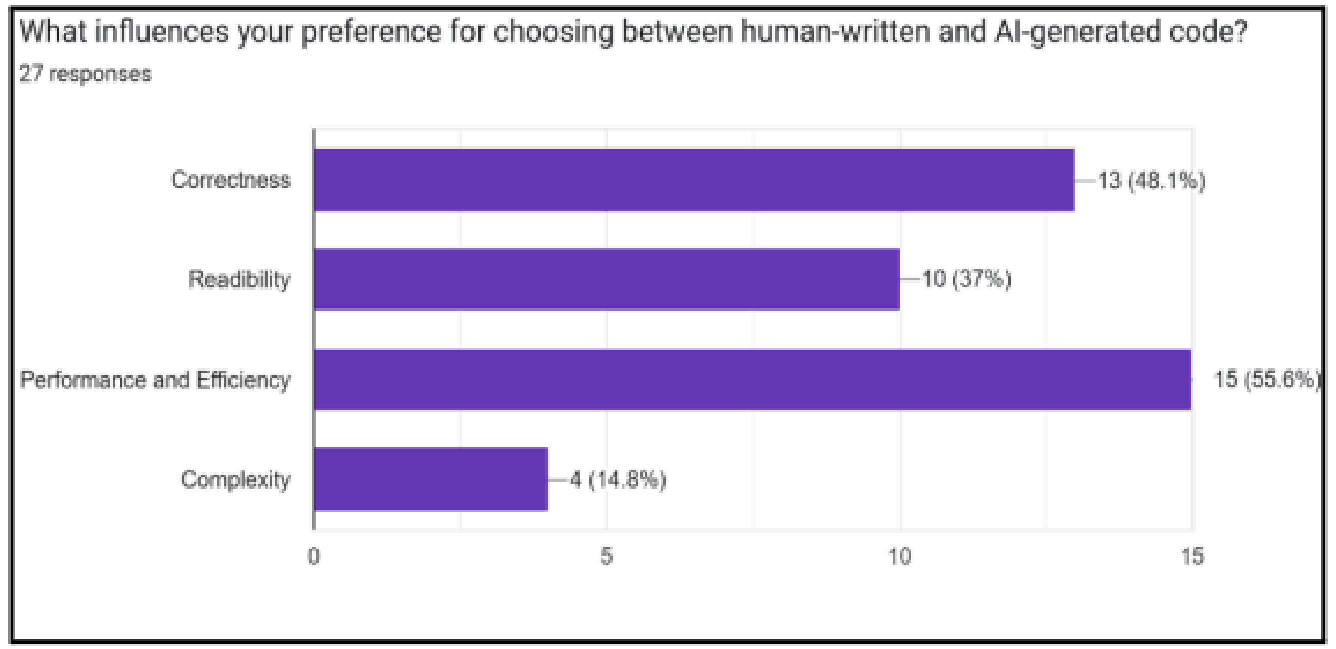
Survey and Real-time Data Collection

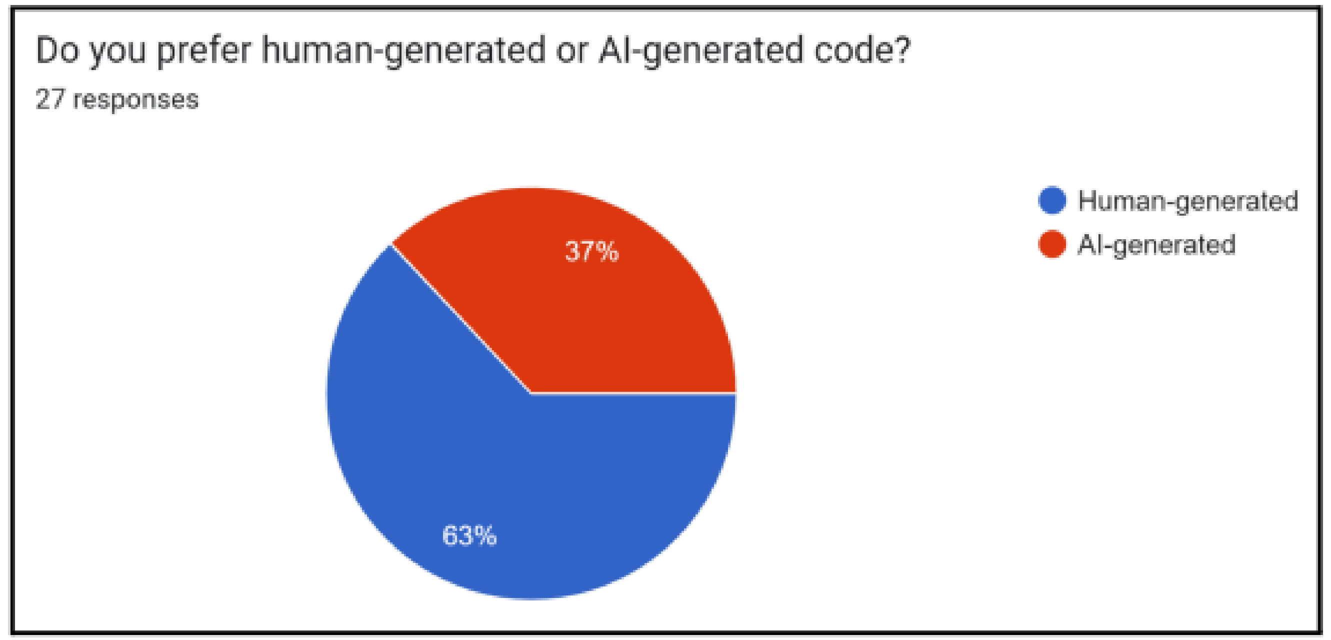
- Conducted surveys to understand user preferences and familiarity with LLM-generated code.
- Collected data on code performance and user satisfaction.
- Real-time data collection via interactive dashboard usage.

Survey data


Which AI code generation tool have you used the most?
27 responses
ChatGPT
Google Gemini
Copilot
74.1%
22.2%


How familiar are you with AI-generated code?
27 responses
Very familiar
Somewhat familiar
Not familiar
40.7%
55.6%


What influences your preference for choosing between human-written and AI-generated code?
27 responses
Correctness
Readibility
Performance and Efficiency
Complexity
13 (48.1%)
10 (37%)
15 (55.6%)
4 (14.8%)
0
5
10
15


Do you prefer human-generated or AI-generated code?
27 responses
Human-generated
AI-generated
37%
63%

Challenges

- Ensuring consistency in code samples.
- Handling complex and nuanced coding problems with LLMs.
- Balancing performance and readability in AI-generated code.
- Addressing prompt sensitivity in LLM outputs.

Future Scope

- Test additional programming languages to broaden the analysis.
- Evaluate newer and more advanced LLMs for code generation.
- Apply the approach to other domains like content generation and data analysis.
- Enhance the interactive dashboard for better usability and insights.

Conclusion

- Human-written code is more reliable for complex tasks but LLM-generated code excels in simplicity and efficiency.
- Clear and concise prompts are crucial for effective LLM performance.
- AI tools are valuable for routine coding tasks, complementing human expertise in specialized scenarios.
- Ongoing research and development can further optimize LLMs for various applications.

References

Research Papers Studied:

- Large Language Models for Code Analysis: Do LLMs Really Do Their Job?
- Advancing GenAI Assisted Programming - A Comparative Study on Prompt Efficiency and Code Quality Between GPT-4 and GLM-4
- Who Answers It Better? An In-Depth Analysis of ChatGPT and Stack Overflow Answers to Software Engineering Questions
- Application of Large Language Models to Software Engineering Tasks: Opportunities, Risks, and Implications

THANK YOU