MULTI-LANGUAGE CODE GENERATION WITH CHATGPT: INSIGHTS INTO AI'S CROSS-LINGUSTIC PROGRAMMING ABILITIES

ABSTRACT

The integration of Large Language Models (LLMs) like ChatGPT into software development has automated code generation from natural language prompts, promising significant productivity gains. However, a critical gap exists in understanding the consistency and quality of this AI-generated code across different programming languages. This research presents a comprehensive, cross-linguistic analysis of ChatGPT-4o's code generation capabilities, systematically comparing its performance in Python and Java. By evaluating solutions for 600 problems from the LeetCode platform, this study assesses functional correctness, runtime performance, memory usage, and critical code quality metrics such as documentation and exception handling. The findings reveal a distinct performance trade-off: Java code demonstrates superior runtime efficiency, especially for complex tasks, while Python code excels in memory efficiency. However, both languages exhibit significant deficiencies in software engineering best practices, including inadequate documentation and poor error management. This study provides crucial insights for developers seeking to leverage AI-assisted programming effectively and highlights the need for language-specific optimizations in future LLM development.

EXISTING SYSTEM

The existing system for AI-powered code generation involves the direct application of general-purpose LLMs, such as ChatGPT-3.5, to interpret natural language prompts and produce source code. These systems are trained on vast corpora of text and code from the internet, enabling them to generate syntactically correct code for a wide array of tasks. The primary interface is a conversational prompt, where a developer describes a problem or requirement, and the model returns a code snippet in a specified language. Evaluation in existing research has typically centered on functional correctness—whether the code passes a set of test cases—with limited analysis of non-functional attributes like efficiency and long-term maintainability.

Disadvantages of the Existing System:

1. Limited Cross-Linguistic Analysis: Prior research is predominantly monolingual, focusing heavily on Python. This provides an incomplete picture, failing to reveal how an AI model's performance and coding style vary with the distinct paradigms, type systems, and standard libraries of different languages like Java.
2. Inadequate Focus on Performance Metrics: The evaluation criteria in existing systems are narrow, primarily concerned with whether the code "works." There is a lack of systematic investigation into critical runtime performance (execution speed) and memory usage, which are decisive factors in production-level software deployment.
3. Neglect of Software Engineering Quality: Generated code from existing systems often lacks the hallmarks of professional software development. This includes a pervasive absence of documentation (comments), poor or non-existent exception handling, and a general disregard for long-term maintainability and robustness, making the code brittle and difficult to integrate into larger projects.

PROPOSED SYSTEM

The proposed system is a memory-centric optimization framework for neural networks, engineered to create "Efficient Brains" for digital medicine. It fundamentally rethinks how CNNs and SNNs manage memory to minimize footprint and energy use while maintaining high diagnostic accuracy.

Advantages of the Proposed System:

1. Dramatically Reduced Memory Consumption: Through techniques like weight pruning, quantization, and efficient encoding of spikes in SNNs, the proposed system significantly compresses the model size. This enables the deployment of sophisticated AI models on hardware with strict memory constraints.
2. Enhanced Energy Efficiency: By minimizing data movement and leveraging the event-driven nature of optimized SNNs, the system drastically reduces power consumption. This facilitates the development of portable, battery-powered diagnostic tools that can operate for extended periods.

3. Maintained High Performance with Faster Inference: The memory-centric design allows for more efficient use of memory bandwidth and computational cores, leading to lower latency and faster inference times. This supports real-time analysis of medical data, a critical requirement for timely clinical decision-making.

SYSTEM REQUIREMENTS

➢ **H/W System Configuration:-**

➢ Processor	-	Pentium –IV
➢ RAM	-	4 GB (min)
➢ Hard Disk	-	20 GB
➢ Key Board	-	Standard Windows Keyboard
➢ Mouse	-	Two or Three Button Mouse
➢ Monitor	-	SVGA

SOFTWARE REQUIREMENTS:

❖ **Operating system**	**:**	Windows 7 Ultimate.
❖ **Coding Language**	**:**	Python.
❖ **Front-End**	**:**	Python.
❖ **Back-End**	**:**	Django-ORM
❖ **Designing**	**:**	Html, css, javascript.
❖ **Data Base**	**:**	MySQL (WAMP Server).