Uncertainty-Guided Hierarchical Multi-Agent Planning for Code Inspection and Debugging

Anonymous ACL submission

Abstract

The rise of large language models and their utilization in generating or modifying code is now considered common. AI agents now elevate this concept with autonomous and environment-aware decision making for complex code generation, debugging, refactoring, and testing. However, this convenient innovation in is still in its early years, and will require extensive research advancement before it becomes an optimal solution for practical generative software development. In this paper, we propose a multi-agent framework for autonomous code inspection and debugging of AI generated code through an uncertainty-aware hierarchical multi-agent structure. This work focuses on reducing logical and syntax errors in generated code through uncertainty-awareness, and preventive cascading error propagation in large language models caused by confidence blindness. Retrieval augmented generation is used to provide knowledge context from established text-to-code datasets. Proof-of-concept experiments show a pass rate (Pass@1) of 83.60% with a average final uncertainty of 0.0252 and average 0.4731 CodeBleu Score.

1 Introduction

The advent of OpenAI's ChatGPT a few years ago ushered in an era of widespread LLM use, including its unsurprising adoption into Software Development Life Cycle (SDLC). With it comes some concerning and pressing issues such as code quality, reproducibility, explainability, bias in generated code, dependency and architectural drift in large and modular projects (Huang et al., 2025; Zhong and Wang, 2024).

LLM-based Multi-Agent Systems (MASs) are recently being considered as an enhanced and improved utilization of the exceptional reasoning and planning capabilities of LLMs. There has been significant research in the application of LLM-based multi-agent systems that interact with complex environments and solve intricate and nuanced tasks (Li et al., 2024). Multiple specialized agents in a heterogeneous system can communicate and collaborate to achieve a sophisticated solutions for a given objective.

Multi-agent systems can be used to generate high quality code compared to code generated by a single LLM (Idrisov et al., 2025). Guided code generation shows remarkable potential, while still facing significant limitations for complex, long-context reasoning tasks (Almorsi et al., 2024). This gives rise to the *hallucination* problem, where LLMs generate false, fabricated or nonsensical output with high confidence leading to cascading error propagation - a snowball effect. (Cash et al., 2025). However, agentic system can also inspect, verify, and improve the the correctness of the generated code. Inspection or review is the most cost-effective of the software Verification and Validation (V&V) process finding up to 60% defects and inefficiencies in code implementations (Sommerville, 2016).

In this spirit, we propose a hierarchical multi-agent environment where an agent generates code from an user prompt, and then calculates the uncertainty score to evaluate the LLM's confidence. The code is then inspected and evaluated by another agent to find whether it passes unit test(s), and assigned a score if it doesn't. The original agent is then given the feedback and is told to generate it again until it reaches the goal. All of this is managed by an orchestrator agent.

We present the following research findings in this study:

If a multi-agent system includes

1. Quantifiable uncertainty or confidence metrics as context,

2. An external knowledge context from a RAG pipeline,

3. And, a recursive self-judgment and self-correction loop,

the accuracy, error/bugs, and pass rate for AI generated code are improved.

2 Related Work

Zhao et al. (2024) propose MAGE, the first open-source multi-agent AI system for generating RTL code (Verilog) that addresses the limitations of single-LLM approaches by introducing high-temperature candidate sampling with debugging and a novel Verilog-state checkpoint mechanism for early error detection. The system achieves 95.7% syntactic and functional correctness on the VerilogEval-Human 2 benchmark, surpassing Claude-3.5-sonnet by 23.3%.

? address the overthinking problem in Chain-of-Thought (CoT) reasoning for code generation, where LLMs unnecessarily apply complex reasoning to simple tasks, by proposing Uncertainty-Aware Chain-of-Thought (UnCert-CoT) that uses entropy-based and probability differential-based confidence measures to selectively activate CoT reasoning only when uncertainty is high. Their approach demonstrates up to 6.1% improvement in PassRate accuracy on the Mostly Hard Python Problems (MHPP) benchmark, showing that targeted application of reasoning based on uncertainty quantification can improve both efficiency and accuracy in code generation tasks.

(Xu et al., 2024) address the challenge of detecting inconsistencies between code and its comments, a problem that can mislead developers and cause bugs. They argue that existing detection models are often trained on "noisy" datasets with labeling errors. The authors propose a novel approach called MCCL (Method-Comment Confidence Learning), which first uses a detection model to identify likely mislabeled or noisy data and then employs a confidence learning component to "denoise" the dataset. The detection model is then retrained on this cleaned dataset. This two-stage process was tested on 1,518 open-source projects, where MCCL achieved an average F1-score of 82.6%, outperforming state-of-the-art methods by 2.4% to 28.0%.

Zhang et al. (2024) propose Planning with Multi-Constraints (PMC), a zero-shot methodology for collaborative LLM-based multi-agent systems that decomposes complex constraint-intensive tasks into hierarchical subordinate tasks, each mapped to executable actions, achieving a 42.68% success rate on TravelPlanner compared to GPT-4's 2.92% and outperforming GPT-4 with ReAct on API-Bank by 13.64%. Their hierarchical task decomposition approach for handling multiple constraints in planning aligns with our proposed hierarchical verification architecture, though their focus is on action planning for general tasks rather than code generation with logical consistency verification.

(Lee et al., 2025) introduce UniDebugger, a novel end-to-end framework designed to overcome the limitations of large language models in software debugging. The authors argue that previous LLM-based methods are fragmented, focusing only on isolated debugging steps. Their proposed solution is a hierarchical multi-agent framework that employs multi-agent synergy to model the entire cognitive workflow of a human developer, with different agents specializing in specific components of the debugging process. Experiments on the Defects4J benchmark show that UniDebugger significantly outperforms existing state-of-the-art methods, fixing 1.25 to 2.56 times more bugs at the repository level.

3 Approach

To address and overcome the challenges of common LLM drawbacks such as hallucination, cascading error amplification, and confidence blindness in LLM-based code generation, we propose an uncertainty-aware multi-agent planning framework. This approach will integrate a hierarchical argentic structure with an uncertainty quantification and confidence score mechanism to improve the reliability and accuracy of automated error correction in a codebase. Our work will focus on investigating how uncertainty and confidence metrics affect code quality and error rates. This approach includes the following components:

3.1 Hierarchical Multi-Agent Architecture

The system utilizes a hierarchical multi-agent planning framework to overcome the limitations of single-agent systems with limited context windows, as well as non-coordinated multi-agent frameworks. The architecture employs three different specialized agents with distinct roles and a retrieval augmented generation (RAG) knowledge base in the following hierarchy:

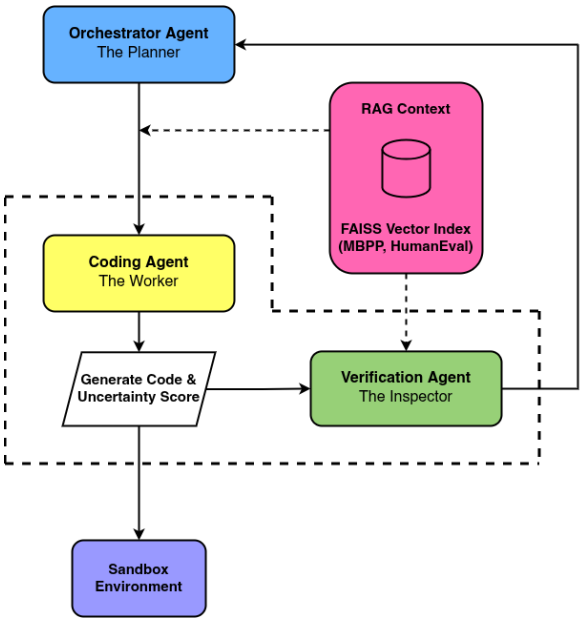

Figure 1: Uncertainty-Guided Hierarchical Multi-Agent Structure

```
return f"""
        {rag_context}

        Your task is to write a single Python function named `{function_name}`
        to solve the following problem. Do not include any import statements.
        Your response must be *only* a markdown code block starting with ```python.

        Problem:
        {task_description}
        """
```

Figure 2: System Prompt

3.1.1 Orchestrator Agent

It is the top-level planning agent in charge of managing the entire workflow, decomposing the initial task, and route information between the other agents. It decides when to accept code, when to verify it and when to send it for refinement. Implementation uses LangGraph's StateGraph with conditional routing based on uncertainty thresholds and verification results. The orchestrator maintains complete conversation state and enforces maximum iteration limits.

4 Results and Analysis

4.1 Implementation Status

The full hierarchical multi-agent framework has been successfully implemented with all proposed components operational. The system includes:

- LangGraph-based orchestration with conditional state routing
- FAISS-powered RAG system with MiniLM embeddings
- Uncertainty quantification from token log-probabilities
- Dual-validation verification (static analysis + unit testing)
- Recursive refinement loop with feedback integration

4.2 Primary Performance Metrics

Experimental evaluation (non-LangGraph) on the complete MBPP-Pro dataset (378 tasks) demonstrates:

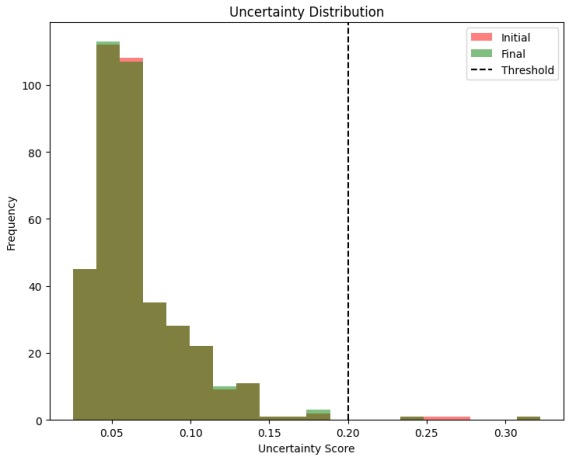

Figure 3: Uncertainty Distribution

The experiments was conducted on the 378 tasks in the MBPP dataset and the uncertainty threshold was set to 0.2 or 20% with maximum 5 attempts. 316 of the tasks passed the tests with a pass@1 rate of 83.60%. The average CodeBleu score was 0.4731 and average uncertainty after the verification loop was 0.0252. he strategy of verifying both high and low uncertainty cases successfully prevents false acceptance.

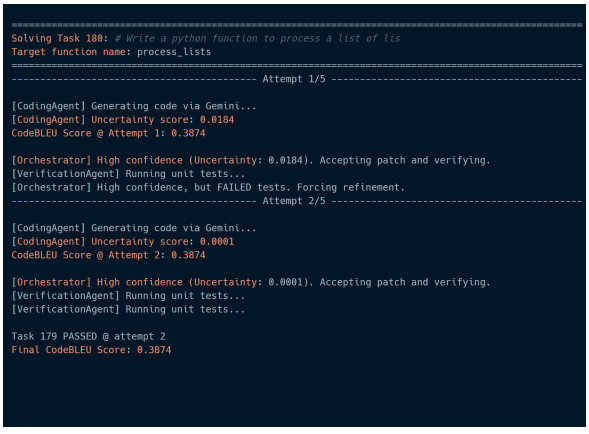

Figure 4: Example Output

4.3 RAG Impact Analysis

Retrieving $k = 2$ relevant examples provides strong contextual guidance, enabling the model to recognize problem patterns and generate correct solutions more reliably.

FAISS index caching reduces RAG overhead to 0.3 seconds per query (compared to 2-3 minutes for initial index building), making the approach practical for production use.Hierarchical Multi-Agent Framework for Recursive Uncertainty-Aware Code Inspection and Debugging

5 Conclusion

This research proposes a novel approach to address the limitations in LLM-based code generation through an uncertainty-aware code generation and error corrected planning system using hierarchical multi-agent architecture. By integrating uncertainty quantification and recursive verification mechanism, we can tackle two fundamental concerns in AI-assisted code generation in production environments: the propagation and amplification of logic errors, and the inability to reliably assess confidence in code generation.

Limitations

Despite promising results, the experimentation faces limitations with the experimental setup and resource availability.

Our current experimental setup, for the above reason, uses the same LLM model for all agents. While self-assigned uncertainty/confidence is an important metric for the experimentation, we also need to validate it with other agent's score since the score might be biased or incorrect due to hallucination and other factors. Similarly, we do not employ any agentic tools at the moment, and without them the experiments cannot be considered comprehensive.

We have set comfortable uncertainty threshold and low maximum attempt for the agents to generate initial results. This needs to be fine-tuned for optimal outcome.

References

Amr Almorsi, Mohanned Ahmed, and Walid Gomaa. 2024. Guided code generation with llms: A multi-agent framework for complex code tasks. In *2024 12th International Japan-Africa Conference on Electronics, Communications, and Computations (JAC-ECC)*, pages 215–218.

Trent N Cash, Daniel M Oppenheimer, Sara Christie, and Mira Devgan. 2025. Quantifying uncert-ai-nty: Testing the accuracy of llms' confidence judgments. *Memory & Cognition*, pages 1–26.

Dong Huang, Jie M. Zhang, Qingwen Bu, Xiaofei Xie, Junjie Chen, and Heming Cui. 2025. Bias testing and mitigation in llm-based code generation. *ACM Trans. Softw. Eng. Methodol.* Just Accepted.

Baskhad Idrisov, Esther Eisenacher, and Tim Schlippe. 2025. Program code generation: Single llms vs. multi-agent systems. In *2025 7th International Conference on Natural Language Processing (ICNLP)*, pages 121–127.

Cheryl Lee, Chunqiu Steven Xia, Longji Yang, Jentse Huang, Zhouruixing Zhu, Lingming Zhang, and Michael R Lyu. 2025. Unidebugger: Hierarchical multi-agent framework for unified software debugging. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 18248–18277.

Xinyi Li, Sai Wang, Siqi Zeng, Yu Wu, and Yi Yang. 2024. A survey on llm-based multi-agent systems: workflow, infrastructure, and challenges. *Vicinagearth*, 1(1):9.

Ian Sommerville. 2016. *Software engineering*, tenth edition edition. Always learning. Pearson, Boston Columbus Indianapolis New York San Francisco Hoboken Amsterdam Cape Town Dubai London.

Zhengkang Xu, Shikai Guo, Yumiao Wang, Rong Chen, Hui Li, Xiaochen Li, and He Jiang. 2024. Code comment inconsistency detection based on confidence learning. *IEEE Transactions on Software Engineering*, 50(3):598–617.

Cong Zhang, Derrick Goh Xin Deik, Dexun Li, Hao Zhang, and Yong Liu. 2024. Planning with multi-constraints via collaborative language agents. *Preprint*, arXiv:2405.16510.

Yujie Zhao, Hejia Zhang, Hanxian Huang, Zhongming Yu, and Jishen Zhao. 2024. Mage: A multi-agent engine for automated rtl code generation. *Preprint*, arXiv:2412.07822.

Li Zhong and Zilong Wang. 2024. Can llm replace stack overflow? a study on robustness and reliability of large language model code generation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(19):21841–21849.