CONTEXTUAL AUGMENTED MULTI-MODEL PROGRAMMING (CAMP): A LOCAL-CLOUD COPILOT SOLUTION

Yuchen Wang
Nanyang Technological University
Singapore
yuchen011@e.ntu.edu.sg

Shangxin Guo
City University of Hong Kong
Hong Kong
sxguo2-c@my.cityu.edu.hk

Chee Wei Tan
Nanyang Technological University
Singapore
cheewei.tan@ntu.edu.sg

ABSTRACT

To bridge the strengths of cloud-based Large Language Models (LLMs) in code generation and the adaptability of locally integrated tools, we introduce CAMP, a collaborative multi-model copilot framework for AI-assisted programming. CAMP employs context-aware Retrieval-Augmented Generation (RAG), dynamically retrieving relevant information from local codebases to construct optimized prompts tailored for code generation tasks. This hybrid strategy enhances LLM effectiveness in local coding environments, yielding a 12.5% performance boost over non-contextual generation and a 6.3% gain compared to a baseline RAG implementation. We demonstrate the practical application of CAMP through "Copilot for Xcode," supporting tasks such as code completion, bug detection, and documentation generation. Its success led to integration with GitHub Copilot, underscoring the real-world impact and scalability of our approach in evolving AI-driven software development practices.[1]

1 INTRODUCTION

AI-assisted programming has emerged as a transformative force in software engineering, enhancing productivity by automating tasks, identifying bugs, and improving code quality Wong et al. (2023). By offloading repetitive processes, developers can focus more on creative and complex problem-solving. Recent advances in Large Language Models (LLMs) have opened new frontiers in this space, enabling conversational coding assistants that can be integrated into programming workflows Li et al. (2022); Chen et al. (2021). This vision echoes the early insights of Edsger W. Dijkstra Dijkstra ((transcribed), emphasizing the interplay between computational logic and human ingenuity.

Popular LLM-powered tools such as Codeium Codeium (2023), GitHub Copilot Friedman (2021), OpenAI ChatGPT OpenAI (2023), and Amazon CodeWhisperer Amazon (2022) deliver state-of-the-art code generation capabilities via cloud APIs. Despite their effectiveness, these tools often incur computational costs and latency, and they may struggle to seamlessly operate within local development environments.

The advent of Retrieval-Augmented Generation (RAG) has further revolutionized AI-assisted programming Lewis et al. (2020). By incorporating retrieval mechanisms that extract relevant context from a corpus before feeding it into LLMs, RAG enhances the model's relevance and adaptability and shows great potential in enhancing programming tasks.

[1]This work was originally published as a full paper in IEEE CAI 2025. The current version is a concise presentation for this workshop, highlighting the key contributions and encouraging further discussion within the community.

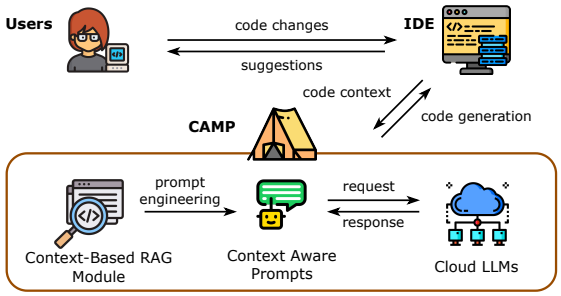

Figure 1: Overview of CAMP: A hybrid AI-assisted programming framework that combines cloud-based LLMs with local context retrieval using RAG.

This paper presents CAMP, a multi-model copilot programming solution that leverages local code context retrieval and cloud LLMs to optimize context-aware code generation. As shown in Figure 1, CAMP integrates cloud LLMs into local development environments, employing a RAG module that dynamically learns from code context to optimize prompt construction. This methodology is implemented in *Copilot for Xcode*[2], a tool providing automatic code completion, error detection, and documentation, synchronized with user interactions and codebase updates. The project was open-sourced and later integrated into GitHub Copilot for Xcode Tan et al. (2023); Wang et al. (2025); GitHub (2024).

2 METHODOLOGY

Our proposed RAG module consists of three major components: (I) a context retriever $\mathcal{R}_{\eta'}(c|x)$ that captures contextual information c from the local development environment x, (II) a content retriever $\mathcal{R}_{\eta}(z|x,c)$ that generates relevant content given the current context and the original input, and (III) a prompt constructor $\mathcal{G}_{\theta}(y_i|x,c,z,y_{1:i-1})$ that creates prompts to assist LLMs from the retrieved information and user queries.

As presented by Figure 2, given the local development environment at a certain timestamp t, the contextual information c is first obtained and utilized for the retrieval of the top-ranked relevant content information z. Both the context c and content z are then utilized in prompt construction for LLMs requests. As the local development environment evolves with t, this workflow synchronizes with user actions and codebase changes, providing on-demand functionalities.

2.1 CONTEXT RETRIEVER

The context retriever obtains contextual information from the local development environment that maximizes the insights brought to the next step. We define τ_c to be the upper limit of the contextual entries to include and have

$$\begin{aligned}\mathcal{R}_{\eta'}(x) &= \text{agg}([\eta'_0 c_0, \eta'_1 c_1, \ldots, \eta'_{\tau_c} c_{\tau_c}]) \\ &= \text{agg}([\eta'_0 f_0(x_0), \eta'_1 f_1(x_1), \ldots, \eta'_{\tau_c} f_{\tau_c}(x_{\tau_c})]) \\ &= \text{agg}(\eta' \cdot f(x))\end{aligned}$$

where $f_i(\cdot)$ represents the detailed data processing for each contextual entry and agg represents the aggregation method. We normalize by setting $\Sigma\eta'_i = 1$ and assign a larger value to η'_i to increase the influence of the corresponding c_i. For null entries, where the number of selected components is below the limit τ_c, we set $\eta'_i = 0$.

We eventually select "cursor position", "absolute repository path", "cached build artifacts", and "index information" as our sources of contextual information based on trials and errors. With the assumption that the relative importance of different factors in the local development environment remains stable, we can obtain a fixed set of optimal η' values over time and across data $(\mathcal{X}, \mathcal{Y})$.

[2] https://github.com/intitni/CopilotForXcode

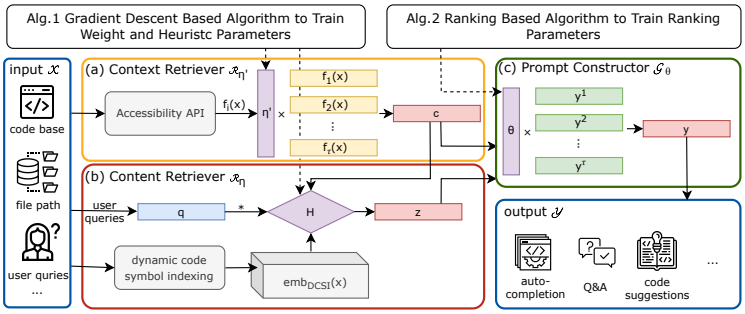

Figure 2: Overview of the RAG module. (a) Context retriever $\mathcal{R}_{\eta'}$ that retrieves contextual information from the local development environment. (b) Content retriever $\mathcal{R}_\eta$ that searches for the most relevant information from local content. (c) Prompt constructor $\mathcal{G}_\theta$ that creates context-aware prompts.

2.2 Content Retriever

The objective of the content retriever is to deliver highly relevant content z that enhances prompt construction with local, context-aware information. This aligns with the core principle of RAG, which provides "documents" to transform general models into specialized ones. The retrieved contextual information c serves two purposes in this step: supporting codebase embedding and facilitating content search.

To balance the modeling power of neural network based encoders with the computational efficiency of lightweight methods, we propose and employ dynamic code symbol indexing ("DCSI") which enables precise source code analysis by capturing each coding token's symbol information, position, relationships with neighboring tokens, and dependencies within the programming graph. It also supports dynamic updates, adapting to changes such as codebase edits and maintaining synchronization with the local context, while remaining computationally efficient. We thus have the following simplified model where the consistent embedding function makes the heuristic H a square matrix.

$$p_\eta(z|x,c) = \frac{\exp\left(\text{emb}_{\text{DCSI}}(z)^T H \text{emb}_{\text{DCSI}}(x)\right)}{\sum_{z'} \exp\left(\text{emb}_{\text{DCSI}}(z')^T H \text{emb}_{\text{DCSI}}(x)\right)} \tag{1}$$

We present a gradient descent algorithm to obtain the optimal values of H and other parameters. Given the embedding function and heuristic matrix, the content retriever identifies

$$\mathcal{R}_\eta(x,c) = \underset{z \in \text{emb}(x)}{\arg\max}\, p(c|H, q*)$$

where q represents the optional user query which is provided in cases involving user interactions, such as in Q&A scenarios.

The goal of the prompt constructor is to determine the optimal combination and ranking of the components. Denote the ith prompt as y_i and the kth configurable component as y^k. Without loss of generality, let τ_k represent the maximum number of configurable components. Each y_i is thus an ordered array of y^k. Consequently, we have

$$\begin{aligned}\mathcal{G}_\theta(x,c,z,y_{1:i-1}) = y_i &= \text{order}([y^1, y^2, \ldots, y^{\tau_k}]) \\ &= [\theta_1 \quad \theta_2 \quad \ldots \quad \theta_k]^T \begin{bmatrix} y^1 & y^2 & \ldots & y^k \end{bmatrix}^T\end{aligned}$$

where θ_k are standard unit vectors that mark the component located on the kth position of y_i.

2.3 Implementation Details on Xcode

We demonstrate the practical utility of Camp by implementing it as a plugin for Xcode. This serves as a pilot trial to validate the methodology's robustness in challenging coding environments with sandboxed architecture that imposes strict restrictions and offers limited access to local contextual information. To address these challenges, we employed: 1) XPC service-level communication to enable interaction with language servers and facilitate real-time code suggestions in the UI, and 2) the `Accessibility API` to capture rich contextual data. These solutions enable accurate

Table 1: Evaluation Results for Code Generation Tasks on CoderEval. The performance of CAMP is compared to baseline models.

Model	class-runnable			file-runnable			project-runnable		
	Pass@1	Pass@5	Pass@10	Pass@1	Pass@5	Pass@10	Pass@1	Pass@5	Pass@10
CloudOnly	8.73%	12.57%	14.55%	21.03%	29.09%	32.35%	9.37%	12.08%	13.04%
BaseRAG	19.84%	35.06%	40.91%	24.98%	35.94%	39.01%	15.66%	21.89%	24.62%
FileContext	**31.23%**	**43.41%**	**47.30%**	29.52%	37.80%	42.30%	11.08%	16.87%	17.92%
CAMP	28.96%	41.72%	46.07%	**35.30%**	**43.45%**	**45.80%**	**21.91%**	**25.05%**	**26.43%**

prompt construction and effective integration with the IDE environment, laying the groundwork for future expansions to other IDEs. When users update their code, CAMP retrieves contextual information, constructs enriched prompts, and facilitates real-time AI-assisted programming. The system dynamically interacts with Xcode to deliver tailored code suggestions and handle questions through the chat panel, thereby enhancing developer productivity and overall coding experience.

3 EVALUATION

We evaluate the performance of CAMP using the CoderEval benchmark Yu et al. (2024), a pragmatic code generation evaluation dataset designed to measure the performance of generative pre-trained models. The benchmark comprises 230 test cases categorized into six runnable levels, from single-function to project-level tasks. For our experiments, we selected the top three categories with the highest runnable levels, representing the most common real-world use cases and encompassing diverse contexts. We compare CAMP against the following baseline models, using GPT-3.5-Turbo as the cloud-based LLM.

- **CloudOnly**: Inputs are processed solely by the cloud-based model, with no local processing or context retrieval.
- **BaseRAG**: Implements standard RAG techniques as proposed by Lewis et al. (2020).
- **FileContext**: A variant of CAMP that prioritizes context retrieved from the currently open files in the IDE. This lightweight version balances performance and resource efficiency.

The results, as summarized in Table 1, demonstrate its superiority over baseline models in code completion tasks across varying complexities, as well as its effectiveness in real-world programming scenarios. Typically, it achieves a 12.5% and 6.3% improvement over CloudOnly and BaseRAG, respectively, in Pass@1 accuracy for the project-runnable category. Compared to the CloudOnly model, CAMP achieves advantageous results in all tasks, demonstrating the impact of retrieved content in enhancing LLM prompts. Similarly, CAMP outperforms the BaseRAG model, highlighting the effectiveness of its context-based retrieval mechanisms in understanding the codebase and generating context-aware solutions. The FileContext model shows comparable performance to CAMP for lower runnable levels, such as class-runnable tasks, but falls behind in cross-file and project-level scenarios. This outcome emphasizes the necessity of broader context retrieval, a key advantage enabled by RAG techniques. The results also suggest that dynamically adjusting the retrieval scope based on task complexity can optimize computational resource without compromising accuracy. For instance, narrowing the retrieval range to specific files for class-level tasks can reduce computational overhead while maintaining high performance.

4 CONCLUSION

This paper presented CAMP, a multi-model programming copilot solution that leverages context-based Retrieval-Augmented Generation (RAG) to enhance AI-assisted programming. By introducing dynamic context retrieval from local codebases, CAMP optimizes context-aware prompt construction, bridging the gap between the generative capabilities of cloud-based LLMs and the contextual efficiency of local models. It also fosters dynamic collaboration between cloud LLMs and local models, paving the way for advanced AI-assisted programming solutions. By enabling seamless integration of human expertise with AI tools, CAMP aligns with Dijkstra's vision of augmenting human intelligence in software development, advancing toward more efficient, reliable, and user-centric programming practices.

ACKNOWLEDGMENTS

This research was supported by the Singapore Ministry of Education Academic Research Fund under Grant RG91/22.

REFERENCES

CodeWhisperer Amazon. AI code generator - amazon codewhisperer. https://aws.amazon.com/codewhisperer, 2022. Accessed on June 1, 2023.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Exafunction Codeium. Codeium - free AI code completion & chat. https://codeium.com/, 2023. Accessed on June 1, 2023.

Edsger Wybe Dijkstra. A preliminary investigation into computer assisted programming. *E. W. Dijkstra Archive (EWD 237)*, (transcribed) 2007.

Nat Friedman. Introducing github copilot: your AI pair programmer, 2021.

GitHub. Github copilot code completion in xcode is now available in public preview. https://github.blog/changelog/2024-10-29-github-copilot-code-completion-in-xcode-is-now-available-in-public-preview/, October 2024. Accessed: January 5, 2025.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33: 9459–9474, 2020.

Zhiyu Li, Shuai Lu, Daya Guo, Nan Duan, Shailesh Jannu, Grant Jenks, Deep Majumder, Jared Green, Alexey Svyatkovskiy, Shengyu Fu, et al. Automating code review activities by large-scale pre-training. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022.

OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Chee Wei Tan, Shangxin Guo, Man Fai Wong, and Ching Nam Hang. Copilot for Xcode: exploring AI-assisted programming by prompting cloud-based large language models. *preprint arXiv:2307.14349*, 2023.

Yuchen Wang, Shangxin Guo, and Chee Wei Tan. From code generation to software testing: AI Copilot with context-based RAG. *IEEE Software*, pp. 1–9, 2025. doi: 10.1109/MS.2025.3549628.

Man-Fai Wong, Shangxin Guo, Ching-Nam Hang, Siu-Wai Ho, and Chee-Wei Tan. Natural language generation and understanding of big code for AI-assisted programming: A review. *Entropy*, 25(6):888, 2023.

Hao Yu, Bo Shen, Dezhi Ran, Jiaxin Zhang, Qi Zhang, Yuchi Ma, Guangtai Liang, Ying Li, Qianxiang Wang, and Tao Xie. Codereval: A benchmark of pragmatic code generation with generative pre-trained models. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, pp. 1–12, 2024.