Secure Coding Unleashed: Boosting Productivity With On-Premise LLM-Powered IDE Plugins

Vasilii Krikunov[1] , **Nikolay Kotlyarov**[2] , **Eugenii Nikolaev**[3,4] and **Vasily Konovalov**[5,1]
[1]MIRAI
[2]Kazakh-British Technical University
[3]ITMO University
[4]AI Talent Hub
[5]Core Language Technologies
vasilkrikunov13@gmail.com

Abstract

The integration of Large Language Model (LLM)-based code assistants into IDEs has boosted developer productivity, but cloud-based solutions pose severe privacy risks for enterprises handling proprietary code. Existing on-premise alternatives lack semantic code retrieval, multi-GPU optimization, and rigorous productivity validation. We present an enterprise-oriented, customizable IDE plugin that uses internally hosted LLMs, combining client-side tiered retrieval (Personal CodeRAG with SiamBERT/CodeSage) and vLLM-optimized inference on dedicated hardware. No code leaves the organization. Our benchmark of four inference frameworks shows vLLM and LMDeploy achieve ≈20 ms time-to-first-token on LLaMA-3-8B, suitable for real-time interaction. In a three-month study with 100 developers, the plugin reduced mean cycle time by 15.4%, with perceived productivity gains of 50-70% for tasks like code explanation and unit test generation. This work provides the first systematic evaluation of on-premise LLM-assisted coding that quantifies both objective and subjective productivity impacts.

1 Introduction

Large Language Models (LLMs) have revolutionized code generation, with cloud-based assistants like GitHub Copilot cutting repetitive coding time by up to 55% [Ziegler *et al.*, 2022]. Yet for enterprises handling sensitive intellectual property: financial algorithms, medical device software, or proprietary trading systems, these productivity gains are largely inaccessible. The reason is not technical capability but data privacy: every code snippet sent to a third-party API creates exposure risks, from unauthorized model training to GDPR violations [Pearce *et al.*, 2025].

On-premise alternatives do exist: Ollama, LocalAI, and other local LLM wrappers, but they are not enterprise-grade. They lack semantic code retrieval tailored to local repositories, offer no optimization for multi-GPU inference, and provide no systematic evaluation of developer productivity. Organizations thus face a false choice: adopt privacy-violating cloud tools or accept sluggish, unvalidated local solutions.

We resolve this false choice with a customizable, on-premise IDE plugin that combines privacy guarantees with enterprise performance. Our solution introduces a tiered CodeRAG pipeline: SiamBERT [Peña *et al.*, 2022] for resource-constrained developer workstations and CodeSage [Zhang *et al.*, 2024] for high-memory environments (>16 GB RAM). All indexing and embedding remain client-side-no code ever leaves the organization. Generation is offloaded to a dedicated on-premise NVIDIA A100 server running vLLM, which delivers sub-100ms time-to-first-token while supporting GGUF-quantized models [Yadav and Chintala, 2025] for storage efficiency.

Despite these advances, three open questions remain: (1) Can on-premise LLM inference achieve latency comparable to cloud-based solutions? (2) Does retrieval-augmented generation improve code completion accuracy without exposing proprietary code? (3) How do subjective developer perceptions of productivity align with objective metrics such as cycle time reduction? This paper answers these questions through systematic benchmarking of inference frameworks and a three-month developer study. This paper makes two contributions:

- An on-premise IDE plugin architecture with client-side tiered RAG and vLLM-optimized inference, ensuring privacy without sacrificing responsiveness.
- A systematic benchmark of four inference frameworks (vLLM, TensorRT, LMDeply, TGI) measuring TTFT, inter-token latency, and throughput on a single A100 GPU.

2 Related Work

Research on AI-powered coding assistants has expanded rapidly in recent years, with commercial tools such as GitHub Copilot, Tabnine, and Codeium demonstrating substantial gains in developer productivity and satisfaction [Kalliamvakou, 2022]. Controlled experiments report faster task completion and reduced cognitive load, with one large-scale study showing up to a 55% reduction in programming task time [Ziegler *et al.*, 2022; Peng *et al.*, 2023].

However, nearly all commercial solutions operate as cloud-based services, requiring code snippets to be sent to third-party servers. This creates data exposure risks that are unacceptable for enterprises handling sensitive intellectual property [Pearce *et al.*, 2025]. Existing on-premise alternatives such as Ollama and LocalAI lack semantic code retrieval, multi-GPU optimization, and systematic productivity evaluation gaps this paper addresses.

A parallel work stream investigates Retrieval-Augmented Generation (RAG), in which LLMs are grounded on project-specific artifacts such as code repositories, documentation, and tickets [Aushev *et al.*, 2025]. These systems commonly employ dense vector indexes (e.g., FAISS) and task-specific encoders to retrieve relevant code or documentation snippets as context for generation [Johnson *et al.*, 2021; Nikolaev *et al.*, 2025]. Prior studies have explored code search, code summarization, and bug-fixing pipelines that rely on semantic retrieval over large mono- or polyglot repositories [Microsoft, 2024]. Our Personal CodeRAG component builds on these ideas but emphasizes fully local operation on the developer's workstation, combining a fine-tuned BERT model for code semantics [Peña *et al.*, 2022] with compact GGUF-based LLMs, and separating local retrieval from on-premise or cloud-hosted generation to maintain strong privacy guarantees.

3 AI-Powered Code Assistant Design

We propose a design for the integration of LLM capabilities within Integrated Development Environments (IDE), using the C4 model as a guiding framework. The design addresses key concerns related to data privacy [Jeong, 2023; Karakurt and Akbulut, 2026], system modularity, and context-aware code generation [Feng *et al.*, 2023].[1]

3.1 C4: Context Level

The Context level provides a high-level overview of interactions between users, software systems, and external elements, clarifying system workflows and data exchanges [Richards and Ford, 2019]. The primary actors identified include: (1) **System Users**: developers and administrators interacting directly with the plugin; (2) **Software Systems**: the core AI-powered code assistant plugin; and (3) **External Software Systems**: supplementary third-party services supporting the core functionalities, such as an email server that provides OTP user authentication and various monitoring tools, which are not detailed in this work.

3.2 C4: Container Level

The Container level details essential system components (containers), their roles, and interdependencies. It comprises: (1) **User interface containers**: Single-page applications for the admin interface and chat interactions, along with separate IDE plugins specifically developed for JetBrains and VSCode environments; (2) **Service containers**: These include an admin service, AI-tool back-end service, message broker, LLM-based suggestion and chat services, as well as a centralized database facilitating logging and traceability.

[1] https://vasilkrikunov14.github.io/C4_llm_plugin/

4 Personal CodeRAG

Personal CodeRAG implements a tiered architecture that strictly separates local data management from high-performance inference. To ensure an tight privacy boundary, all indexing and embedding operations are performed exclusively on the client-side. The system utilizes a dual-model embedding strategy: SiamBERT [Peña *et al.*, 2022] serves as a lightweight default for standard workstations, while CodeSage [Zhang *et al.*, 2024] is deployed on machines with more than 16 GB RAM to provide superior semantic grounding. The retrieval pipeline employs a sliding window chunking strategy (512 tokens) and a local FAISS vector index, allowing for sub-millisecond context retrieval without external data exposure. While retrieval remains local, the computationally intensive generation task is offloaded to a dedicated on-premise NVIDIA A100 80GB RAM server. By utilizing the vLLM inference backend, the system maintains the developer's "flow state". This configuration eliminates the risks associated with third-party cloud providers while providing appropriate code generation quality.

5 LLM Component Implementation

Efficient chatbot systems require careful selection of suitable LLMs and inference frameworks, considering factors such as model architecture, scalability, quantization support, compatibility with GGUF models, and real-time response capabilities [Vaswani *et al.*, 2017; Sawarkar *et al.*, 2024].

We analyze several inference frameworks: vLLM, TensorRT (trt), LMDeploy, and Text Generation Inference (TGI) using three models: `LLaMA-3-8B`, `LLaMA-3-70B`, and `Mistral-8x7B`. Evaluation focused on four critical criteria [Joshi *et al.*, 2024]: (i) Concurrency and scalability; (ii) Quantization support; (iii) GGUF model support; (iv) Streaming and real-time inference.

Performance was evaluated using three metrics (Table 1): throughput (tokens per second, Tput), time to first token (TTFT, milliseconds) and inference token latency (ITL, milliseconds). Throughput (Tput) was consistent across frameworks, with `LLaMA-3-8B` and `Mistral-8x7B` achieving around 1,200 tokens/s, and `LLaMA-3-70B` around 600 tokens/s, indicating minimal dependency on the inference framework under standard conditions [Dettmers *et al.*, 2023]. Time to first token (TTFT) showed significant variance: vLLM and LMDeploy consistently delivered faster TTFT (approximately 20 ms for `LLaMA-3-8B`), outperforming TensorRT (50 ms) and TGI (100 ms), highlighting their suitability for real-time scenarios (Table 2).

Model	Users	lmDeploy	MLC-LLM	TGI	TRT-LLM	vLLM
Llama-3-8B	10	30	28	92	70	40
	50	52	25	300	90	30
	100	170	300	550	1000	70
Llama-3-70B	10	180	400	480	70	40
	50	350	480	2950	490	510
	100	450	6800	6300	6500	750

Table 2: Average Time To First Token (TTFT) in milliseconds under different user loads. Lower values are better.

Model	Metric	vLLM	TRT	lmDeploy	TGI
Llama-3-8B	Throughput ↑	1250	1320	1260	1240
	TTFT ↓	20	50	22	110
	ITL ↓	17	14	13	18
Llama-3-70B	Throughput ↑	720	680	750	745
	TTFT ↓	70	130	65	460
	ITL ↓	43	35	35	45
Mistral-8x7B	Throughput ↑	820	930	790	810
	TTFT ↓	100	100	1450	480
	ITL ↓	39	35	60	39

Table 1: Performance benchmark comparison. Green indicates better performance within a row. Throughput is maximized, while TTFT and ITL are minimized.

6 AI-Tool Usage

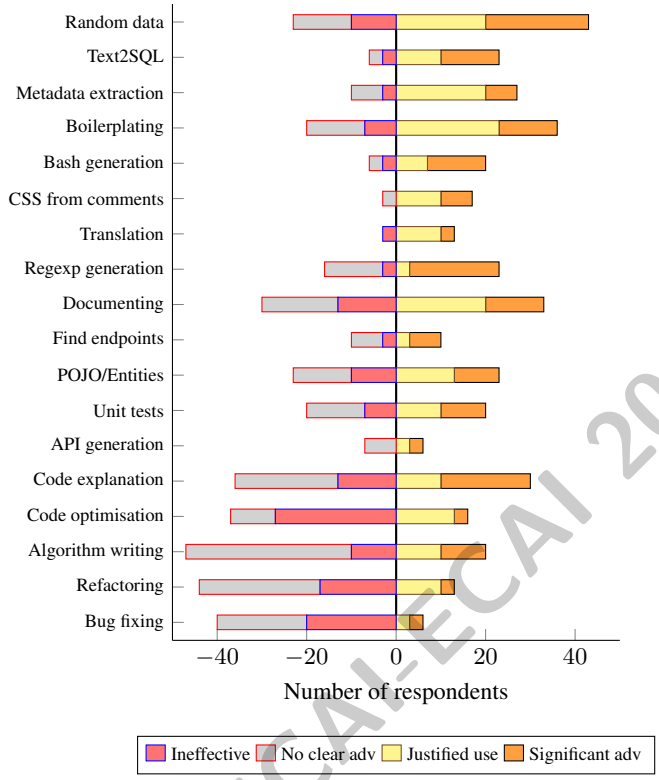

Figure 1: Productivity gains across AI-assisted coding scenarios: percentage improvements by task.

To quantify the impact of the proposed solution, we conducted a comprehensive study involving 100 Java and Python developers over a period of three months. To ensure methodological rigor and mitigate the bias inherent in self-reported data, we employed a hybrid evaluation framework combining qualitative surveys with quantitative Jira software analytics.This hybrid methodology follows principles of adaptive evaluation, where task-specific characteristics inform the assessment strategy [Marina *et al.*, 2025].

We utilized the Jira to extract "Mean Cycle Time" (MCT) – defined as the duration from the "In Progress" status to "Resolved" – for over 2500 individual tasks. Tasks were automatically categorized using Jira labels and issue types. We compared the MCT of these categories against a baseline established during the three months prior to the plugin's deployment, filtering for tasks of similar complexity (Story Points). The AI assistant's ability to perform effectively across diverse coding tasks without per-task fine-tuning is conceptually similar to cross-lingual transfer [Konovalov *et al.*, 2020].

The results, illustrated in Figure 1, reveal that while developers perceived a 50-70% improvement in high-toil activities like **Code Explanation** and **Unit Test Generation**, the objective Jira data showed a more measured 15.4% reduction in overall cycle time. This conservative aggregate accounts for non-coding overhead (meetings, code reviews).

Code Sketching/Boilerplating was the most frequently used (56 cases), delivering a 50% productivity increase for tasks like scaffolding and project setup. **Code Explanation** followed with 66 cases, yielding around 60% productivity improvement in understanding unfamiliar or legacy code. **Code Documenting** (63 cases) and **Random Data Generation** (66 cases) also showed strong results, each achieving roughly 70% productivity gains. **Regexp Generation** (39 cases) and **Bash Generation** (26 cases) provided moderate benefits, with productivity improvements of 40% and 20%, respectively. Less frequently used scenarios like **Code Translation** (16 cases) and **Text2SQL** (29 cases) offered significant productivity improvements of approximately 70% and 50% respectively, valuable for multilingual code-bases and database queries. **CSS** from comments (20 cases) resulted in a 30% gain, useful for rapid prototyping.

Considering that developers also spend significant time on design, requirement analysis, collaboration, and debugging, the overall productivity increase attributed to AI-assistant usage was conservatively estimated at 15%. AI-assistant provides the strongest productivity boosts for rapid prototyping or repetitive tasks. Tasks involving code explanation and documentation significantly benefit from generative AI, facilitating faster onboarding and knowledge transfer.

7 Conclusion

This paper demonstrates that enterprise-grade, privacy-preserving AI code assistance is not only feasible but also practical. We introduced an on-premise IDE plugin architecture that combines client-side tiered retrieval (Personal CodeRAG) with vLLM-optimized inference on dedicated hardware, ensuring that no proprietary code leaves the organization. Our benchmark of four inference frameworks showed that vLLM and LMDeploy achieve the lowest time-to-first-token (≈20 ms for LLaMA-8B), making them suitable for real-time developer interactions. In a three-month study with 100 developers, the plugin reduced mean cycle time by 15.4%, with particularly strong perceived productivity gains (50–70%) in tasks such as code explanation, documentation, and unit test generation.

References

[Aushev *et al.*, 2025] Islam Aushev, Egor Kratkov, Evgenii Nikolaev, Andrei Glinskii, Vasilii Krikunov, Alexander Panchenko, Vasily Konovalov, and Julia Belikova. RAGulator: Effective RAG for regulatory question answering. In Tuba Gokhan, Kexin Wang, Iryna Gurevych, and Ted Briscoe, editors, *Proceedings of the 1st Regulatory NLP Workshop (RegNLP 2025)*, pages 114–120, Abu Dhabi, UAE, January 2025. Association for Computational Linguistics.

[Dettmers *et al.*, 2023] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

[Feng *et al.*, 2023] Siyuan Feng, Bohan Hou, Hongyi Jin, Wuwei Lin, Junru Shao, Ruihang Lai, Zihao Ye, Lianmin Zheng, Cody Hao Yu, Yong Yu, and Tianqi Chen. Tensorir: An abstraction for automatic tensorized program optimization. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS 2023, page 804–817, New York, NY, USA, 2023. Association for Computing Machinery.

[Jeong, 2023] Cheonsu Jeong. A study on the implementation of generative ai services using an enterprise data-based llm application architecture. *Advances in Artificial Intelligence and Machine Learning*, 2023.

[Johnson *et al.*, 2021] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *IEEE Trans. Big Data*, 7(3):535–547, 2021.

[Joshi *et al.*, 2024] Pankaj Joshi, Aditya Gupta, Pankaj Kumar, and Manas Sisodia. Robust multi model rag pipeline for documents containing text, table & images. *Proceedings of the 3rd International Conference on Applied Artificial Intelligence and Computing, ICAAIC 2024*, pages 993–999, 2024.

[Kalliamvakou, 2022] Eirini Kalliamvakou. Research: quantifying github copilot's impact on developer productivity and happiness. *GitHub Blog*, 2022.

[Karakurt and Akbulut, 2026] Ehlullah Karakurt and Akhan Akbulut. Retrieval-augmented generation (rag) and large language models (llms) for enterprise knowledge management and document automation: A systematic literature review. *Applied Sciences*, 16(1), 2026.

[Konovalov *et al.*, 2020] Vasily Konovalov, Pavel Gulyaev, Alexey Sorokin, Yury Kuratov, and Mikhail Burtsev. Exploring the bert cross-lingual transfer for reading comprehension. In *Computational Linguistics and Intellectual Technologies*, pages 445–453, 2020.

[Marina *et al.*, 2025] Maria Marina, Nikolay Ivanov, Sergey Pletenev, Mikhail Salnikov, Daria Galimzianova, Nikita Krayko, Vasily Konovalov, Alexander Panchenko, and Viktor Moskvoretskii. LLM-independent adaptive RAG: Let the question speak for itself. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, editors, *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 8697–8709, Suzhou, China, November 2025. Association for Computational Linguistics.

[Microsoft, 2024] Microsoft. Rag patterns and use cases for software development. https://www.microsoft.com and related engineering blogs, 2024. Accessed 2025-12-02.

[Nikolaev *et al.*, 2025] Evgenii Nikolaev, Ivan Bondarenko, Islam Aushev, Vasilii Krikunov, Andrei Glinskii, Vasily Konovalov, and Julia Belikova. FactDebug at SemEval-2025 task 7: Hybrid retrieval pipeline for identifying previously fact-checked claims across multiple languages. In Sara Rosenthal, Aiala Rosá, Debanjan Ghosh, and Marcos Zampieri, editors, *Proceedings of the 19th International Workshop on Semantic Evaluation (SemEval-2025)*, pages 2190–2196, Vienna, Austria, July 2025. Association for Computational Linguistics.

[Pearce *et al.*, 2025] Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. Asleep at the keyboard? assessing the security of github copilot's code contributions. *Commun. ACM*, 68(2):96–105, 2025.

[Peña *et al.*, 2022] Francisco J. Peña, Angel Luis Gonzalez, Sepideh Pashami, Ahmad Al-Shishtawy, and Amir Hossein Payberah. Siambert: Siamese bert-based code search. In *Swedish Artificial Intelligence Society Workshop, SAIS 2022, Stockholm, Sweden, June 13-14, 2022*, pages 1–7. IEEE, 2022.

[Peng *et al.*, 2023] Sida Peng, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer. The impact of AI on developer productivity: Evidence from github copilot. *CoRR*, abs/2302.06590, 2023.

[Richards and Ford, 2019] Mark Richards and Neal Ford. *Fundamentals of Software Architecture: An Engineering Approach*. O'Reilly Media, 2019. Discusses the C4 model as an emerging standard for diagramming software architecture.

[Sawarkar *et al.*, 2024] Kunal Sawarkar, Abhilasha Mangal, and Shivam Raj Solanki. Blended rag: Improving rag (retriever-augmented generation) accuracy with semantic search and hybrid query-based retrievers. *2024 IEEE 7th International Conference on Multimedia Information Processing and Retrieval (MIPR)*, pages 155–161, 2024.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

[Yadav and Chintala, 2025] Agatsya Yadav and Bhargavi Renta Chintala. Optimizing llms using quantization for mobile execution. *CoRR*, abs/2512.06490, 2025.

[Zhang *et al.*, 2024] Dejiao Zhang, Wasi Uddin Ahmad, Ming Tan, Hantian Ding, Ramesh Nallapati, Dan Roth, Xiaofei Ma, and Bing Xiang. CODE REPRESENTATION LEARNING AT SCALE. In *The Twelfth International Conference on Learning Representations*, 2024.

[Ziegler *et al.*, 2022] Albert Ziegler, Eirini Kalliamvakou, X. Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, and Edward Aftandilian. Productivity assessment of neural code completion. In Swarat Chaudhuri and Charles Sutton, editors, *MAPS@PLDI 2022: 6th ACM SIGPLAN International Symposium on Machine Programming, San Diego, CA, USA, 13 June 2022*, pages 21–29. ACM, 2022.