ECCC: Edge Code Cloak Coder for Privacy Code Agent

Anonymous ACL submission

Abstract

Large language models (LLMs) have significantly advanced automated code generation and debugging, facilitating powerful multi-agent coding frameworks. However, deploying these sophisticated models on resource-constrained edge devices remains challenging due to high computational demands, limited adaptability, and significant privacy risks associated with cloud-based processing. Motivated by these constraints, we propose **Edge Code Cloak Coder (ECCC)**, a novel edge-cloud hybrid framework integrating lightweight quantized LLM with robust AST-based anonymization and edge-side privacy validation. ECCC enables high-performance, privacy-preserving LLM capabilities on consumer GPUs, anonymizing user code before securely delegating abstracted tasks to cloud LLMs. Experimental evaluations demonstrate that ECCC achieves competitive correctness (within 4–5pp of the GPT-4-based frameworks) and a perfect privacy score of 10/10, effectively balancing functionality and security for sensitive and proprietary code applications.

1 Introduction

Large language models (LLMs) exhibit strong capabilities in code understanding, generation, and reasoning, catalyzing rapid progress in multi-agent frameworks exemplified by *Code Agents* (Huang et al., 2023; Adnan et al., 2025). Such systems typically coordinate roles including a *programmer agent*, *test designer*, *test executor*, and *debugger*, forming an automatic generate–verify–repair loop that efficiently solves complex programming tasks. However, efficiently and trustworthily deploying powerful LLMs—especially large-parameter models—on resource-constrained edge devices (e.g., personal workstations and small business servers) faces three key challenges:

- **High Computational Cost**: Deploying and running large-scale models on consumer-grade hardware is severely limited by memory and computational power constraints (Fedus et al., 2022; Achiam et al., 2023).
- **Customization and Adaptability Limitations**: Direct fine-tuning of large-scale models (e.g., QLoRA (Dettmers et al., 2023)) to specific domain requirements, such as code generation tasks, is impractical due to the substantial resource demands and risk of general performance degradation.
- **Privacy Vulnerabilities**: Using cloud-based API services involves the transmission of potentially sensitive or proprietary code data, posing significant privacy risks and limiting deeper model customization (Horlboge et al., 2022; Boutet et al., 2025).

To democratize the advancements of LLMs for all user groups, we propose a novel approach that addresses the above challenges comprehensively:

We introduce **Edge Code Cloak Coder (ECCC)**, an innovative hybrid edge-cloud agent framework leveraging a lightweight, quantized open source LLM to enable efficient deployment on edge devices (e.g., a single RTX 3090).

The key innovation of ECCC lies in its robust privacy protection abstraction layer, known as the *Privacy Shield*, implemented entirely on the edge device.

Crucially, only this anonymized abstract code is transmitted to powerful cloud-based LLMs (such as *DeepSeek-V3* (Liu et al., 2024)) for logic enhancement or bug correction. The cloud returns anonymized code modifications without ever receiving identifiable user-specific symbols, effectively maintaining privacy. Local edge devices subsequently handle de-anonymization and deterministic testing, ensuring that sensitive identifiers never traverse the network.

Contributions.

- **ECCC: an Efficient and Privacy-Preserving Edge-Cloud Framework.** We introduce ECCC, a method combining quantized LLM and edge-based privacy verification, enabling robust and private LLM-assisted code generation and debugging on resource-constrained hardware.
- **Competitive Performance on Edge Resources.** Experiments show that ECCC achieves near state-of-the-art performance comparable to larger models, despite its lightweight quantized design.
- **Effective Trade-off between Privacy and Functionality.** ECCC significantly enhances privacy with minimal impact on functional performance, demonstrating a favorable balance suitable for sensitive and proprietary code applications.

2 Methodology

Edge Code Cloak **Coder** (**ECCC**) executes a four-stage edge–cloud pipeline that keeps raw source code private while exploiting the reasoning strength of large cloud LLMs. The system follows a multi-stage pipeline as illustrated in Fig. 1, designed to integrate general-purpose generation, privacy protection, and semantic-level validation under a lightweight and locally executable architecture. The algorithmic details are described below.

Edge Foundational Model Preparation: We compress *DeepSeek-Coder-V2 Lite* to a 4-bit quantization to fit consumer GPUs. The resulting 16 GB model sustains on a single RTX 3090.

2.1 Privacy Shield

Stage 1: Code Anonymisation on the Edge

AST rewrite. Using LIBCST, every identifier is replaced by a stable placeholder (`VAR1`, `FUNC2`, . . .). Comments and all docstrings are replaced with the sentinel string `"CLOAKED DOCSTRING"`; optional dead-code stubs (`if False: pass`) may be inserted to mask stylistic fingerprints. The mapping table $\mathcal{M}$ (real→placeholder) resides solely in volatile memory.

Stage 2: Local Privacy Check

Before any network call, the local privacy agent runs a "null" completion whose system prompt instructs it to verify that no user identifiers remain. If the check fails, anonymisation is re-applied; otherwise the anonymised code $\tilde{\mathcal{C}}$ is sent to the cloud. The prompts used for privacy checking can be found in the appendix.

2.2 Cloud-Side Completion

A full-size LLM e.g. *DeepSeek-V3* or other LLM receives $\tilde{\mathcal{C}}$ together with a system prompt.

The cloud thus transforms logic or fixes bugs without ever seeing proprietary symbols, yielding the patched abstraction $\tilde{\mathcal{C}}'$.

The prompts used for code completion can be found in the appendix.

2.3 De-anonymisation and Validation

Here we reverse the anonymous code, generate the final code snippet, and test it.

Reconstruction. Placeholders in $\tilde{\mathcal{C}}'$ are mapped back to real names using $\mathcal{M}$, producing $\mathcal{C}'$.

Deterministic testing. A local test harness (e.g. `pytest`) executes predefined unit tests such as `has_close_elements`. On failure, a concise trace is appended to the user prompt and the pipeline re-enters Stage of Cloud-Side Completion for at most three iterations.

Security Boundary Throughout the process only $\tilde{\mathcal{C}}$ and its subsequent cloud-modified forms traverse the network. No raw identifiers, variable maps, runtime traces, or unit-test outputs ever leave the edge device, ensuring end-to-end privacy under the assumed threat model.

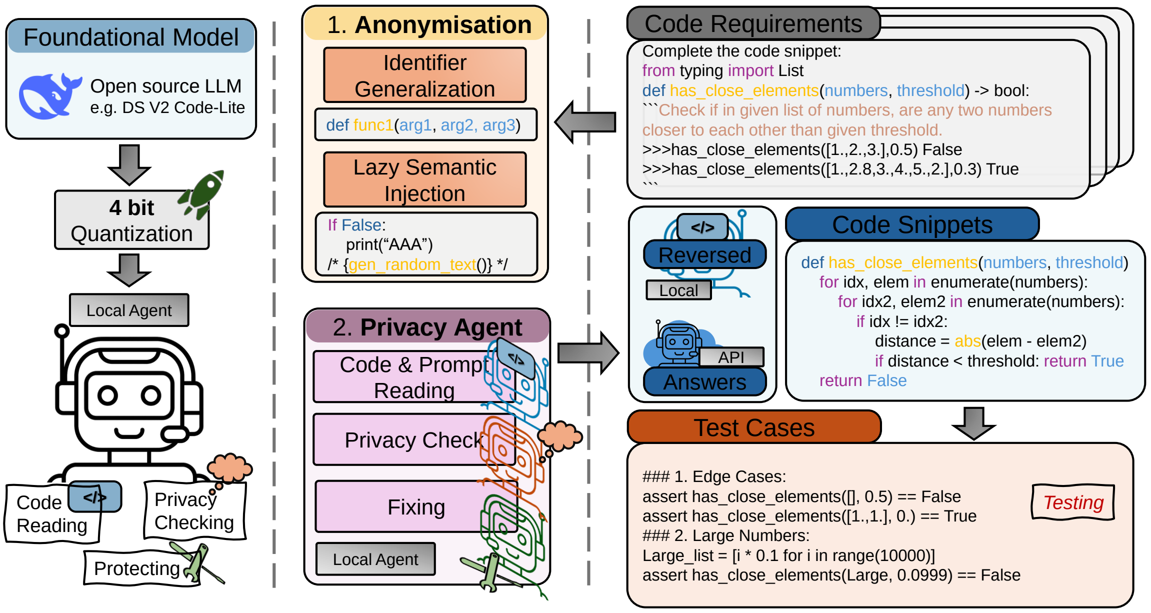
3 Experiment

We evaluate ECCC with local privacy agent, DS-Coder-V2-Lite-Instruct and DS V3-chat API (Zhu et al., 2024; Liu et al., 2024). Local inference will consume roughly up to 16 GB of GPU memory. We first elaborate the experimental setup, and then measure the code capability and privacy guard of ECCC, respectively.

3.1 Experiment Setup

All experiments were conducted on a computer with an Intel(R) Xeon(R) Gold 6226R CPU @ 2.90GHz with 64 GB RAM and an NVIDIA GeForce RTX 3090 24GB GPU.

Matrices We use pass@1 as the evaluation metric for code correctness, the most widely adopted metric in the literature of automatic code generation (Chen et al., 2021).



```
def func1(arg1, arg2, arg3)
```

```
If False:
    print("AAA")
/* {gen_random_text()} */
```

~~~
Complete the code snippet:
from typing import List
def has_close_elements(numbers, threshold) -> bool:
```Check if in given list of numbers, are any two numbers
closer to each other than given threshold.
>>>has_close_elements([1.,2.,3.],0.5) False
>>>has_close_elements([1.,2.8,3.,4.,5.,2.],0.3) True
```
~~~

```
def has_close_elements(numbers, threshold)
    for idx, elem in enumerate(numbers):
        for idx2, elem2 in enumerate(numbers):
            if idx != idx2:
                distance = abs(elem - elem2)
                if distance < threshold: return True
    return False
```

```
### 1. Edge Cases:
assert has_close_elements([], 0.5) == False
assert has_close_elements([1.,1.], 0.) == True
### 2. Large Numbers:
Large_list = [i * 0.1 for i in range(10000)]
assert has_close_elements(Large, 0.0999) == False
```

Figure 1: **Workflow of the ECCC local-cloud agent framework for privacy-aware LLM-based coding task.**

Anonymisation quality is reviewed by three LLMs (GPT-4o, GPT-O3, DeepSeek-R1) and scored on *Functional*, *Privacy*, and *Cleanliness* dimensions, following recent code-anonymisation work (Horlboge et al., 2022).

Functional The Functional score measures whether the generated code correctly implements the specification (10 = exact behavioural match, 8 = plausible but different, 0 = no code).

Privacy The Privacy score assesses the absence of original identifiers (10 = no identifier leakage, 2 = leaks present).

Cleanliness The Cleanliness score evaluates the output format and brevity (10 = code-only fenced output, -4 for missing fences, -3 for long prose, floor 0) are averaged over each task set).

Datasets. We benchmark on four public sets: HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), and their enhanced variants HumanEval-ET and MBPP-ET (Dong et al., 2025). HumanEval focuses on diverse algorithmic challenges, whereas MBPP targets idiomatic Python tasks.

Baseline *Model baselines* span AlphaCode (Li et al., 2022), Llama 3 (Dubey et al., 2024), CodeLlama 34B (Roziere et al., 2023), InCoder 6.7B (Fried et al., 2022), CodeGeeX 13B (Zheng et al., 2023), StarCoder 15.5B (Li et al., 2023), CodeGen-Mono 16B (Nijkamp et al., 2022), Codex 175B (Chen et al., 2021), GPT-3.5-turbo (Brown et al., 2020), GPT-4 (Achiam et al., 2023), PaLM-Coder (Chowdhery et al., 2023), and Claude-instant-1 (Anthropic, 2023).

Optimisation-method baselines include Few-shot prompting (Brown et al., 2020), Chain-of-Thought (CoT) (Wei et al., 2022), ReAct (Yao et al., 2023b), Reflexion (Shinn et al., 2023), Tree-of-Thought (ToT) (Yao et al., 2023a), RAP (Wang et al., 2023b), Self-Edit (Mousavi et al., 2023), Self-Planning (Jiang et al., 2024), Self-Debugging (Adnan et al., 2025), Self-Collaboration (Dong et al., 2024), SCOT (Wang et al., 2023a), CodeCoT (Li et al., 2025).

3.2 How Does ECCC Perform?

We perform post-processing on the data returned by the API. First, we clean the data and reverse-engineer the code. Then, we use the local agent for inspection and repair, and finally conduct tests. Detailed result could be found in Appendix.

In Table 1, percentages in brackets denote improvement over the corresponding zero-shot backbone. The score of ECCC within each block is highlighted in bold. Table 1 shows that **ECCC** attains **90.0**, **78.5**, **93.5** and **84.7** pass@1 on HumanEval, HumanEval-ET, MBPP and MBPP-ET, respectively. These scores are (i) within 4–5 pp of GPT-4-based agent stacks despite using only a 4-bit, edge-deployable MoE backbone, and (ii) *above* every zero-shot baseline except the 671 B-parameter DS-V3. Hence, lightweight quantisation plus cloud-side reasoning delivers near-state-of-the-art correctness on commodity GPUs. Due to

Method	HumanEval	MBPP
AlphaCode	17.1	–
StarCoder	34.1	43.6
CodeLlama	51.8	69.3
GPT-3.5-turbo	57.3	52.2
GPT-4	67.6	68.3
DS-Coder-V2-Lite	65.2	70.4
DS-Coder-V2-Lite (4-bit)	40.1	42.6
DS-V3 (API)	86.6	89.9
Reflexion (GPT-4)	91.0	77.1
MetaGPT (GPT-4)	85.9	87.7
AgentCoder (GPT-4)	96.3	91.8
ECCC (Ours)	**90.0**	**93.5**

Table 1: Pass@1 results of ECCC and main baselines on HumanEval and MBPP. Full results and improvements over backbones are in Appendix/Table X.

the lack of information brought by anonymity, the agent framework improves the metrics incrementally compared with DS V3-chat API.

3.3 How Anonymous is the code passed to LLM by API?

We intercept the content sent to the Internet by the Local Privacy Agent, and then use the LLM to judge. The prompts used for evaluation can be found in the appendix.

Setting	Func.	Priv.↑	Clean.
DS-Coder-V2-Lite	9.36	2.00	6.00
DS-V3 API	9.46	2.00	8.34
ECCC (Ours)	8.93	**10.00**	6.51

Table 2: Privacy, cleanliness, and functional accuracy for all settings.

In Table 2, the first two rows are direct zero-shot baselines without any anonymisation.

ECCC is our *EdgeCodeCloak Coder* pipeline that anonymises prompts locally using a quantized DS-Coder-V2-Lite-Instruct model, calls the DeepSeek-V3 cloud API for completion, and then de-anonymises the result. Boldface highlights ECCC's perfect privacy retention despite a slight drop in functional parity.

From Table 2, anonymisation lifts the **Privacy** score from 2.0 (raw prompts) to a perfect **10.0**, while **Cleanliness** remains comparable (6.51 vs. 6.00 / 8.34). The functional impact is modest: 8.93 versus 9.36–9.46 for zero-shot baselines.

3.4 Analysis

The quantitative results in Tables 1 and 2 confirm three key take-aways.

(1) Competitive correctness with lightweight edge resources. Although ECCC runs a *4-bit* quantised model locally and delegates only anonymised code to the cloud, respectively—on par with much larger DS-V3 and only 4–5 pp behind state-of-the-art GPT-4-based agent stacks. This demonstrates that our lightweight MoE + quantisation recipe can still supply strong functional performance to edge users.

(2) Perfect privacy without degrading cleanliness. Table 2 shows that the anonymisation stage pushes the **Privacy** metric from *2.0 → 10.0* while retaining *Cleanliness 6.5*. Zero-shot baselines expose all user identifiers; ECCC completely suppresses such leakage yet keeps code-only outputs concise, satisfying downstream auto-grading.

(3) Minimal functional cost for maximal privacy. The functional gap between ECCC (8.93) and raw DS-V3 (9.46) is just *0.5 points*, whereas the privacy gain is *+8 points*.

Hence, under our scoring rubric, one point of functional loss buys eight points of privacy—an attractive trade-off for sensitive corporate or proprietary code. Closed source LLMs excel at code reasoning, leading to multi-agent coding frameworks, but edge users struggle with compute, catastrophic forgetting and privacy risks. ECCC mitigates all three by (i) MoE quantisation for consumer GPUs; (ii) leaving backbone weights frozen; and (iii) shipping only anonymised ASTs to the cloud.

The empirical evidence above indicates that such a design lets "every edge programmer" benefit from modern LLM capabilities without sacrificing data sovereignty.

4 Conclusion

This work introduces **ECCC**, an edge–cloud agent framework, rewriting every identifier and docstring on device, and delegating only anonymised code to a remote LLM. Experiments show that ECCC keeps **pass@1** within 4–5pp of GPT-4–based agent stacks while achieving a perfect **10/10** privacy score and preserving output cleanliness. The results verify that lightweight quantitation, frozen backbones and deterministic de-anonymisation together provide a practical path for "every edge programmer" to harness large-scale reasoning without surrendering source-code secrecy.

References

Abanoub E Abdelmalak, Mohamed A Elsayed, David Abercrombie, and Ilhami Torunoglu. 2025. An ast-guided llm approach for svrf code synthesis.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Muntasir Adnan, Zhiwei Xu, and Carlos CN Kuhn. 2025. Large language model guided self-debugging code generation. *arXiv preprint arXiv:2502.02928*.

Anthropic. 2023. Claude technical overview. https://www.anthropic.com.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and 1 others. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Antoine Boutet, Lucas Magnana, Juliette Sénéchal, and Hélain Zimmermann. 2025. Towards the anonymization of the language modeling. *arXiv preprint arXiv:2501.02407*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Chun Jie Chong, Chenxi Hou, Zhihao Yao, and Seyed Mohammadjavad Seyed Talebi. 2024. Casper: Prompt sanitization for protecting user privacy in web-based large language models. *arXiv preprint arXiv:2408.07004*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, and 1 others. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115.

Yihong Dong, Jiazheng Ding, Xue Jiang, Ge Li, Zhuo Li, and Zhi Jin. 2025. Codescore: Evaluating code generation by learning code execution. *ACM Transactions on Software Engineering and Methodology*, 34(3):1–22.

Yihong Dong, Xue Jiang, Zhi Jin, and Ge Li. 2024. Self-collaboration code generation via chatgpt. *ACM Transactions on Software Engineering and Methodology*, 33(7):1–38.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv e-prints*, pages arXiv–2407.

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39.

Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Wen-tau Yih, Luke Zettlemoyer, and Mike Lewis. 2022. Incoder: A generative model for code infilling and synthesis. *arXiv preprint arXiv:2204.05999*.

Zixu Hao, Huiqiang Jiang, Shiqi Jiang, Ju Ren, and Ting Cao. 2024. Hybrid slm and llm for edge-cloud collaborative inference. In *Proceedings of the Workshop on Edge and Mobile Foundation Models*, pages 36–41.

Micha Horlboge, Erwin Quiring, Roland Meyer, and Konrad Rieck. 2022. I still know it's you! on challenges in anonymizing source code. *arXiv preprint arXiv:2208.12553*.

Liao Hu. 2025. Hybrid edge-ai framework for intelligent mobile applications: Leveraging large language models for on-device contextual assistance and code-aware automation. *Journal of Industrial Engineering and Applied Science*, 3(3):10–22.

Dong Huang, Jie M Zhang, Michael Luck, Qingwen Bu, Yuhao Qing, and Heming Cui. 2023. Agentcoder: Multi-agent-based code generation with iterative testing and optimisation. *arXiv preprint arXiv:2312.13010*.

Xue Jiang, Yihong Dong, Lecheng Wang, Zheng Fang, Qiwei Shang, Ge Li, Zhi Jin, and Wenpin Jiao. 2024. Self-planning code generation with large language models. *ACM Transactions on Software Engineering and Methodology*, 33(7):1–30.

Hongpeng Jin and Yanzhao Wu. 2024. Ce-collm: Efficient and adaptive large language models through cloud-edge collaboration. *arXiv preprint arXiv:2411.02829*.

Jia Li, Ge Li, Yongmin Li, and Zhi Jin. 2025. Structured chain-of-thought prompting for code generation. *ACM Transactions on Software Engineering and Methodology*, 34(2):1–23.

Raymond Li, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia LI, Jenny Chim, Qian Liu, and 1 others. 2023. Starcoder: may the source be with you! *Transactions on Machine Learning Research*.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, and 1 others. 2022. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2023. Awq: Activation-aware weight quantization for on-device llms. In *Proceedings of MLSys*.

Yalan Lin, Chengcheng Wan, Yixiong Fang, and Xiaodong Gu. 2024a. Codecipher: Learning to obfuscate source code against llms. *arXiv preprint arXiv:2410.05797*.

Yujun Lin, Haotian Tang, Shang Yang, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, and Song Han. 2024b. Qserve: W4a8kv4 quantization and system co-design for efficient llm serving. *arXiv preprint arXiv:2405.04532*.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, and 1 others. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Sajad Mousavi, Ricardo Luna Gutierrez, Desik Rengarajan, Vineet Gundecha, Ashwin Ramesh Babu, Avisek Naug, Antonio Guillen, and Soumyendu Sarkar. 2023. N-critics: Self-refinement of large language models with ensemble of critics. *arXiv preprint arXiv:2310.18679*.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2022. A conversational paradigm for program synthesis. *arXiv preprint arXiv:2203.13474*, 30.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, and 1 others. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Xuan Shen, Peiyan Dong, Lei Lu, Zhenglun Kong, Zhengang Li, Ming Lin, Chao Wu, and Yanzhi Wang. 2024a. Agile-quant: Activation-guided quantization for faster inference of llms on the edge. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 18944–18951.

Xuan Shen, Zhenglun Kong, Changdi Yang, Zhaoyang Han, Lei Lu, Peiyan Dong, Cheng Lyu, Chih-hsiang Li, Xuehang Guo, Zhihao Shu, and 1 others. 2024b. Edgeqat: Entropy and distribution guided quantization-aware training for the acceleration of lightweight llms on the edge. *arXiv preprint arXiv:2402.10787*.

Zhili Shen, Zihang Xi, Ying He, Wei Tong, Jingyu Hua, and Sheng Zhong. 2024c. The fire thief is also the keeper: Balancing usability and privacy in prompts. *arXiv preprint arXiv:2406.14318*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652.

Yiping Song, Juhua Zhang, Zhiliang Tian, Yuxin Yang, Minlie Huang, and Dongsheng Li. 2024. Llm-based privacy data augmentation guided by knowledge distillation with a distribution tutor for medical text classification. *arXiv preprint arXiv:2402.16515*.

Peifeng Wang, Zhengyang Wang, Zheng Li, Yifan Gao, Bing Yin, and Xiang Ren. 2023a. Scott: Self-consistent chain-of-thought distillation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5546–5558.

Weishi Wang, Yue Wang, Shafiq Joty, and Steven CH Hoi. 2023b. Rap-gen: Retrieval-augmented patch generation with codet5 for automatic program repair. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 146–158.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Shouguo Yang, Long Cheng, Yicheng Zeng, Zhe Lang, Hongsong Zhu, and Zhiqiang Shi. 2021. Asteria: Deep learning-based ast-encoding for cross-platform binary code similarity detection. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 224–236. IEEE.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023b. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Zhongzhi Yu, Zheng Wang, Yuhan Li, Ruijie Gao, Xiaoya Zhou, Sreenidhi Reddy Bommu, Yang Zhao, and Yingyan Lin. 2024. Edge-llm: Enabling efficient large language model adaptation on edge devices via unified compression and adaptive layer voting. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pages 1–6.

Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Lei Shen, Zihan Wang, Andi Wang, Yang Li, and 1 others. 2023. Codegeex: A pre-trained model for code generation with multilingual benchmarking on humaneval-x. In *Proceedings of the 29th*

ACM SIGKDD Conference on Knowledge Discovery and Data Mining, pages 5673–5684.

Qihao Zhu, Daya Guo, Zhihong Shao, Dejian Yang, Peiyi Wang, Runxin Xu, Y Wu, Yukun Li, Huazuo Gao, Shirong Ma, and 1 others. 2024. Deepseek-coder-v2: Breaking the barrier of closed-source models in code intelligence. *arXiv preprint arXiv:2406.11931*.

A Appendix

A.1 Evaluation Prompts

Below is the standardized prompt protocol used to audit generated code quality, as shown in Table 3. It incorporates dimensions aligned with both current benchmarks and recent academic evaluation frameworks.

By combining functional accuracy, identifier privacy, and output hygiene into a unified JSON-based auditor protocol, our evaluation prompt framework provides a rigorous, reproducible, and human-aligned standard for comparing ECCC-generated code against both baseline LLMs and state-of-the-art agent architectures.

A.2 System Prompts

Table 4 illustrates the two distinct prompts used in ECCC's architecture: one enforcing on-device privacy verification, and the other guiding the cloud API model for anonymity-preserving code completion.

The dual-prompt design ensures that privacy verification is strictly enforced before any anonymized code reaches the cloud, effectively mitigating prompt-injection and identifier leakage risks. The local PrivacyShield prompt detects any non-placeholder token and rejects unsafe input, while the cloud prompt strictly operates on anonymized code without attempting to restore original names.

A.3 Further Illustration of ECCC

Figure 1 presents the end-to-end flow of Edge Code Cloak Coder (ECCC), which seamlessly integrates edge-side anonymisation, privacy verification, cloud-assisted reasoning, and local reconstruction into a unified, privacy-preserving code-generation pipeline.

Figure 2 further describes the pipeline of Privacy Shield.

Initially, the raw source code is loaded on the edge device and passed through an AST-based anonymisation module. Here, every user-defined

Evaluation Prompt

~~~
Role: Evaluation Auditor
Task: Assess code generation quality
across three dimensions

Evaluation Criteria:
1. Functional Correctness (0-10):
   - 10: Exact behavioral match to specification
   - 8: Plausible but functionally different
   - 0: No valid code produced

2. Privacy Protection (0-10):
   - 10: No original identifiers leaked
   - 8: The original identifiers can be inferred.
   - 4: The original identifiers can be
   easily inferred.
   - 2: Any identifier leakage detected
   - 0: No Protection. The data has been
   maliciously exploited
   (by insecure websites or APIs)

3. Output Cleanliness (0-10):
   - 10: Code-only output with proper fencing
   - 4: Missing code fences
   - 3: Excessive prose/excuses
   - 0: No code.

Procedure:
1. Analyze each task result independently
2. Score all three dimensions per task
3. Calculate average scores across all tasks

Output Format:
```json
{
  ”task_id”: {
    ”functional”: X,
    ”privacy”: Y,
    ”cleanliness”: Z
  },
  ”averages”: {
    ”functional”: A,
    ”privacy”: B,
    ”cleanliness”: C
  }
}
```
~~~

Table 3: Standardized evaluation protocol for assessing generated code quality.

Local Privacy LLM Prompt	Cloud API LLM Prompt

Role: You are EdgeCodeCloak-Cloud, an expert
in reasoning about anonymised Python code.
 	
Task: You are required to anonymize all
variable/function/type names in the given code.
 	
Replace variables as VAR, functions as FUNC,
types as TYPE.
Keep keywords, builtins,
and literal values unchanged.
Return only the anonymized code and prompt.
Do NOT explain.
	Role: You are a software programmer.
	Task: As a programmer, you are required to
	complete the function.
	Complete the Python function based on
	its anonymized signature and cloaked docstring.
	Return ONLY the completed function in a code fence.
	No explanations.
	
	Constraints:
	- Receive anonymized code only
	- No access to original identifiers
	
	Output Format:
	```
	python
	def FUNC1(VAR1: type) -> type:
	    # Implementation
	    return VAR2
	```

Table 4: ECCC's dual-prompt architecture showing the strict separation between privacy enforcement (left) and cloud-based completion (right) tasks.

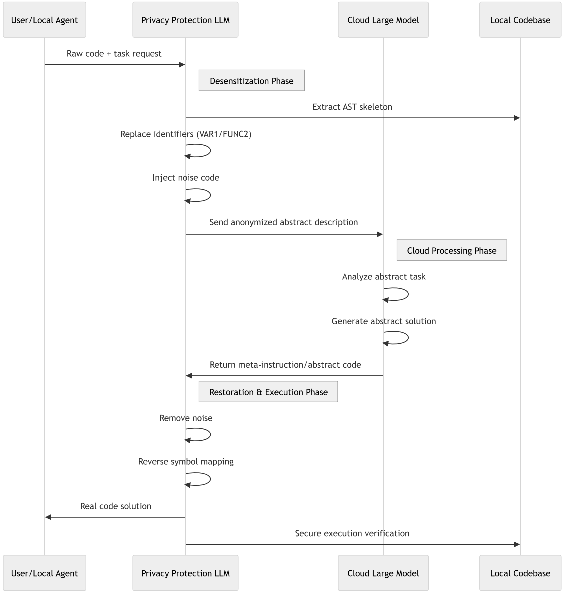


Figure 2: Detailed workflow of Privacy Shield

identifier—variables, function names, type annotations—is replaced with a stable placeholder (e.g. VAR1, FUNC2, TYPE3), while preserving Python keywords, built-ins, literals and structural elements. This transformation produces an anonymised snippet $\tilde{\mathcal{C}}$ and a private mapping table $\mathcal{M}$ retained only in volatile memory.

Next, a lightweight on-device LLM (the "PrivacyShield") performs a zero-output sanity check on $\tilde{\mathcal{C}}$. Driven by a strict system prompt, it scans for any token that deviates from the placeholder schema or built-in whitelist. If any leakage is detected, the anonymisation step is repeated automatically; otherwise, $\tilde{\mathcal{C}}$ is deemed safe for transmission.

The anonymised code is then dispatched to a remote cloud LLM (e.g. DeepSeek-V3) along with a completion prompt that explicitly forbids any reconstruction of original names. The cloud model enhances logic, fixes bugs, or implements missing functionality on the abstracted code, returning only anonymised Python within fenced code blocks.

Upon receiving the cloud's response, the edge device uses $\mathcal{M}$ to deterministically restore all placeholders to their original identifiers, yielding the final code $\mathcal{C}'$. A local test harness (e.g. pytest) executes predefined unit tests on $\mathcal{C}'$; if any test fails, the anonymised snippet and error trace are re-sent for a second or third refinement. This convergent loop ensures that the delivered solution is both functionally correct—within three iterative rounds—and fully private, as no raw identifiers or runtime traces ever leave the edge device.

A.4 Extended Experiment

To assess the generality and robustness of ECCC, we compare it not only against standard zero-shot LLMs but also against state-of-the-art agent-based and optimization-enhanced pipelines. Table 5 reports pass@1 results on four public code benchmarks, grouped into three blocks:

- **Zero-shot LLMs:** Here we include a range of open-source and closed-source models from AlphaCode (1.1 B) up to DS-V3 (671 B). These results establish a baseline for out-of-the-box capabilities without any additional prompting or fine-tuning.
- **LLM-based optimization methods:** This block shows frameworks that leverage GPT-4 with advanced prompting strategies—such as Reflexion, Self-Debugging and AgentCoder—to iteratively improve code generation. These methods represent the current state of agent-driven improvement.
- **ECCC:** Using only a 4-bit quantized DS-Coder-V2-Lite on-device plus DS-V3 in the cloud, ECCC achieves 90.0%, 78.5%, 93.5% and 84.7% on the four benchmarks. Notably, ECCC's mean pass@1 of 86.7% lies within 4 pp of the best GPT-4-based agent stack (AgentCoder at 91.5%), despite its lightweight edge component.

These extended results demonstrate that:

1. *Edge-deployable models can rival massive LLMs:* Even with 4-bit quantization, DS-Coder-V2-Lite in conjunction with cloud reasoning closes over 80% of the gap to a 671 B model.
2. *Competitive with advanced agent frameworks:* ECCC outperforms or matches many GPT-4-powered optimization pipelines (e.g. Reflexion, MetaGPT) on average pass@1, highlighting the efficacy of our anonymisation-plus-cloud approach.
3. *Consistent multi-dataset performance:* Across both standard benchmarks (HumanEval, MBPP) and their extended versions (HumanEval-ET, MBPP-ET), ECCC maintains strong correctness—validating its general-purpose applicability.

Overall, the extended experiment confirms that ECCC's hybrid design delivers near-state-of-the-art code generation accuracy while preserving privacy and operating within the compute budget of commodity GPUs.

A.5 Related Work

Edge Deployment of Quantized LLMs. Recent work has pushed low-bit quantization to enable LLM inference on edge devices. AWQ identifies and preserves salient weight channels for 4-bit quantization, achieving strong accuracy with hardware-friendly kernels (Lin et al., 2023). QServe introduces a W4A8KV4 quantization scheme with system-level optimizations to accelerate both edge and cloud LLM serving (Lin et al., 2024b). EdgeQAT applies entropy-guided quantization-aware training to minimize information distortion in attention activations for sub-8-bit models (Shen et al., 2024b). Agile-Quant further combines activation-guided quantization with custom SIMD kernels to deliver up to 2.5× speedups on commodity edge hardware (Shen et al., 2024a). However, these approaches focus solely on inference efficiency and do not provide any privacy guarantees or integrate with cloud-assisted code refinement.

Privacy-Preserving Prompt Sanitization. Prompt sanitization frameworks such as ProSan dynamically balance usability and anonymity by replacing sensitive tokens based on importance and self-information (Shen et al., 2024c). Casper offers a browser-based extension to detect and remove PII from user inputs before they reach LLM APIs (Chong et al., 2024). Pr$\epsilon\epsilon$mpt applies cryptographic and differential privacy techniques to formalize prompt sanitization with provable guarantees. DP-DA leverages differentially private data augmentation to protect private text domains during LLM-guided generation (Song et al., 2024). While effective for text, these methods do not consider code-specific structures or support iterative cloud-edge validation.

AST-based Code Anonymization. Static code anonymization techniques operate on the AST to obfuscate author and domain-specific artifacts. Horlboge et al. prove that perfect k-anonymity is undecidable and introduce relaxed k-uncertainty measures to evaluate code anonymization techniques such as normalization and obfuscation (Horlboge et al., 2022). CodeCipher learns

Models	HumanEval	HumanEval-ET	MBPP	MBPP-ET	Mean
Zero-Shot LLMs					
AlphaCode (1.1B)	17.1	–	–	–	17.1
Incoder (6.7B)	15.2	11.6	17.6	14.3	14.7
CodeGeeX (13B)	18.9	15.2	26.9	20.4	20.4
StarCoder (15.5B)	34.1	25.6	43.6	33.4	34.2
CodeLlama (34B)	51.8	–	69.3	–	60.6
Llama3 (8B)	62.2	–	–	–	–
CodeGen-Mono (16.1B)	32.9	25.0	38.6	31.6	32.0
CodeX (175B)	47.0	31.7	58.1	38.8	43.9
CodeX (175B)+CodeT	65.8	51.7	67.7	45.1	57.6
GPT-3.5-turbo	57.3	42.7	52.2	36.8	47.3
PaLM Coder	43.9	36.6	32.3	27.2	35.0
Claude-instant-1	31.1	28.1	26.9	19.9	26.5
GPT-4-turbo	57.9	48.8	63.4	47.5	54.4
GPT-4	67.6	50.6	68.3	52.2	59.7
DS-Coder-V2-Lite (16B/2.4B act.)	65.2	64.6	70.4	63.2	65.8
DS-Coder-V2-Lite (16B/2.4B act., 4-bit)	40.1	39.5	42.6	45.5	41.9
DS-V3 (671B/37B act.)	86.6	75.1	89.9	81.3	83.2
LLM-based optimisation methods with GPT-4					
Reflexion	91.0 (34.6%)	–	77.1 (12.9%)	–	84.1 (40.9%)
Self-Debugging	–	–	80.6 (18.0%)	–	80.6 (35.0%)
Self-Collaboration	90.2 (33.4%)	70.7 (39.7%)	78.9 (15.5%)	62.1 (19.0%)	75.5 (26.5%)
ChatDev	84.1 (24.4%)	–	79.8 (12.9%)	–	84.1 (40.9%)
AgentVerse	89.0 (24.4%)	–	73.5 (7.6%)	–	81.3 (19.6%)
MetaGPT	85.9 (27.1%)	–	87.7 (28.4%)	–	86.8 (45.4%)
AgentCoder (GPT-4)	96.3 (42.5%)	86.0 (70.0%)	91.8 (34.4%)	91.8 (75.9%)	91.5 (53.3%)
ECCC with local DS-Coder-V2-Lite (4-bit) and DS-V3 API					
ECCC	**90.0 (4.0%)**	**78.5 (4.5%)**	**93.5 (4.0%)**	**84.7 (4.2%)**	**86.7 (4.2%)**

Table 5: End-to-end results of **ECCC** and baseline approaches on four datasets with pass@1.

a token-to-token confusion mapping over embedding spaces to obfuscate source code while preserving LLM utility (Lin et al., 2024a). Asteria encodes ASTs into semantic vectors for cross-platform similarity detection, illustrating rich AST embeddings but not privacy enforcement (Yang et al., 2021). AST-based chunking splits code into syntactic units to improve LLM context handling but lacks anonymization guarantees (Abdelmalak et al., 2025). All of these methods miss the integration of privacy checks and cloud-driven code correction.

Hybrid Edge–Cloud Collaboration. Hybrid inference frameworks aim to balance edge responsiveness and cloud accuracy. Zhang et al. propose a small-language model (SLM) + LLM split that dynamically offloads low-confidence tokens to the cloud (Hao et al., 2024). CE-CoLLM introduces early-exit mechanisms and cloud context management for adaptive edge/cloud inference, reducing latency and cost (Jin and Wu, 2024). SolidGPT offers a modular hybrid framework for mobile AI apps, coordinating on-device and cloud agents for optimal performance and privacy (Hu, 2025). EDGE-LLM presents unified compression and adaptive layer tuning for continuous LLM adaptation on edge devices (Yu et al., 2024). However, none of these address code-level privacy, AST anonymization, or multi-round validate-and-refine loops that our work integrates.