Dual-Expert Guided Adaptive Framework for Secure Code Generation

Boyu Wei[1,2], Yurong Wu[1,2], Qiuhong Zhang[1,2], Shuo Zhang[2†], Zhiming Ding[2†]
[1]University of Chinese Academy of Sciences, Beijing, China
[2]Institute of Software, Chinese Academy of Sciences, Beijing, China
Email:{weiboyu22, wuyurong20, zhangqiuhong22}@mails.ucas.ac.cn,
{zhangshuo, zhiming}@iscas.ac.cn

***Abstract*—Large language models are increasingly applied to code-related tasks, raising growing concerns about the safety of their generated code. Existing methods either directly update the model or adjust generation behavior without changing parameters to improve security. The latter approach is more valuable due to stronger scalability but often improves safety at the cost of functional correctness. Inspired by knowledge transfer, we propose Adaptive Secure Code Generation, a framework leveraging two expert models, respectively focused on security and correctness, to collaboratively guide the code generation of target model. We collect representative code repair and instruction data to train the two experts. To achieve a balanced integration of objectives, we further introduce an entropy-based logits fusion strategy that adaptively adjusts expert guidance strength, thereby enhancing the reliability of the generated code. Specifically, our implementation achieves improvements of 6–13% in overall performance.**

***Index Terms*—Large Language Models, Code Generation, Program Security**

I. Introduction

In recent years, large language models (LLMs) have rapidly advanced in software engineering, evolving from function- and file-level tasks to repository-level challenges [1]–[3]. However, with the growing reliance on LLM-generated code to automate software development, security concerns have become increasingly salient. Practitioners frequently struggle to identify latent vulnerabilities in generated code, and in some cases, may simply neglect the potential security hazards. This poses serious risks to enterprise software and user privacy, making secure code generation imperative.

To address this issue, recent research has proposed numerous techniques for secure code generation. One straightforward approach involves fine-tuning models with vulnerable and fixed code pairs [4], [5]. In contrast, alternative approaches avoid modifying model parameters and instead intervene at different stages of the generation process, including before input [6], during inference [7], or after output [8], [9]. These methods typically employ strategies such as collaborative decoding or post-processing to adjust model behavior. However, considering factors such as scalability, training costs, and inference efficiency, decoding-time strategies demonstrate significant potential. However, a persistent limitation is that efforts to inject security knowledge often inadvertently compromise functional correctness, highlighting an imbalance between these two critical objectives.

Considering the above constraints, we propose **A**daptive **S**ecure **C**ode **G**eneration (ASCG), a framework that leverages two expert models to adjust the behavior of the target LLM, with one emphasizing code security and the other functional correctness. To ensure semantic alignment, each expert is selected as the smallest model from the same family as the target LLM and trained via Low-Rank Adaptation (LoRA) [10], guaranteeing both efficiency and scalability. Departing from the probability-based manipulation employed in existing decoding-time strategies such as CoSec [7], ASCG fundamentally intervenes at the logit level during inference. This design aligns with knowledge transfer paradigms to more effectively convey critical signals, such as vulnerability patterns and functional logic, from the experts to the target model. To balance the influence of both experts, we further introduce an entropy-based logit fusion mechanism that adaptively coordinates their contributions, thereby maintaining an good trade-off between security and correctness.

For empirical evaluation, we train the expert models using high-quality dataset, including a code repair corpus [4] and a diverse collection of instruction-tuning samples [11]–[14]. Extensive experiments are conducted on multiple LLMs covering different architectures and scales, and results consistently demonstrate competitive performance across two high quality benchmarks. The results indicate that ASCG achieves average improvements of 11.1% in security and 6.5% in functional correctness, along with an enhancement of approximately 6–13% in the overall reliability of generated code, thereby underscoring its practical significance.

Our main contributions are as follows:

- We propose ASCG, a novel framework for secure code generation that integrates two expert models focused on security and functional correctness. They collaboratively guide the target LLM during decoding, leading to more reliable and trustworthy code generation.
- We introduce an entropy-based logit fusion strategy that adaptively integrates expert knowledge into the target model during inference. This mechanism can achieve

†Corresponding Author.

‡This work was supported by the National Key R&D Program of China (No. 2022YFF0503900), and the Research Project of Institute of Software, Chinese Academy of Sciences (ISCAS-ZD-202401, ISCAS-ZD-202403).

effective knowledge transfer with minimal impact on inference speed.
- We conduct extensive experiments across multiple code LLMs and benchmarks, demonstrating the practicality and effectiveness of the method.

II. Related Work

A. Large Language Model for Code Generation

Driven by advances in model architecture, data scale, and cross-domain generalization, LLMs have rapidly evolved in recent years. In programming tasks, they exhibit strong code understanding and generation capabilities, enabling applications in software development, debugging assistance, and automation tools. A series of code-centric LLMs have emerged, such as CodeGen [15], the first to adopt a pure decoder architecture and achieve leading program generation performance, and Code Llama [16], optimized through long-context fine-tuning (LCFT), thereby achieving improved performance on extended code inputs. More recently, mainstream model families including Claude, DeepSeek, and Qwen [17], [18] have released specialized code generation variants, reflecting an increasingly diverse and specialized code LLM ecosystem.

B. Secure Code Generation

Security is a critical property of code, and the Common Weakness Enumeration (CWE) serves as a widely adopted standard for classifying software vulnerabilities [19]. With the growing use of LLMs in real-world programming, concerns have arisen over the security of their outputs. A systematic assessment [20] showed that about 40% of Copilot's outputs on CWE Top 25 tasks contained vulnerabilities, suggesting that LLMs may replicate unsafe patterns from training data. This has motivated research on secure code generation, including fine-tuning with security-aware objectives [4], [5], collaborative decoding [7], and agent-based frameworks that iteratively refine outputs with security feedback [8], [9].

C. Decoding-time Strategy

Decoding-time methods steer LLM outputs without modifying model parameters. A representative early work, DExperts [21], controlled text generation via expert–anti-expert ensembles on logits. Subsequent approaches including Proxy-tuning [22] and EFT [23] further explored different logit-based decoding-time strategies across common tasks. Moreover, CoSec [7] adopted a collaborative decoding method with simple probability operations, demonstrating the effectiveness of decoding-time strategies in security-sensitive contexts. Building on these insights, we propose a dynamic logits fusion strategy designed for secure code generation.

III. Methodology

In this section, we provide a detailed description of ASCG. As shown in Figure 1, the framework consists of two stages: training and inference. During training, small pre-trained models are finetuned with preference-specific data to obtain security and functional correctness experts. In inference, the frozen target model is guided by experts at each decoding step, with an entropy-based dynamic weighting adaptively fusing their information to generate more reliable code.

A. Security Expert Model Fine-Tuning

Extensive research [4], [7] has demonstrated that fine-tuning on vulnerable–fixed code pairs helps models learn safer code representations. They leveraged code differences before and after vulnerability fixes to guide the model toward security-related contexts. Given its intuitive efficiency, we follow this setting to train the security expert model (hereafter, security expert). Specifically, the fine-tuning process employs a loss function with two parts. The first is the standard cross-entropy loss, which encourages the model to accurately predict tokens directly related to security modifications:

$$\mathcal{L}^{\text{sec}}(\mathbf{x}, \mathbf{m}) = -\sum_{t} m_t \cdot \log P(x_t|\mathbf{x}_{<t}), \tag{1}$$

Here, $\mathbf{x}$ denotes the secure code snippet, i.e., the fixed one, and $\mathbf{m}$ is a binary mask where $m_t = 1$ if token x_t originates from the security fix and 0 otherwise, such that $\mathcal{L}^{\text{sec}}$ encourages the model to attend to fix-related tokens.

The second term of the loss function incorporates KL divergence, also known as relative entropy, to constrain the deviation in prediction distributions between the security expert and its pre-fine-tuning counterpart, defined as follows:

$$\mathcal{L}^{\text{KL}}(\mathbf{x}, \mathbf{m}) = \sum_{t}(1 - m_t)\text{KL}(\mathcal{S}(x_t|\mathbf{x}_{<t})||\mathcal{B}(x_t|\mathbf{x}_{<t})), \tag{2}$$

Where $\mathcal{S}$ and $\mathcal{B}$ denote the prediction distributions of the security expert and its base model prior to fine-tuning, respectively. We integrate KL divergence as a regularization term primarily to prevent the security expert from overfitting to domain-specific patterns, which could otherwise introduce functional errors or abnormal code structures. However, as demonstrated in our experiments, relying solely on this constraint, as in CoSec, is insufficient to ensure functional correctness, indicating a need for stronger guidance signals. This limitation consequently motivates our training of a dedicated utility-oriented expert.

B. Correctness Expert Model Fine-Tuning

Previous studies [24], [25] have shown that instruction fine-tuning can effectively enhance a model's ability to understand user intent, thereby improving downstream task performance. In code generation, related work [26], [27] further indicates that fine-tuning models using instruction-based code data can significantly improve their understanding and reasoning capabilities regarding code logic and semantics. Inspired by these research, we fine-tune an expert model (hereafter, correctness expert) on large-scale, high-quality instruction-based code data to specialize in correct code generation. Specifically, each sample consists of an instruction $\mathbf{i}$ and its correct code output $\mathbf{x}$, and the training objective is the negative log-likelihood loss:

$$\mathcal{L}^{\text{corr}}(\mathbf{x}, \mathbf{i}) = -\sum_{t} \log P(x_t|\mathbf{x}_{<t}, \mathbf{i}). \tag{3}$$

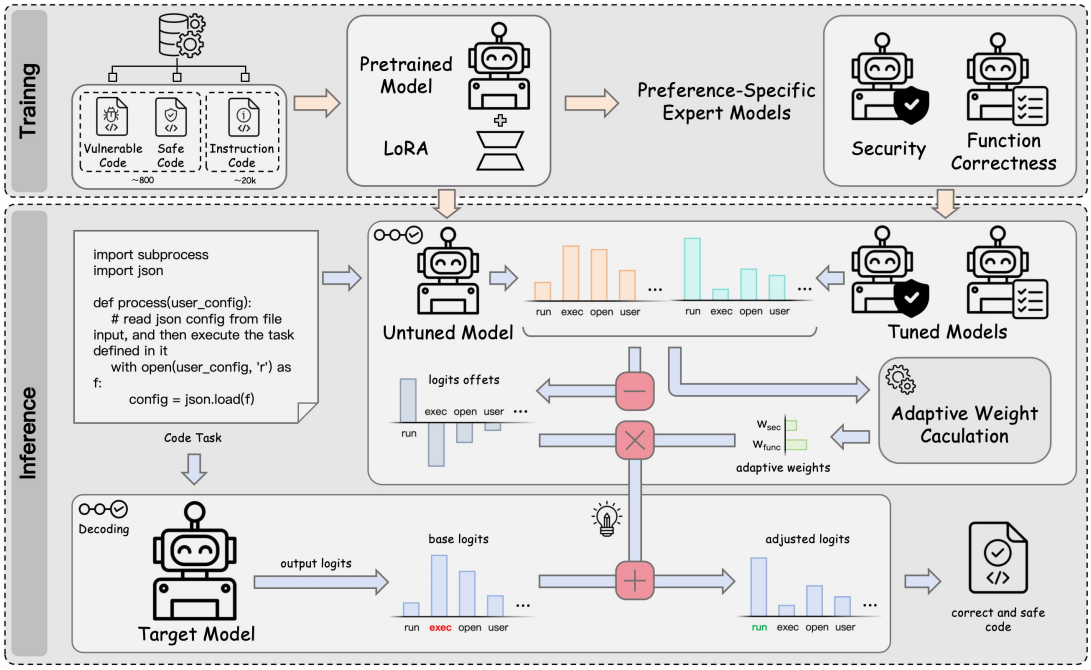

Fig. 1. The overall framework of ASCG consists of two stages: training preference-specific experts and inference with adaptive logits fusion. Here, compared to 'exec', the token 'run' sampled from the adjusted distribution is more reliable.

C. Adaptive Logit Fusion

While logit-based guidance from fine-tuned small models has been utilized in general reasoning tasks [22], [28], its application to the code generation domain remains underexplored. In this work, we extend this paradigm to the coding domain, investigating its potential to enhance the reliability of generated code. However, existing strategies predominantly rely on static weighting for logit fusion. This rigidity ignores the dynamic nature of generation, where an expert model's sensitivity to context and predictive reliability can fluctuate significantly across decoding steps, potentially leading to suboptimal fusion outcomes.

To address this, we leverage entropy as a robust proxy for uncertainty. We posit that the entropy shift between the fine-tuned and base models quantifies the confidence gain from the fine-tuning process. Specifically, a significant reduction in entropy implies that the expert model has acquired additional confidence through fine-tuning. Consequently, we propose an entropy-based adaptive logit fusion strategy. Instead of using fixed weights, this mechanism dynamically modulates the expert's guidance strength based on the magnitude of this confidence gap, thereby prioritizing expert knowledge when it is most reliable. Formally, entropy is defined as follows:

$$H(\pi_i) = -\sum_{x \in V} \pi_i(x) \log \pi_i(x), \tag{4}$$

Here, V denotes the entire vocabulary, and $\pi_i(x)$ denotes the probability that model i predicts the next token to be x. Then, at each decoding step t, the weight w_i of expert model i is determined according to the following formula:

$$\delta(\pi_i) = 1 - \frac{H^{ft}(\pi_i)}{H^{pt}(\pi_i)}, \; w_i = \beta \cdot \frac{\exp \delta(\pi_i)}{\sum_j \exp \delta(\pi_j)}, \tag{5}$$

Where $H^{pt}(\pi_i)$ and $H^{ft}(\pi_i)$ represent the entropy of the token prediction distribution before and after fine-tuning for expert i, and β is a hyperparameter governing the overall guidance strength. Consistent with our hypothesis, a substantial drop in entropy signals enhanced predictive confidence derived from fine-tuning. Therefore, our mechanism assigns a larger weight to such high-confidence predictions, allowing the guidance signal to dynamically adapt to the specific decoding context. Finally, the adjusted logits $\tilde{\mathbf{z}}$ are computed as:

$$\tilde{\mathbf{z}} = \mathbf{z} + w_s(\mathbf{z}_s^{ft} - \mathbf{z}_s^{pt}) + w_c(\mathbf{z}_c^{ft} - \mathbf{z}_c^{pt}). \tag{6}$$

IV. Experiments

A. Experimental Setup

1) Models: Considering the open-source ecosystem, community popularity, and computational resource constraints, we evaluate ASCG on three representative code LLM families: CodeGen (350M, 2B, 6B), StarCoder (1B, 3B, 7B), and Deepseek-Coder (1.3B, 6.7B). In each series, the smallest variant is fine-tuned to serve as the expert model.

2) Datasets: For security tuning, we utilize the dataset proposed by SVEN [4], which comprises Python and C vulnerability–fix pairs across 9 common CWE categories. Regarding the correctness expert, we constructed a high-quality instruction-tuning corpus for training by extracting approximately 6k C and 20k Python instances from CodeFeedback,

TABLE I
MAIN RESULTS ON CODEGUARDPLUS†.

Target Model	Setting	CodeGuardPlus†		
		secure@1	pass@1	secure-pass@1
CodeGen-2B	n/a	51.56	59.08	30.75
	with ASCG	**67.65**	**63.46**	**43.96**
CodeGen-6B	n/a	62.16	59.42	34.71
	with ASCG	**78.17**	**68.88**	**51.71**
StarCoder-3B	n/a	68.95	66.40	46.67
	with ASCG	**77.57**	**71.46**	**56.29**
StarCoder-7B	n/a	69.37	71.50	51.83
	with ASCG	**76.09**	**78.38**	**62.00**
Deepseek-Coder-6.7B	n/a	64.13	**76.17**	51.17
	with ASCG	**72.38**	75.75	**58.42**

TABLE II
ABLATION STUDIES ON CODEGUARDPLUS†.

Target Model	Component	Metrics		
		secure@1	pass@1	s-p@1
CodeGen-2B	w/o correctness expert	**68.47**	56.42	41.46
	w/o logit strategy	56.33	63.21	39.33
	w/o dynamic weight	64.92	60.75	40.41
	with all	67.65	**63.46**	**43.96**
StarCoder-7B	w/o correctness expert	75.83	71.75	57.96
	w/o logit strategy	70.53	73.04	53.12
	w/o dynamic weight	**76.26**	76.94	61.12
	with all	76.09	**78.38**	**62.00**
Deepseek-Coder-6.7B	w/o correctness expert	70.23	**76.50**	58.38
	w/o logit strategy	65.15	75.71	53.04
	w/o dynamic weight	71.42	74.86	57.65
	with all	**72.38**	75.75	**58.42**

TABLE III
MAIN RESULTS ON HUMANEVAL.

Target Model	Setting	HumanEval	
		pass@1	pass@10
CodeGen-2B	n/a	13.1	24.5
	with ASCG	**16.4**	**29.1**
CodeGen-6B	n/a	18.6	28.2
	with ASCG	**19.7**	**32.3**
StarCoder-3B	n/a	0.1	0.2
	with ASCG	**1.0**	**5.8**
StarCoder-7B	n/a	0.5	3.3
	with ASCG	**2.7**	**11.9**
Deepseek-Coder-6.7B	n/a	44.4	71.6
	with ASCG	**48.8**	**72.5**

TABLE IV
ABLATION STUDIES ON HUMANEVAL.

Target Model	Component	Metrics	
		pass@1	pass@10
CodeGen-2B	w/o correctness expert	13.1	25.1
	w/o logit strategy	12.6	21.8
	w/o dynamic weight	16.1	28.5
	with all	**16.4**	**29.1**
StarCoder-7B	w/o correctness expert	0.4	3.5
	w/o logit strategy	0.8	4.2
	w/o dynamic weight	2.4	11.6
	with all	**2.7**	**11.9**
Deepseek-Coder-6.7B	w/o correctness expert	44.0	72.1
	w/o logit strategy	44.1	67.7
	w/o dynamic weight	47.5	**73.4**
	with all	**48.8**	72.5

Magicoder-Evol, GlaiveCode, and OpenCodeInstruct [11]–[14]. For both datasets, we randomly split the data into 90% for training and 10% for validation.

3) Evaluation: Effectiveness is assessed on two benchmarks tailored to specific code generation scenarios. First, CodeGuardPlus [29] is adopted to assess method performance in security-sensitive scenarios where code completions may introduce vulnerabilities. Compared to the widely adopted SVEN, CodeGuardPlus improves task prompt construction and, crucially, integrates well-structured unit tests. It covers both Python and C tasks, employing CodeQL and SonarQube for vulnerability detection along with unit tests for functionality. Second, following previous studies, we use HumanEval [1] to assess the capability of methods in general-purpose scenarios. This widely established benchmark serves as a standard for evaluating the functional correctness of generated code, comprising 164 Python function-completion tasks.

4) Metrics: We first introduce the notations used for our evaluation metrics. Let n denote the total number of sampled completions for each task. n_p, n_s and n_c respectively represent the number of compilable, secure (passing security analysis), and correct samples (passing all unit tests), while n_{sc} denotes the number of samples that are both secure and correct.

pass@k: This metric, used in both HumanEval and CodeGuardPlus, reflects the functional correctness of the top-k generated completions and it is computed as:

$$\text{pass@k} = \mathbb{E}_{x \in X}\left[1 - \frac{\binom{n-n_c}{k}}{\binom{n}{k}}\right]. \tag{7}$$

secure-pass@k: Introduced in CodeGuardPlus, this metric highlights the importance of preserving functional correctness given secure code generation. Its formulation is:

$$\text{secure-pass@k} = \mathbb{E}_{x \in X}\left[1 - \frac{\binom{n-n_{sc}}{k}}{\binom{n}{k}}\right]. \tag{8}$$

secure@k: Inspired by the observation in CodeGuardPlus that the original security metric, Security Ratio, overestimates in the presence of duplicated vulnerable completions, we introduce secure@k as an alternative that emphasizes security while retaining the same form as the aforementioned metrics:

$$\text{secure@k} = \mathbb{E}_{x \in X}\left[1 - \frac{\binom{n_p-n_s}{k}}{\binom{n_p}{k}}\right]. \tag{9}$$

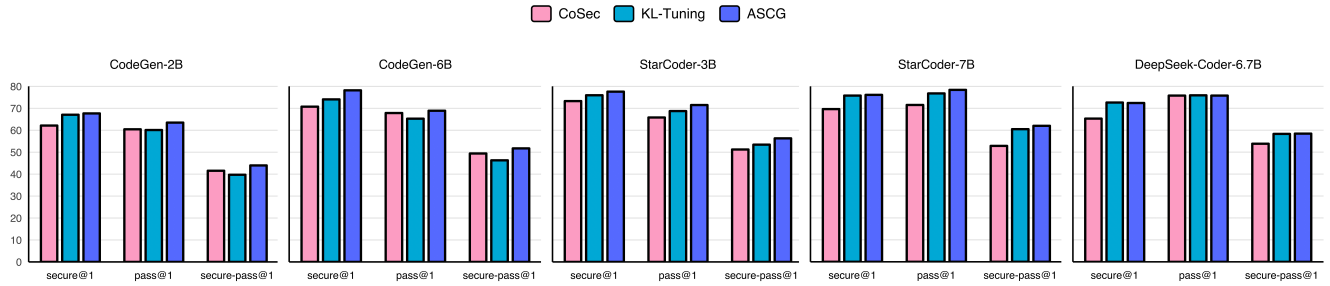

Fig. 2. Comparison of ASCG with CoSec and KL-Tuning across five code LLMs on CodeGuardPlus†. The results indicate that ASCG exhibits a stronger advantage in terms of code security.

TABLE V
DETAILED HYPERPARAMETER CONFIGURATIONS.

Training Configuration		
Parameter	**Security Expert**	**Correctness Expert**
Warm Up Steps	50	120
Learning Rate	5e-5	1e-4
Adam Epsilon	1e-8	1e-8
Weight Decay	0.01	0.01
Gradient Acc Step	2	16
Max. Gradient Norm	1.0	1.0
Batch Size	1	4
Epochs	8	4
LoRA Rank	8	16
LoRA Alpha	16	32
LoRA Dropout	0.05	0.05
Max. Input Length	2048 (CodeGen) / 4096 (Others)	
Inference Configuration		
Method / Variant	**Setting**	
ASCG	$\beta = 1.2$	
CoSec	$r = 0.3$	
KL-Tuning	$\alpha \in [0, 2]$, step $= 0.1$	
w/o correctness expert	$\alpha = 0.6$	
w/o dynamic weight	$\alpha_i = 0.6, i \in \{1, 2\}$	
w/o logit strategy	$r_i = 0.3, i \in \{1, 2\}$	

5) Other Details: We fine-tuned both experts using the AdamW optimizer with a linear warm-up strategy. For evaluation, we employed Nucleus Sampling (top-p=0.95, Temp=0.4) to generate 25 candidates. Detailed training and inference settings are provided in Table V. To ensure the reliability of our results, we repeated sampling 25 times using different random seeds.

All experiments are conducted on 4 NVIDIA V100 GPUs (32GB) and 2 NVIDIA A800 GPUs (40GB).

B. Main Results

Our main results are presented in Table I and Table III. Specifically for CodeGuardPlus, we employ a subset, denoted as CodeGuardPlus†, which includes tasks associated with the 9 CWEs (out of the full 43) encountered during training. We observe that while base models generally struggle to jointly ensure security and functional correctness (secure-pass@1) relative to their independent performance (secure@1 or pass@1), the ASCG framework achieves consistent improvements across both dimensions. This effectiveness is substantial, reflected in secure-pass@1 gains of approximately 7–13%. Notably, Deepseek-Coder-6.7B exhibits a suboptimal pass@1 performance on CodeGuardPlus†, which we attribute to the relatively limited domain knowledge of the correctness expert. Nonetheless, its improvements on HumanEval confirm the robustness of ASCG. Furthermore, a detailed breakdown of CodeGen-2B performance on CodeGuardPlus† across individual CWEs and scenarios reveals that ASCG yields consistent security improvements in nearly all cases, while simultaneously enhancing functional correctness in over half of the scenarios (18 out of 32).

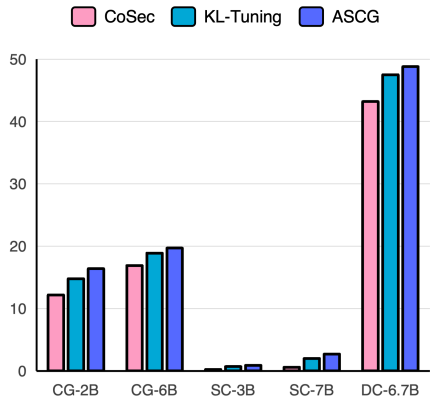

Fig. 3. Comparison of ASCG with CoSec and KL-Tuning across five code LLMs on HumanEval. Here, CG denotes CodeGen, SC denotes StarCoder, and DC denotes DeepSeek-Coder. The results indicate that ASCG exhibits a stronger advantage in terms of function correctness.

C. Ablation Studies

Next, we evaluate the contribution of each key component of ASCG through three ablation baselines to assess their impact on generating secure and functionally correct code across different models, with the results presented in Tables II and IV. Details of the experimental setup are provided in Table V.

For the 'w/o correctness expert' baseline, we remove the functional correctness expert, retaining only the security expert with a fixed weight. This generally leads to the most significant pass@k degradation across benchmarks, highlighting the

TABLE VI
AVERAGE DECODING TIME (MS) PER CALL ($>$ 10K CALLS).

Target Model	Metric	n/a	CoSec	KL-Tuning	ASCG
CodeGen-2B	T_{logit}	1.01	2.47	310.84	1.55
	T_{step}	53.22	79.05	441.91	114.20
CodeGen-6B	T_{logit}	1.03	2.45	306.38	1.48
	T_{step}	73.12	97.17	456.36	131.18
StarCoder-3B	T_{logit}	1.00	2.41	292.30	1.58
	T_{step}	21.94	41.39	369.43	77.92
StarCoder-7B	T_{logit}	0.96	2.55	247.73	1.50
	T_{step}	33.19	54.76	336.79	91.30
Deepseek-Coder-6.7B	T_{logit}	0.96	2.45	220.85	1.36
	T_{step}	55.15	93.59	386.07	164.87

TABLE VII
ASCG PERFORMANCE ON CODEGUARDPLUS‡.

Target Model	Method	pass@1	secure@1	secure-pass@1
CodeGen-2B	n/a	43.43	57.99	22.74
	ASCG	41.49	58.63	23.20
StarCoder-7B	n/a	67.54	55.10	34.23
	ASCG	68.69	59.10	38.30
Deepseek-Coder-6.7B	n/a	72.69	56.19	38.00
	ASCG	70.40	56.71	37.94

critical role of the correctness expert. However, an exception is observed with Deepseek-Coder-6.7B, possibly due to its sufficient intrinsic knowledge rendering the expert redundant.

In the second baseline "w/o logit strategy", we replace logit fusion with a token selection method adapted from CoSec [7]. Specifically, among the tokens predicted by the security expert, we retain only those favored by both the correctness expert and the target model. Results indicate that all metrics degrade compared to the full framework, with a maximum drop of 11.32%, demonstrating that fine-grained logits-level guidance is more effective than probability-based token filtering.

The last baseline, "w/o dynamic weight", employs fixed weights in logit fusion (Eq. 6). It achieves the best ablation results but still underperforms our full method, with an average 3% drop on CodeGen-2B. We attribute the marginal gap to the target model's existing proficiency and the potential saturation of expert guidance. However, the results confirm that adaptive weighting offers critical gains, particularly when models lack the inherent capability to balance objectives.

D. Comparison Results

Now we perform a comparative analyses against two established baselines: CoSec [7] and KL-Tuning. CoSec integrates a security expert to collaboratively decode, with token selection determined by the probabilistic ratio. In contrast, KL-Tuning modifies the ASCG by substituting its logit fusion mechanism with a KL-divergence-based search strategy [28] to dynamically optimize expert weights at each decoding step. To facilitate an equitable comparison, all methods share the same security and correctness experts. Baseline hyperparameters follow their original publications, with details in Table V.

The comparative results are presented in Figures 2 and 3. ASCG generally outperforms CoSec, substantially enhancing both security and functional correctness, which is largely attributed to the domain knowledge provided by the correctness expert and the improved guidance efficiency enabled by the logit fusion mechanism. Regarding KL-Tuning, we observe that it narrows the performance gap and occasionally achieves results comparable to ASCG, such as on DeepSeek-Coder-6.7B. We further assess the inference speed in Table VI, where T_{logit} denotes the time consumed solely for selecting a token from logits, and T_{step} represents the latency of a complete decoding step, including the forward pass time. The results indicate that KL-Tuning exhibits considerable computational overhead due to its iterative search for optimal weights at each decoding step. In contrast, ASCG adopts a simpler entropy-driven approach, achieving a speedup of up to 300$\times$ in T_{logit} over KL-Tuning, thereby offering a significantly more practical solution with comparable effectiveness.

Furthermore, comparisons with the baseline and CoSec reveal notable findings. Regarding T_{logit}, ASCG is slightly faster than CoSec. We attribute this efficiency to the fact that CoSec necessitates computationally intensive probability-space transformations involving redundant normalizations, whereas ASCG minimizes overhead by operating directly on unnormalized logits with simple arithmetic. In terms of T_{step}, ASCG inevitably incurs higher latency due to the additional forward passes required by the multiple expert models, though this overhead remains acceptable. Notably, while our current implementation executes the target and expert models sequentially, this inefficiency can be significantly mitigated by parallelizing their forward propagation processes.

E. Performance on Unseen CWEs

Finally, we evaluate ASCG on the remaining tasks in CodeGuardPlus involving CWEs unseen during training (denoted as CodeGuardPlus‡). The results, presented in Table VII, indicate that ASCG achieves marginal gains on certain models. However, we conducted a further granular analysis which reveals distinct performance disparities across different unseen CWEs. Consequently, ASCG does not significantly enhance reliability on these unseen scenarios. This observation aligns with prior findings from SafeCoder [5], which demonstrated that direct instruction tuning fails to generalize to unseen vulnerabilities. We attribute this to the fact that vulnerability mitigation relies on specific domain knowledge that is difficult to acquire through simple fine-tuning. Given that ASCG explores the paradigm of knowledge injection via fine-tuning, such results are reasonable. We leave this for future investigation.

V. CONCLUSION

In this work, we propose ASCG, an adaptive framework for secure code generation. ASCG leverages two expert models, specialized in security and functional correctness, to steer the target LLM's behavior. During training, the experts are separately optimized with vulnerability-fixing and instruction code

data. We introduce an entropy-based logit fusion mechanism that dynamically balances their contributions. Experiments across multiple code LLMs demonstrate the effectiveness of ASCG in enhancing the quality of generated code. Future work proceeds along two directions. One is to enhance expert specialization for stronger guidance, and the other is to improve inference efficiency and memory utilization, as the current strategy requires concurrent execution of multiple models.

REFERENCES

[1] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[2] D. Fried, A. Aghajanyan, J. Lin, S. Wang, E. Wallace, F. Shi, R. Zhong, S. Yih, L. Zettlemoyer, and M. Lewis, "Incoder: A generative model for code infilling and synthesis," in *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. [Online]. Available: https://openreview.net/forum?id=hQwb-lbM6EL

[3] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. R. Narasimhan, "Swe-bench: Can language models resolve real-world github issues?" in *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. [Online]. Available: https://openreview.net/forum?id=VTF8yNQM66

[4] J. He and M. T. Vechev, "Large language models for code: Security hardening and adversarial testing," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*, W. Meng, C. D. Jensen, C. Cremers, and E. Kirda, Eds. ACM, 2023, pp. 1865–1879. [Online]. Available: https://doi.org/10.1145/3576915.3623175

[5] J. He, M. Vero, G. Krasnopolska, and M. T. Vechev, "Instruction tuning for secure code generation," in *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. [Online]. Available: https://openreview.net/forum?id=MgTzMaYHvG

[6] J. Res, I. Homoliak, M. Peresíni, A. Smrcka, K. Malinka, and P. Hanácek, "Enhancing security of ai-based code synthesis with github copilot via cheap and efficient prompt-engineering," *CoRR*, vol. abs/2403.12671, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2403.12671

[7] D. Li, M. Yan, Y. Zhang, Z. Liu, C. Liu, X. Zhang, T. Chen, and D. Lo, "Cosec: On-the-fly security hardening of code llms via supervised co-decoding," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2024, Vienna, Austria, September 16-20, 2024*, M. Christakis and M. Pradel, Eds. ACM, 2024, pp. 1428–1439. [Online]. Available: https://doi.org/10.1145/3650212.3680371

[8] A. Nunez, N. T. Islam, S. K. Jha, and P. Najafirad, "Autosafecoder: A multi-agent framework for securing LLM code generation through static analysis and fuzz testing," *CoRR*, vol. abs/2409.10737, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2409.10737

[9] R. Saul, H. Wang, K. Sen, and D. A. Wagner, "Scgagent: Recreating the benefits of reasoning models for secure code generation with agentic workflows," *CoRR*, vol. abs/2506.07313, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2506.07313

[10] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," *CoRR*, vol. abs/2106.09685, 2021. [Online]. Available: https://arxiv.org/abs/2106.09685

[11] W. U. Ahmad, A. Ficek, M. Samadi, J. Huang, V. Noroozi, S. Majumdar, and B. Ginsburg, "Opencodeinstruct: A large-scale instruction tuning dataset for code llms," *CoRR*, vol. abs/2504.04030, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2504.04030

[12] T. Zheng, G. Zhang, T. Shen, X. Liu, B. Y. Lin, J. Fu, W. Chen, and X. Yue, "Opencodeinterpreter: Integrating code generation with execution and refinement," *CoRR*, vol. abs/2402.14658, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2402.14658

[13] ise-uiuc, "Magicoder-Evol-Instruct-110K," https://huggingface.co/datasets/ise-uiuc/Magicoder-Evol-Instruct-110K, 2023, accessed: 2025-07-26.

[14] glaiveai, "Glaive-code-assistant-v3," https://huggingface.co/datasets/glaiveai/glaive-code-assistant-v3, 2023, accessed: 2025-07-26.

[15] E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, S. Savarese, and C. Xiong, "Codegen: An open large language model for code with multi-turn program synthesis," in *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. [Online]. Available: https://openreview.net/forum?id=iaYcJKpY2B_

[16] B. Rozière, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, T. Remez, J. Rapin, A. Kozhevnikov, I. Evtimov, J. Bitton, M. Bhatt, C. Canton-Ferrer, A. Grattafiori, W. Xiong, A. Défossez, J. Copet, F. Azhar, H. Touvron, L. Martin, N. Usunier, T. Scialom, and G. Synnaeve, "Code llama: Open foundation models for code," *CoRR*, vol. abs/2308.12950, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2308.12950

[17] D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, X. Bi, Y. Wu, Y. K. Li, F. Luo, Y. Xiong, and W. Liang, "Deepseek-coder: When the large language model meets programming - the rise of code intelligence," *CoRR*, vol. abs/2401.14196, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2401.14196

[18] B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, T. Liu, J. Zhang, B. Yu, K. Dang, A. Yang, R. Men, F. Huang, X. Ren, X. Ren, J. Zhou, and J. Lin, "Qwen2.5-coder technical report," *CoRR*, vol. abs/2409.12186, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2409.12186

[19] MITRE, "CWE: common weakness enumerations," https://cwe.mitre.org/, 2025, accessed: 2025-07-26.

[20] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, "Asleep at the keyboard? assessing the security of github copilot's code contributions," in *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*. IEEE, 2022, pp. 754–768. [Online]. Available: https://doi.org/10.1109/SP46214.2022.9833571

[21] A. Liu, M. Sap, X. Lu, S. Swayamdipta, C. Bhagavatula, N. A. Smith, and Y. Choi, "Dexperts: Decoding-time controlled text generation with experts and anti-experts," 2021. [Online]. Available: https://arxiv.org/abs/2105.03023

[22] A. Liu, X. Han, Y. Wang, Y. Tsvetkov, Y. Choi, and N. A. Smith, "Tuning language models by proxy," *CoRR*, vol. abs/2401.08565, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2401.08565

[23] E. Mitchell, R. Rafailov, A. Sharma, C. Finn, and C. D. Manning, "An emulator for fine-tuning large language models using small language models," in *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. [Online]. Available: https://openreview.net/forum?id=Eo7kv0sllr

[24] H. W. Chung, L. Hou, S. Longpre, B. Zoph, Y. Tay, W. Fedus, E. Li, X. Wang, M. Dehghani, S. Brahma, A. Webson, S. S. Gu, Z. Dai, M. Suzgun, X. Chen, A. Chowdhery, S. Narang, G. Mishra, A. Yu, V. Y. Zhao, Y. Huang, A. M. Dai, H. Yu, S. Petrov, E. H. Chi, J. Dean, J. Devlin, A. Roberts, D. Zhou, Q. V. Le, and J. Wei, "Scaling instruction-finetuned language models," *CoRR*, vol. abs/2210.11416, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2210.11416

[25] J. Wei, M. Bosma, V. Y. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le, "Finetuned language models are zero-shot learners," in *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. [Online]. Available: https://openreview.net/forum?id=gEZrGCozdqR

[26] Y. Wang, H. Le, A. D. Gotmare, N. D. Q. Bui, J. Li, and S. C. H. Hoi, "Codet5+: Open code large language models for code understanding and generation," 2023. [Online]. Available: https://arxiv.org/abs/2305.07922

[27] K. Li, Q. Hu, X. Zhao, H. Chen, Y. Xie, T. Liu, Q. Xie, and J. He, "Instructcoder: Instruction tuning large language models for code editing," 2024. [Online]. Available: https://arxiv.org/abs/2310.20329

[28] C. Fan, Z. Lu, W. Wei, J. Tian, X. Qu, D. Chen, and Y. Cheng, "On giant's shoulders: Effortless weak to strong by dynamic logits fusion," 2024. [Online]. Available: https://arxiv.org/abs/2406.15480

[29] Y. Fu, E. Baker, and Y. Chen, "Constrained decoding for secure code generation," *CoRR*, vol. abs/2405.00218, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2405.00218