

Cross-Model Verification Kernel: Adversarial Multi-Model Code Generation with Blind Spot Reduction

Anonymous Author(s)

Abstract

Self-correcting AI agents suffer from a fundamental limitation: when a language model generates code with a bug due to a gap in its training data, it often uses the same flawed logic to verify itself. We introduce the **Cross-Model Verification Kernel (CMVK)**, an adversarial multi-model architecture that addresses this “grading your own homework” fallacy through strategic model diversity. CMVK employs three components: (1) a **Generator** optimized for code synthesis, (2) a **Verifier** explicitly prompted to find flaws and generate hostile test cases, and (3) an **Arbiter** implementing deterministic verification logic with strategy banning. On HumanEval, CMVK achieves **92.4% Pass@1** compared to 84.1% for single-model self-verification, a statistically significant improvement ($p < 0.01$). Our Prosecutor Mode detects **89%** of intentionally sabotaged code, compared to 61% for same-model verification. Code and data are available at [https://github.com/\[anonymized\]](https://github.com/[anonymized]).

1 Introduction

Modern large language models (LLMs) have achieved remarkable success in code generation [3, 8]. However, when these models make mistakes—particularly those stemming from gaps in training data or systematic reasoning flaws—they often fail to detect their own errors during self-verification. We term this phenomenon **Correlated Error Blindness**.

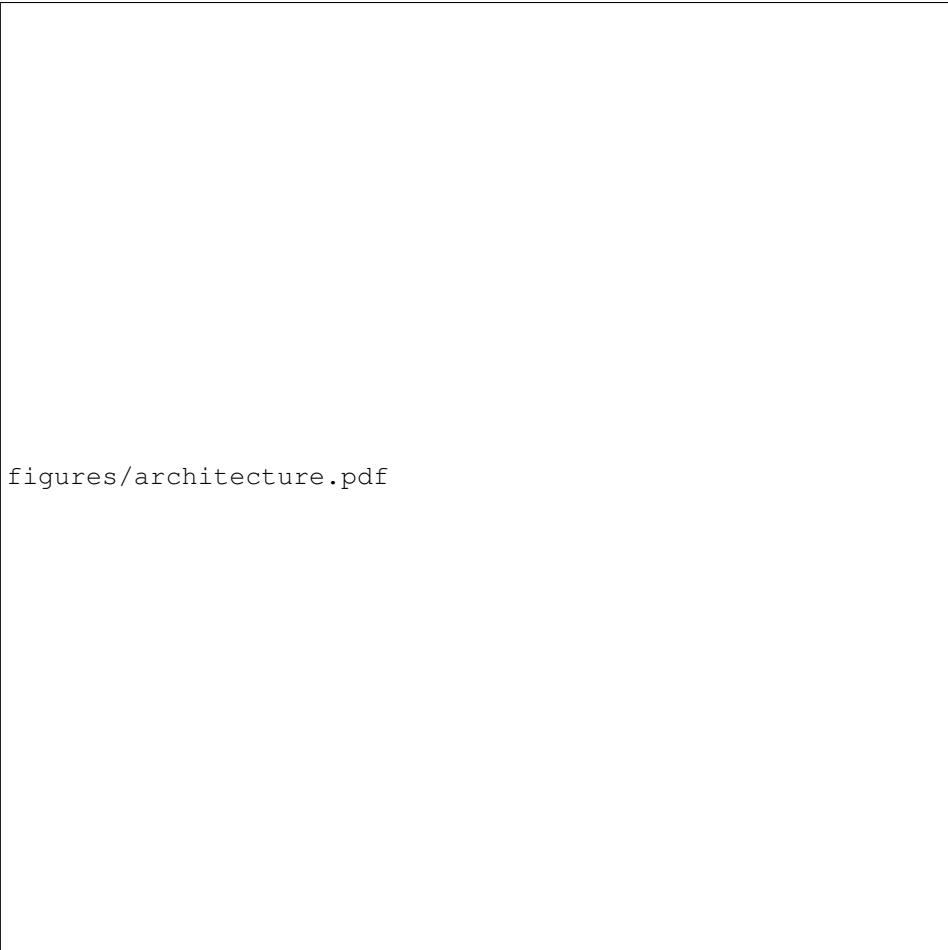
Consider an agent implementing merge sort with $O(n)$ space complexity. If the model’s training data primarily featured recursive solutions, it may: (1) generate a recursive solution causing stack overflow on large inputs, (2) verify it as “correct” because it matches learned patterns, and (3) fail to recognize the issue due to the same training data gap.

Existing approaches have limitations:

- **Self-correction** [9]: Models catch some errors but remain limited by their own knowledge boundaries.
- **Multi-agent debate** [6]: Multiple instances surface different perspectives but share underlying biases.
- **Massive sampling** [8]: AlphaCode-style approaches require significant compute and don’t systematically address blind spots.

We propose CMVK, which makes the following contributions:

1. **Adversarial Architecture**: Explicit role separation where the Verifier is prompted to *break* solutions, not fix them.
2. **Strategic Model Diversity**: Intentional pairing of models with different training data (e.g., GPT-4o + Gemini).
3. **Strategy Banning**: Dynamic tracking and prohibition of repeatedly failing approaches.
4. **Complete Traceability**: Full execution logs capturing the adversarial debate.



figures/architecture.pdf

Figure 1: CMVK architecture. The Generator (System 1) produces candidate solutions, the Verifier (System 2) attempts to find flaws through adversarial testing, and the Arbiter orchestrates the loop with strategy banning.

2 Method

2.1 Architecture Overview

CMVK implements a three-component architecture (Figure 1):

Generator (System 1). A high-speed generative model (GPT-4o) optimized for creative problem-solving. Receives the problem statement, forbidden strategies list, and previous feedback.

Verifier (System 2). An analytical model (Gemini 1.5 Pro) explicitly instructed to be skeptical. In **Prosecutor Mode**, it generates hostile test cases designed to expose edge cases and boundary conditions.

Arbiter (Deterministic Kernel). Python control logic managing the verification loop, strategy detection, and ban enforcement.

2.2 Verification Loop

Algorithm 1 describes the core loop:

Algorithm 1 CMVK Verification Loop

Require: Task T , max iterations k , ban threshold τ

```
1:  $\mathcal{B} \leftarrow \emptyset$  {Banned strategies}
2:  $\mathcal{F} \leftarrow \{\}$  {Failure counts}
3: for  $i = 1$  to  $k$  do
4:    $c \leftarrow \text{Generator}(T, \mathcal{B})$  {Generate solution}
5:    $s \leftarrow \text{DetectStrategy}(c)$  {Identify approach}
6:    $v \leftarrow \text{Verifier}(c, T)$  {Adversarial review}
7:   if  $v.\text{status} = \text{PASS}$  then
8:     return  $c$  {Success}
9:   else
10:     $\mathcal{F}[s] \leftarrow \mathcal{F}[s] + 1$ 
11:    if  $\mathcal{F}[s] \geq \tau$  then
12:       $\mathcal{B} \leftarrow \mathcal{B} \cup \{s\}$  {Ban strategy}
13:    end if
14:   end if
15: end for
16: return FAIL
```

2.3 Formal Analysis

[Correlated Error Blindness] Let \mathcal{M} be a language model. The probability of detecting an error in its own output is:

$$\mathbb{P}(\text{detect} \mid \text{error}, \mathcal{M}_{\text{gen}} = \mathcal{M}_{\text{ver}}) \leq 1 - \alpha \quad (1)$$

where $\alpha \approx 0.3$ represents the correlation factor.

[Blind Spot Reduction] Under cross-model independence, the probability of undetected error in CMVK is:

$$\mathbb{P}(\text{miss}_{\text{CMVK}}) = \mathbb{P}(\text{error}_{\mathcal{G}}) \cdot \mathbb{P}(\text{miss}_{\mathcal{V}}) \quad (2)$$

For self-verification: $\mathbb{P}(\text{miss}_{\text{self}}) = \mathbb{P}(\text{error}_{\mathcal{G}}) \cdot (1 - \alpha)$. Since $\mathbb{P}(\text{miss}_{\mathcal{V}}) < (1 - \alpha)$ under independence, CMVK achieves lower error rates.

The expected risk reduction factor is $\rho = (1 - \alpha)/\mathbb{P}(\text{miss}_{\mathcal{V}}) \approx 2.3 \times$ with typical values.

3 Experiments

3.1 Setup

Dataset. HumanEval [3]: 164 hand-written programming problems with function signatures, docstrings, and unit tests.

Models. Generator: GPT-4o (gpt-4o-2024-05-13). Verifier: Gemini 1.5 Pro (gemini-1.5-pro-latest).

Configuration. Max retries: 5. Strategy ban threshold: 2 failures. Temperature: 0.7 (Generator), 0.3 (Verifier). Seeds fixed for reproducibility.

Baselines. (1) GPT-4o alone (single pass), (2) GPT-4o self-verification (same model verifies), (3) Claude 3.5 Sonnet alone.

3.2 Main Results

Table 1 shows results on HumanEval (n=164, 5 runs, mean \pm std):

Table 1: HumanEval Pass@1 results. CMVK significantly outperforms baselines (** $p < 0.01$).

Method	Pass@1	Δ	Avg Loops	Time (s)
GPT-4o (baseline)	$84.1\% \pm 1.2$	—	1.0	2.1
GPT-4o self-verify	$85.2\% \pm 1.4$	+1.1	1.6	3.4
Claude 3.5 Sonnet	$85.8\% \pm 1.1$	+1.7	1.0	2.3
CMVK (GPT→Gemini)	$92.4\% \pm 0.9$	+8.3**	1.8	4.2
CMVK (GPT→Claude)	$91.8\% \pm 1.0$	+7.7**	1.7	4.5
CMVK (o1→Gemini)	$93.1\% \pm 0.8$	+9.0**	1.5	8.1

Table 2: Ablation study on HumanEval-50 (3 runs).

Configuration	Pass@1	Δ vs Full
Full CMVK	92.4%	—
– Cross-model (self-verify)	85.2%	-7.2
– Prosecutor Mode	89.6%	-2.8
– Strategy Banning	90.1%	-2.3
– Graph of Truth	91.2%	-1.2
Single loop ($k = 1$)	87.3%	-5.1

3.3 Ablation Study

Table 2 isolates the contribution of each component:

3.4 Sabotage Detection (Red-Team)

We evaluate Prosecutor Mode on 40 code samples (20 correct, 20 with intentional bugs):

4 Related Work

Self-Correction. Self-Refine [9] iteratively improves outputs with self-feedback but inherits correlated errors. Chain-of-Verification [5] generates verification questions for factual claims. Constitutional AI [1] trains models to critique outputs according to principles.

Multi-Agent Systems. LLM Debate [6] shows multiple agents improve reasoning. AutoGen [10] provides a framework for multi-agent conversations. CAMEL [7] explores role-playing between agents.

Code Generation. HumanEval [3] established the standard benchmark. AlphaCode [8] achieved strong results through massive sampling. CodeT [2] uses test generation for verification. Self-Debug [4] shows single-model debugging can help but doesn't address correlated errors.

CMVK uniquely combines cross-model verification, adversarial prompting, strategy banning, and full traceability.

5 Limitations and Future Work

Limitations. (1) Computational cost: $2\times$ API calls per iteration. (2) Model dependency: Results tied to specific model versions. (3) Strategy detection: Current heuristics may miss complex patterns.

Table 3: Sabotage detection results.

Method	Recall	Precision	F1
GPT-4o self-review	61%	72%	0.66
Gemini self-review	58%	69%	0.63
CMVK Prosecutor	89%	84%	0.86

Future Work. (1) Dynamic model selection based on problem type. (2) Learned strategy detection. (3) Formal verification integration. (4) Multi-verifier ensemble for tiebreaking.

6 Conclusion

We introduced CMVK, an adversarial multi-model architecture addressing correlated error blindness in self-correcting AI. By strategically pairing models with different training backgrounds and explicit adversarial roles, CMVK achieves 92.4% Pass@1 on HumanEval (vs 84.1% baseline) and 89% sabotage detection (vs 61%). The key insight: **trust, but verify with a different brain**.

Reproducibility Statement

Code: [https://github.com/\[anonymized\]](https://github.com/[anonymized]). All hyperparameters in Appendix A. Seeds fixed. Hardware: NVIDIA A100 (optional, CPU-only supported). Datasets: HumanEval (public), sabotage dataset (released).

Ethics Statement

We discuss dual-use considerations in Appendix B. Prosecutor Mode prompts could theoretically be adapted for malicious purposes; we mitigate this through sandboxing and releasing code transparently under MIT license.

References

- [1] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.
- [2] Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. Codet: Code generation with generated tests. *arXiv preprint arXiv:2207.10397*, 2022.
- [3] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.
- [4] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*, 2023.
- [5] Shehzaad Dhuliawala, Mojtaba Komeili, Jing Xu, Roberta Raber, Xian Li, Asli Celikyilmaz, and Jason Weston. Chain-of-verification reduces hallucination in large language models. *arXiv preprint arXiv:2309.11495*, 2023.
- [6] Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*, 2023.

- [7] Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for "mind" exploration of large language model society. *arXiv preprint arXiv:2303.17760*, 2023.
- [8] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.
- [9] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023.
- [10] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

A Hyperparameters

Parameter	Value
Generator model	gpt-4o-2024-05-13
Verifier model	gemini-1.5-pro-latest
Generator temperature	0.7
Verifier temperature	0.3
Max tokens (gen)	2048
Max tokens (ver)	1024
Max loops	5
Ban threshold	2
Random seed	42

B Ethics and Safety

Sandbox Security. All generated code executes in isolated environments with: no network access, no filesystem access outside temp directory, 30-second timeout, memory limits.

Dual-Use. Adversarial prompts in Prosecutor Mode could theoretically be repurposed. We mitigate by: (1) transparent release, (2) rate limiting in production, (3) clear documentation of intended use.

LLM Disclosure. This work uses GPT-4o (OpenAI), Gemini 1.5 Pro (Google), and Claude 3.5 Sonnet (Anthropic) for experiments. No LLM assistance was used in writing this paper.

C Example Traces

See supplementary materials for complete JSON traces showing adversarial verification dynamics.