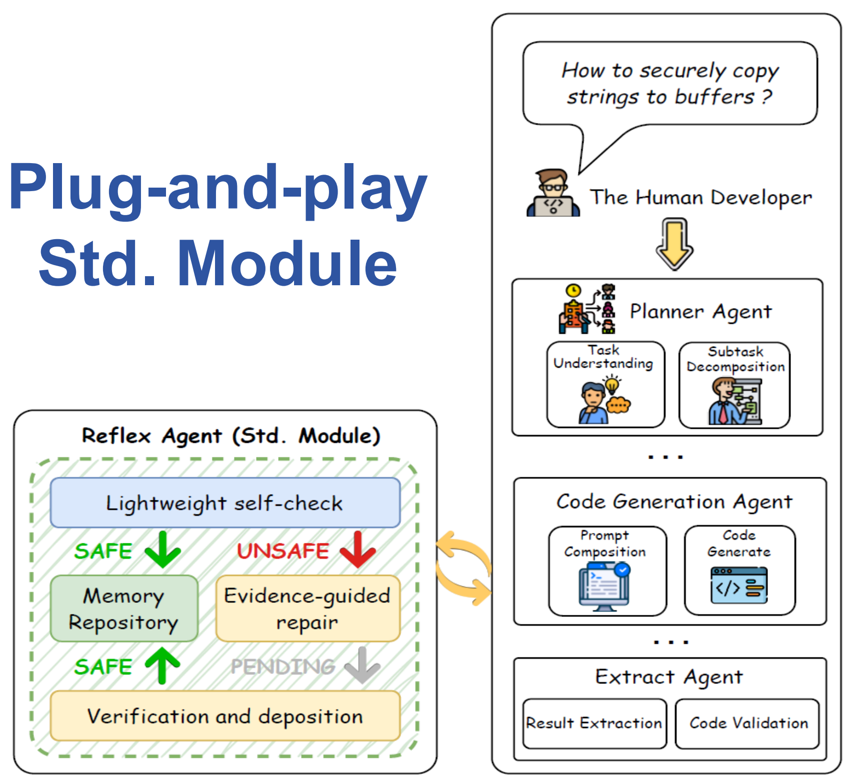
Reflection-Driven Control for Trustworthy Code Agents

Bin Wang[1], Jiazheng Quan[2], Xingrui Yu[3], Hansen Hu[1], Yuhao[1], Ivor Tsang[3]
[1]School of Computer Science, Peking University, China
[2]Xiamen University, China
[3]Centre for Frontier AI Research (CFAR), Agency for Science, Technology and Research (A*STAR)

Abstract

Contemporary large language model (LLM) agents are remarkably capable, but they still lack reliable safety controls and can produce unconstrained, unpredictable, and even actively harmful outputs. To address this, we introduce Reflection-Driven Control, a standardized and pluggable control module that can be seamlessly integrated into general agent architectures. Reflection-Driven Control elevates "self-reflection" from a post hoc patch into an explicit step in the agent's own reasoning process: during generation, the agent runs an internal reflection loop that monitors and evaluates its own decision path. When potential risks are detected, the system retrieves relevant repair examples and secure coding guidelines from an evolving reflective memory, injecting these evidence-based constraints directly into subsequent reasoning steps. We instantiate Reflection-Driven Control in the setting of secure code generation and systematically evaluate it across eight classes of security-critical programming tasks. Empirical results show that Reflection-Driven Control substantially improves the security and policy compliance of generated code while largely preserving functional correctness, with minimal runtime and token overhead. Taken together, these findings indicate that Reflection-Driven Control is a practical path toward trustworthy AI coding agents: it enables designs that are simultaneously autonomous, safer by construction, and auditable.

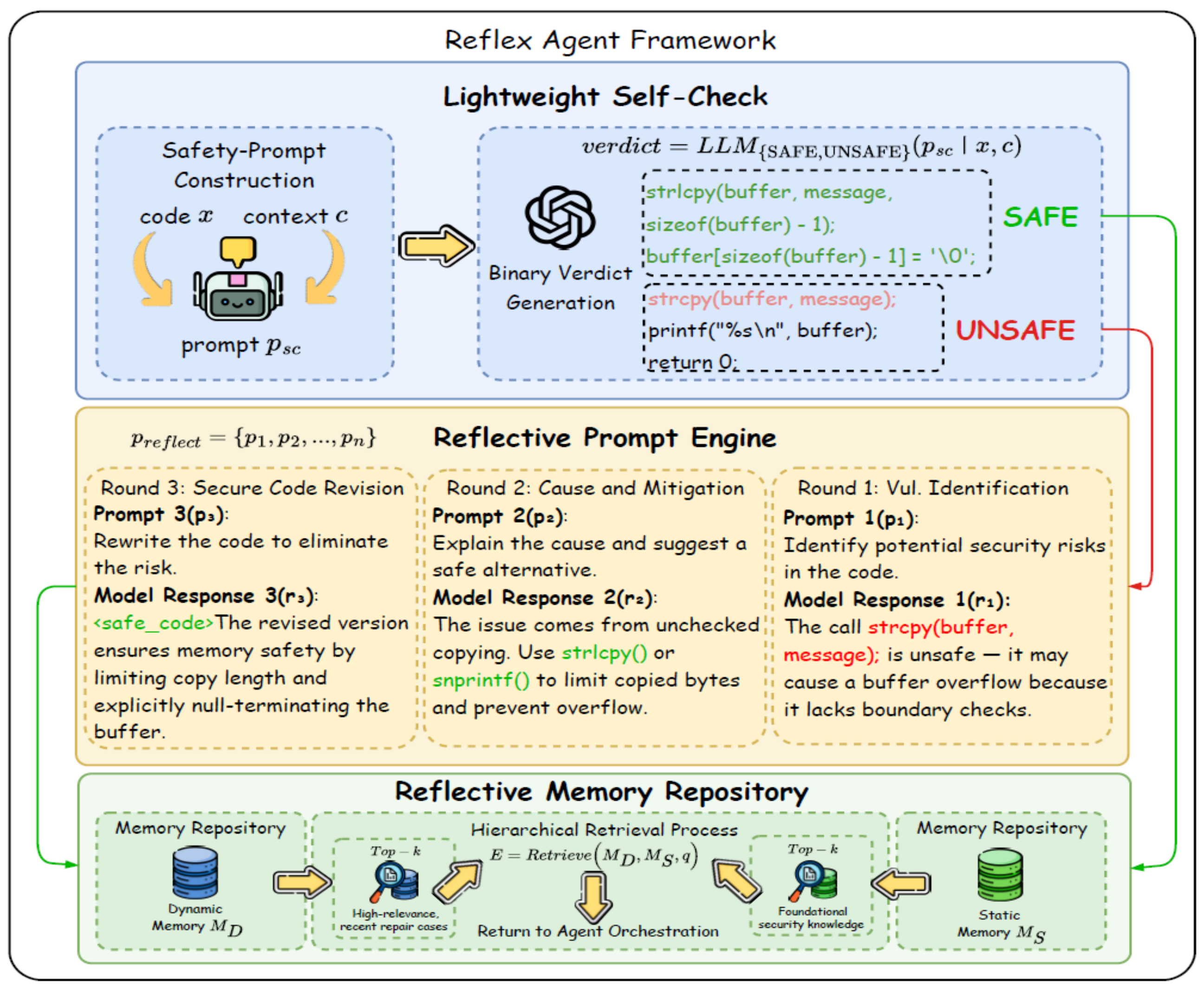
Plug-and-play Std. Module



Main contributions:

- ***A reflection-driven, closed-loop control framework.*** We integrate reflection as a first-class control circuit that spans planning, execution, and verification, with an auditable evidence trail rather than ad-hoc post-processing, and implemente it as the Reflect layer in a Plan–Reflect–Verify agentic framework.
- ***A practical instantiation for secure code generation.*** We compose lightweight self-checks, dynamic memory/RAG, reflective prompting, and tool governance (compiler/tests/CodeQL) into an evidence-grounded generation pipeline that maintains autonomy while enforcing safety.
- ***Comprehensive evaluation and analysis.*** On public security-oriented code-generation benchmarks with strict compile/run/static-analysis validation, our framework delivers consistent improvements over agent baselines, alongside ablations and case studies that quantify each component's impact and illuminate failure modes in high-risk settings.

Framework Overview


Reflex Agent Framework
Lightweight Self-Check
Safety-Prompt Construction
code x
context c
prompt p_{sc}
$verdict = LLM_{\{SAFE,UNSAFE\}}(p_{sc} \mid x, c)$
Binary Verdict Generation
strlcpy(buffer, message, sizeof(buffer) - 1); buffer[sizeof(buffer) - 1] = '\0';
SAFE
strcpy(buffer, message); printf("%s\n", buffer); return 0;
UNSAFE
$P_{reflect} = \{p_1, p_2, \ldots, p_n\}$
Reflective Prompt Engine
Round 3: Secure Code Revision
Prompt 3(p_3): Rewrite the code to eliminate the risk.
Model Response 3(r_3): <safe_code>The revised version ensures memory safety by limiting copy length and explicitly null-terminating the buffer.
Round 2: Cause and Mitigation
Prompt 2(p_2): Explain the cause and suggest a safe alternative.
Model Response 2(r_2): The issue comes from unchecked copying. Use strlcpy() or snprintf() to limit copied bytes and prevent overflow.
Round 1: Vul. Identification
Prompt 1(p_1): Identify potential security risks in the code.
Model Response 1(r_1): The call strcpy(buffer, message); is unsafe — it may cause a buffer overflow because it lacks boundary checks.
Reflective Memory Repository
Memory Repository
Dynamic Memory M_D
Hierarchical Retrieval Process
$E = Retrieve(M_D, M_S, q)$
Top-k
High-relevance, recent repair cases
Foundational security knowledge
Return to Agent Orchestration
Static Memory M_S

Reflex Module:
We instantiate the REFLEX module as the Reflect layer of a agentic framework, where reflection functions as a first-class control circuit supervising planning, execution, and verification. This design realizes a closed loop of low-cost, frontloaded reflection, evidence-driven generation, and auditable knowledge accumulation, without fine-tuning the underlying models. Building on this design, we implement a standard REFLEX agent that plugs into the system with three core components.

Lightweight Self-Check and Routing Mechanism:
The lightweight self-checker serves as a front-end filtering layer for the Reflex module, responsible for performing an initial, trustworthy safety diagnosis of the input code.

Reflective Prompt Engine:
The Reflective Prompt Engine serves as the core driving module of the reflection process, enabling the agent to perform deep analysis and self-improvement on problematic code.

Reflective Memory Repository:
The Reflective Memory Repository is the central component enabling the system's continual learning and experience reuse.

Evaluation Metrics

Metric	Definition / Interpretation
Security Rate (Sec. Rate)	Portion of compilable samples that are also security-clean. Sec. Rate $= \frac{\lvert\mathcal{S}\rvert}{\lvert\mathcal{C}\rvert} \times 100\%$. This reflects the model's ability to generate *executable* code with no detected CWE-class vulnerabilities.
Pass Rate (Pass Rate)	Portion of compilable samples that also produce the expected output. Pass Rate $= \frac{\lvert\mathcal{P}\rvert}{\lvert\mathcal{C}\rvert} \times 100\%$. This captures functional correctness under the task's I/O spec.
Total Efficiency (Eff. Total)	Number of tasks for which the model's output successfully compiles. Eff. Total $= \lvert\mathcal{C}\rvert$. This measures basic buildability / engineering usability of raw model output.
Security Count (Sec. Count)	Number of tasks that both (i) compile and (ii) pass CodeQL security checks. Sec. Count $= \lvert\mathcal{S}\rvert$. This is the numerator of Sec. Rate, shown as an absolute count.
Unresolved Count (Unres. Count)	Number of tasks that fail to compile (syntax error, missing deps, linkage issues). Unres. Count $= \lvert\mathcal{T}\rvert - \lvert\mathcal{C}\rvert$. This highlights early failure cases where code is not even buildable.

Qualitative Evaluation Metrics:

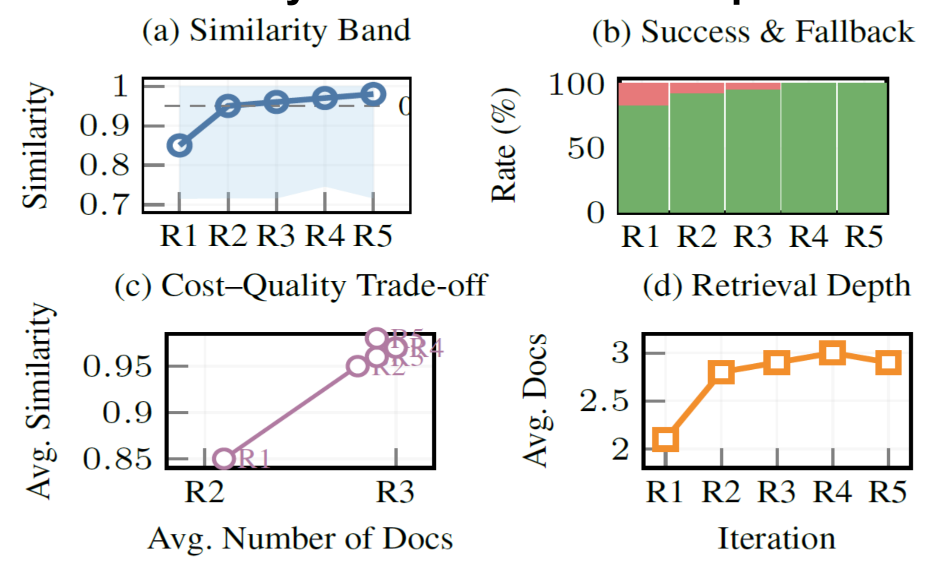
- ✓ **Code Quality**: We review readability, modularity, and maintainability. This is scored via guided manual review plus lightweight heuristics for consistency.
- ✓ **Security Completeness**: We check for input validation, error handling, privilege boundaries, sanitization, and coverage of common exploit classes. This reflects robustness under adversarial or unexpected inputs.
- ✓ **Compliance**: Whether the code respects privacy / data-handling / access-control expectations. Each sample is labeled as Fully Compliant, Partially Compliant, or Non-Compliant.

Evaluation Results

Reflex module agent performance summary

Metric	gpt-3.5-turbo		gpt-4o		qwen3-coder-plus		gemini-2.5-pro	
	Base	Base+Reflex	Base	Base+Reflex	Base	Base+Reflex	Base	Base+Reflex
Sec. Rate	93.7	96.6 (↑**2.9**)	85.7	95.0 (↑**9.3**)	83.7	94.9 (↑**11.2**)	88.0	97.1 (↑**9.1**)
Pass Rate	88.0	92.4 (↑**4.4**)	95.2	94.9 (↓0.3)	86.7	80.1 (↓6.6)	91.4	94.9 (↑**3.5**)
Eff. Total	22.0	23.1 (↑**1.1**)	23.8	23.7 (↓0.1)	21.6	20.2 (↓1.4)	22.9	23.7 (↑**0.8**)
Sec. Count	23.4	24.1 (↑**0.7**)	21.4	23.8 (↑**2.4**)	17.9	23.7 (↑**5.8**)	22.0	24.3 (↑**2.3**)
Unres. Count	3.0	1.9 (↓1.1)	1.2	1.3 (↑**0.1**)	3.3	4.8 (↑**1.5**)	2.1	1.3 (↓0.8)

Evolution of dynamic RAG retrieval performance


(a) Similarity Band
Similarity
R1 R2 R3 R4 R5
(b) Success & Fallback
Rate (%)
R1 R2 R3 R4 R5
(c) Cost–Quality Trade-off
Avg. Similarity
R1
R2
R3
Avg. Number of Docs
(d) Retrieval Depth
Avg. Docs
R1 R2 R3 R4 R5
Iteration