GRADUAL: A STATEFUL DUAL-AGENT ARCHITECTURE AGAINST INDIRECT PROMPT INJECTION

Anonymous authors
Paper under double-blind review

ABSTRACT

As agentic systems become more capable of handling complex, long-horizon tasks and gain broader privileges in open environments, indirect prompt injection (IPI) becomes a salient risk: tool use provides a channel through which adversarial content enters the model's context and induces unintended actions. Most existing IPI defenses rely on probabilistic detectors or system-level designs. Probabilistic ones fail under repeated attempts or are bypassed by adversarial attacks, whereas system-level ones are deterministic, but incur substantial cost in utility and require hand-crafted or static policies. We present **GraDual**, a dynamic **Gra**ph-driven **Dual**-agent framework; the main-dual-agent design isolates the trusted context from the threat of untrusted content, while both agents are capable of reasoning and acting under constraints; with main agent driven by a dynamic graph state, the system can also preserve high utility. Two agents collaborate closely under the framework but comply with an explicit interaction boundary. Generally, GraDual creates an information asymmetry to enable a successful task outsourcing while providing a dynamic defending guardrail to the trusted context. Specifically, in each outsourcing, main agent orchestrates two major components: outsourced task with resources, and a precommitment kept to itself. The first initiates dual with proper capabilities and constraints, and leverages dual's own task-solving ability. The second encodes the asymmetry as main's advantage, by enforcing return-path checks with code on the dual's untrusted data flow. Such design relies neither on direct probabilistic judgment, nor on system design with non-semantic use of data taint tracking or static policies. It fully leverages the agent's autonomy and intelligence to accomplish tasks securely. In evaluations across diverse tasks and IPI attacks, GraDual consistently achieves a 0% attack success rate while maintaining task utility competitive with baseline defenses. The ablation shows effectiveness of our dual-agent framework design as well as the importance of dynamically growing graph state.

1 INTRODUCTION

Agentic systems are granted broader privileges in open environments to act through tool use, web retrieval, memory and reusable skills. Adversarial prompting against LLM-centered systems has shifted from jailbreaks, which aim at what a model says (Li et al., 2023; Xu et al., 2024; Deng et al., 2024), to attacks that carry real-world consequences. Direct prompt injection shows that an instruction placed in the input can override the developer's own (Perez & Ribeiro, 2022; Liu et al., 2023), where the user is the assumed adversary. With more third-party tools and external APIs integrated into agentic systems, indirect prompt injection (IPI) introduces a new assumption that the user's query is benign, but a third-party adversary embeds malicious instructions in content returned by external interactions (Greshake et al., 2023). Hence, the manipulated prompt finds another channel to override a model's intended behaviour and induce tool calls with malicious purposes, and every external resource the agent touches becomes a potential command channel. Benchmarks have demonstrated the threat systematically (Debenedetti et al., 2024; Zhan et al., 2024; Jiang et al., 2026), CVEs have been assigned to vulnerabilities in deployed coding agents (National Institute of Standards and Technology, 2025; Narang, 2025), and IPI is now treated as an important deployment blocker in industry (Anthropic, 2025; OpenAI, 2025).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Defenses have been developing in recent years. The earliest defense measures are baked inside the model, marking untrusted spans, training the model to follow a separation between instruction and data, or adding a detector to judge whether malicious content comes in (Hines et al., 2024; Chen et al., 2025a; Liu et al., 2025; Zhu et al., 2025). Attacks adapt as well, from payloads optimized offline against a particular defense (Zhan et al., 2025) to forged chat-template markup that the tokenizer reads as a genuine conversational turn (Chang et al., 2026). A judgement made by a neural model is a probabilistic one, and an attacker may keep attempting, since some fraction of injections is obeyed (Ji et al., 2025; Abdelnabi & Bagdasarian, 2026). Certainty therefore requires the decision to be taken outside the probabilistic model (Willison, 2023), and researchers propose defenses that use code for certainty, converging on provenance by borrowing information-flow control from software security (Debenedetti et al., 2025; Costa et al., 2025; Fan et al., 2026). To keep that flow checkable, many of them settle the action plan in advance, fixing the control and data flow before anything is read (An et al., 2025; Li et al., 2026a), thereby trading utility for security (Xiang et al., 2026). The most recent work therefore defers the decision to execution time to recover task utility (Li et al., 2026b; Lin et al., 2026), or preserves flexibility through human-authored policies or user endorsement (Shi et al., 2025; Hu et al., 2026; Kolluri et al., 2026). In industrial practice, Anthropic gates the rollout of its browser-using agent behind site permissions and human confirmation (Anthropic, 2025). OpenAI states that robustness to such attacks remains an open research problem (OpenAI, 2025). Google vets each action its agentic Chrome proposes with a separate Gemini-based critic isolated from untrusted content and asks the user to confirm critical steps (Parker, 2025). What every one of these defenses meets is the same dilemma (Bhatt et al., 2026), and it is intrinsic to IPI: untrusted content must be kept out of the context that the model reads, yet the task cannot proceed without precisely what only that content holds.

We are motivated to resolve the dilemma by proposing **GraDual**, a dynamic **Gra**ph-driven **Dual**-agent architecture. We follow the isolation of the dual-LLM pattern (Willison, 2023), but not the IFC-based security model prior work adopts. We redesign it by first upgrading both sides into fully capable agents that can plan, reason and call given tools. Second, we build a co-working framework with an explicit trust boundary and an outsourcing protocol: a main agent that only reads trusted content and issues subtasks, and a dual agent that interacts with the untrusted environment and returns subtask results. Security does not come from limiting the capability of the dual agent, but from constructing an *information asymmetry* at test time. Before outsourcing a subtask, the main agent specifies a precommitment that constrains which returned results it will accept. It sends only the subtask and a capability grant to the dual agent, keeping the precommitment private. The dual agent reasons about the subtask and returns results, or negotiates with the main agent through a dedicated channel. Only results admitted by checks compiled from the precommitment update the dynamic graph that represents the main agent's state. This design separates the dual agent's exposure to adversarial content from the main agent's control over actions.

Our main contributions are summarized as follows.

- **A stateful dual-agent architecture with an explicit trust boundary and information asymmetry.** To defend against indirect prompt injection, we extend the ordinary single-agent loop into a coupled main–dual loop. The main agent plans from trusted context, delegates subtasks with tool access, and retains private precommitments that govern result admission. The dual agent processes untrusted content, while only code-validated results are admitted into the main agent's state, separating untrusted-content processing from task control.
- **A dynamic graph for runtime planning.** To preserve task utility under these security constraints, we maintain the main agent's state as a dynamic graph that grows with admitted results. The graph incorporates external results through admission checks and expands as execution proceeds. This allows the agent to resolve missing parameters and extend its plan at runtime, supporting tasks whose execution cannot be fully determined in advance.
- **An improved security–utility trade-off under long-horizon prompt injection.** We evaluate GraDual against long-horizon indirect prompt injection benchmark AgentLAB. Our evaluation covers four task suites, four attack types including an adaptive attack, and two primary backbone LLMs. Across these evaluations, GraDual is the only evaluated defense that consistently achieves an observed ASR of 0.0%. It exceeds the undefended agent in average task utility on both backbones, with gains of up to 18.1% over evaluated defense baselines. Ablation studies support the importance of code-enforced precommitments for security and dynamic graph growth for utility.

2 Related Work

2.1 Indirect Prompt Injection

Agents are gaining more capabilities over environments they do not control. However, retrieved content is another potential injection channel. Indirect prompt injection (IPI) has been identified as a significant threat to deployed LLM-integrated systems (Greshake et al., 2023). Subsequent benchmarks have demonstrated the threat systematically (Debenedetti et al., 2024; Zhang et al., 2025; Zhan et al., 2024; Yi et al., 2025; Jiang et al., 2026). The earliest and still the most common form of IPI is an explicit override prompt: an instruction telling the model to disregard what it was asked to do and do otherwise (Perez & Ribeiro, 2022; Liu et al., 2023). Instead of imperative method, TopicAttack induces the unwanted action through a gradual topic transition (Chen et al., 2025d). Payloads can also be optimized automatically against a particular defense rather than written by hand (Zhan et al., 2025). The most recent attacks have moved from phrasing to transport: ChatInject forges the backbone's own template inside tool responses so that the tokenizer parses injected text as a genuine conversation (Chang et al., 2026). Such attacks have been demonstrated against deployed agent systems (Rehberger, 2024) and have been assigned CVEs in coding agents (National Institute of Standards and Technology, 2025; Narang, 2025). The threat of IPI is treated as a deployment blocker in industry as well. Anthropic gates the rollout of its browser-using agent on prompt injection, reporting a 23.6% attack success rate over 123 adversarial test cases spanning 29 scenario (Anthropic, 2025). OpenAI also states that robustness to such attacks remains an open research problem (OpenAI, 2025).

2.2 Probabilistic Defenses

A representative class of defenses intervenes on the model itself, or on the representation of its input, so that instruction and data can be separated by the model's own reasoning. Methods differ in how that behaviour is instilled: by training, by steering with annotation, or through an auxiliary model. Spotlighting is the earliest and best known attempt, marking untrusted spans with delimiters, datamarking, or encodings (Hines et al., 2024). StruQ (Chen et al., 2025a) and Meta SecAlign (Chen et al., 2025c) introduce special roles that separate instructions from data and train the model to follow the separation. By contrast, DataSentinel adversarially trains a specialised detector LLM (Liu et al., 2025). Defensive tokens offer another route, steering a model toward instruction-hierarchy compliance (Chen et al., 2025b). Alternatively, MELON re-executes the task under a masked user query and flags actions that survive the mask (Zhu et al., 2025). However, evaluations of this class report high attack success rates under repeated attempts (Zhan et al., 2025; Ji et al., 2025).

2.3 System-level Defenses

The other class comprises defenses that constrain the system around the model, or harness, so that the flow of untrusted content is governed by deterministic components. Beurer-Kellner et al. (2025) systematize the recurring design patterns. Earlier designs of this kind provide security by fixing the control and data flow before any untrusted content is read. IPIGuard pre-compiles a tool-dependency graph for the task and refuses invocation unknown to graph (An et al., 2025). ACE separates planning into an abstract phase that never sees retrieved data, and a concrete phase where data access is mediated by capability barriers (Li et al., 2026a). Progent specifies tool-call privileges in a policy DSL enforced at runtime (Shi et al., 2025). AgentArmor recovers a program from execution and checks by static analysis (Wang et al., 2025). FIDES and PRUDENTIA propagates explicit confidentiality and integrity labels along the flow (Costa et al., 2025), and latter one hides untrusted content via variables and uses human endorsement to update taint labels (Kolluri et al., 2026). Xiang et al. (2026) argues that static plans and fixed policies are insufficient once environments are realistic, and a fixed flow generally buys security at high cost in utility. A more recent group of approaches therefore defers the decision to execution time to resolve the issue. DRIFT pairs dynamic rule-based constraints with injection isolation (Li et al., 2026b). VIGIL commits a speculative action only after an intent-grounded verification step (Lin et al., 2026). PACT assigns semantic roles to tool arguments and validates each value's provenance against a role-specific trust contract (Fan et al., 2026). One design pattern that cuts across both the static and the dynamic designs is the dual-LLM pattern (Willison, 2023), in which a privileged model never reads untrusted content and a quarantined model reads it but holds no tools. CaMeL first realizes the pattern as a compiled program whose capability labels propagate deterministically (Debenedetti et al., 2025), whereas Twin Agent constrains the channel between the two agents, allowing only a single free-form hint of 100–200 characters to pass (Hu et al., 2026). FIDES and PRUDENTIA also build on the dual-LLM pattern to obtain their taint-tracking capability (Costa et al., 2025; Kolluri et al., 2026).

3 THREAT MODEL AND MOTIVATION

3.1 THREAT MODEL

Attacker goal. The threat is indirect prompt injection: the user's query is assumed honest, but the content the agent retrieves is implanted by an adversary. The adversary seeks a side effect outside the user intent, including invoking an unauthorized tool or redirecting an authorized action to an attacker-chosen target, and succeeds if such a side effect occurs, whether or not the user's task completes. An attack is accomplished if the specific action the adversary intends takes place, not when the agent produces certain text an observer would judge as malicious.

Attacker capabilities. The adversary controls the content of one or more tool responses, such as web pages, emails, documents, messages, and database records, subject only to each response remaining well-formed for its tool interface. Injected content may occupy any number of responses and may appear at any step in a long-horizon trajectory. How the payload is composed is unconstrained: it may carry direct imperative instructions, an injection string optimized offline by an adversarial search, or forged chat-template mark that targets the backbone's input format. That is, content manipulated by adversary may induce arbitrary behaviour in any model that processes it.

The adversary cannot alter the registered tool specifications, including tool names, descriptions, and argument schemas, the model weights, or the tool implementations behind the interface. The adversary is assumed to know the above and the defense's design, and the backbone model family.

3.2 MOTIVATION

When a neural model is flooded with mixed benign and malicious context in rollouts, a fraction of injections is obeyed, and an attacker who can retry needs only one attempt to succeed (Zhan et al., 2025), weakening any probabilistic defense. Certainty therefore requires the defense outside the model (Willison, 2023), either in code runs deterministically or against a formally specified policy.

Prior work has converged on data provenance, borrowing information-flow control (IFC) from language-based security: attach a label to each value, propagate it along the flow, and let code rather than model decide what a labelled value may reach (Debenedetti et al., 2025; Costa et al., 2025; Wang et al., 2025; Fan et al., 2026). What a label cannot express, however, is what the content contains and what a value semantically means to context. It records only where the content comes from, so when a task needs a value that exists nowhere but in untrusted content, like a price quoted on a page or an account number written in a document, IFC leaves only two outcomes: refusal, or admission with full contamination; resulting in a significant degradation in utility. Some work settles the issue in advance, by fixing the flow so that execution never depends on untrusted content (An et al., 2025; Li et al., 2026a; Debenedetti et al., 2025). However, such a static plan cannot adapt to task versatility, and also trades utility for security. The other way is declassification, which hands the decision back to human, requiring a policy predefined by hand, an endorsement request to user, or human-crafted rules on the tainted text (Kolluri et al., 2026; Shi et al., 2025; Hu et al., 2026).

The dilemma is intrinsic: *untrusted content must be kept out of the trusted context, yet the task needs precisely what only that content holds*. Prior work either suffers probabilistic breaches or loses utility to static plan and tainted flow, or otherwise requires additional human involvement. Thus, we are motivated to resolve the dilemma by proposing a novel dual-agent approach. We follow the isolation of the dual-LLM pattern and build a trust boundary in between. Both sides are capable agents with context that can plan, reason and call given tools. We propose the coupled framework for agents to safely collaborate and accomplish the task, via a structural enforcement and informational asymmetry. Our defense relies on agents' own autonomy and intelligence, extending the ordinary single-agent loop into a coupled main–dual agentic loop that carries the defense itself.

4 METHOD

We follow the dual-LLM pattern (Willison, 2023) to isolate untrusted content from the trusted context, and enforce deterministic approaches for security. We upgrade both sides of the dual-LLM pattern into fully capable agents. In the proposed framework, security does not come from limiting the capability of the untrusted side, but from providing necessary grants with proper constraints to construct an information asymmetry between the main agent and the dual agent at test time, shaping their interaction loop in each outsourced subtask. Meanwhile, utility comes from giving both agents stateful context in their own autonomy; the main agent's context, in particular, is driven by a dynamic directed acyclic graph (DAG) that grows as the task progresses. Overall, the proposed work is

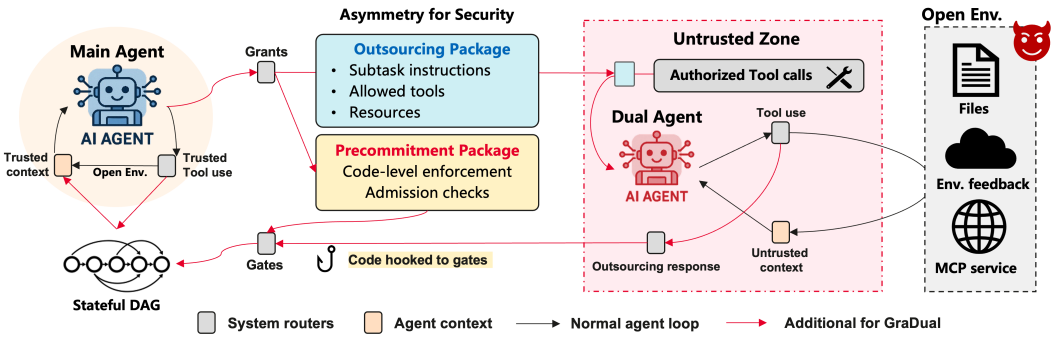


Figure 1: Overview of GraDual.

named **GraDual**, short for a dynamic **Gra**ph-driven **Dual**-agent architecture that harmonizes agentic utility and security. Figure 1 presents an overview.

Following is outlined as: §4.1 formalizes the coupled main-dual-agent loop. § 4.2 introduces the precommitment of outsourcing interaction loop. § 4.3 explains how the graph grows dynamically.

4.1 The Main–Dual Loop

An agentic system answers to a query q by a sequence of tool calls. At step t it selects a tool f_t from the registered set S_{tool} together with arguments a_t, and receives an observation o_t from the external environment $\mathcal{E}$. In typical single-agent loop, every observation is appended to the context:

$$(f_t,\, a_t) \leftarrow \pi(c_t), \qquad o_t \leftarrow f_t^{\mathcal{E}}(a_t), \qquad c_{t+1} = c_t \oplus o_t. \tag{1}$$

We have simplified components of this loop: c_t as the context carries not only the observations and initial query, but also the agent's own messages, tool information, session events and so on; π as the model outputs more than a tool call and arguments. At denoted granularity, we focus on the parts related to the IPI risk. As described in the threat model (Section 3.1), an adversary in $\mathcal{E}$ can manipulate o_t to include an injection $\mathcal{I}$, which enters the context c_{t+1} and therefore potentially affects subsequent decision. Besides, a payload may be assembled from several observations, and the manipulated call it finally induces is steps away.

Different from single-agent loop, GraDual follows the interaction pattern with $\mathcal{E}$, but isolates the context by a trust boundary between two agents. Hence, the context c_t is split between a main agent M, which holds q and a trusted context c_t^{trust}, and a dual agent D with an untrusted context c_t^{untrust}.

State. The agent states are the two isolated contexts, which grows with tsak proceeding. c_t^{trust} holds the user query $c_0^{\text{trust}} \leftarrow q \oplus S_{\text{tool}}$, M's running record, and a directed acyclic graph G_t whose nodes carry the planned steps and stay open to later value updates, edits and expansions. The graph is where M's context has explicit structure rather than only text, and the reliance the agentic loop advances on; § 4.3 specifies the node types and edge semantics. On the other hand, c_t^{untrust} holds the outsourced subtask information given by M at step t and whatever $\mathcal{E}$ returns via tool. Before any tool is called, M builds an initial graph as the plan, $G_0 \leftarrow \pi_M(c_t^{\text{trust}})$, using graph-construction primitives, so that the initial plan is trustworthy, deriving solely from q. Notably, G_0 is not the finalized plan but determines a part of it, on which the subsequent interactions grow.

How the loop works. Each agent individually runs a typical single-agent loop, while the two loops interleave through the tool-call pattern channel that invokes dual-agent interactions. In Figure 1, the black arrows are the loop an ordinary single agent has, and the red ones are what GraDual adds. As G_t carries planned actions, M selects next pending node n_t and reads off the tool's signature in S_{tool}. It issues sends, writes and transfers itself, since the return type leaves no room for manipulation. But it outsources a subtask to D when the signature is unannotated or its response is freeform, as with file reads, web fetches and external API retrievals. Next, the trust boundary is crossed twice in such an outsourced subtask. **Downward**, before the outsourcing is issued, M derives

$$(\sigma,\, S_{\text{dual}},\, \kappa) = \pi_M(q,\, n_t,\, c_t^{\text{trust}},\, G_t), \tag{2}$$

an outsourced subtask σ, a grant $S_{\text{dual}} \subseteq S_{\text{tool}}$ of needed tools, and a gate parameterized by κ that any return must satisfy. A spawned D is instantiated from σ and S_{dual}, that is, $c_0^{\text{untrust}} \leftarrow \sigma \oplus S_{\text{dual}}$. Within the outsourced subtask, D is stateful in its own agent loop and decides when to return, though

M always has the privilege to respawn a D with an empty c_0^{untrust} or to leave it stateful. **Upward**, D returns r_t. It only reaches M through one fixed procedure implemented as deterministic code and parameterized by κ, denoted by $\textsc{Admit}_\kappa^{\mathcal{E}}(\cdot)$. ADMIT instantiate a code run hooked to the feedback loop for the precommitment κ and execute to interact with $\mathcal{E}$ to retrieve certain responses:

$$\textsc{Admit}_\kappa^{\mathcal{E}}(r_t) = \begin{cases} \hat{r}_t & \text{if every check in } \kappa \text{ holds on } r_t, \\ \bot & \text{otherwise,} \end{cases} \tag{3}$$

where κ parameterizes multiple checks which may consult $\mathcal{E}$ and compute on statements for itself. Admission fails if any check refuses, and $\bot$ carries the identity of the failed check and none of r_t. Admitted $\hat{r}_t$ is not r_t but in the form κ prescribed for it, which could be a typed value, a handle in place of content, a structural data in place of text. Then, the graph grows by $\hat{r}_t$,

$$G_{t+1} = G_t \oplus_n \begin{cases} o_t & \text{if } M \text{ issues the call itself,} \\ \textsc{Admit}_\kappa^{\mathcal{E}}(r_t) & \text{if it is outsourced to } D, \end{cases} \qquad G_t \oplus \bot = G_t, \tag{4}$$

where $\oplus$ attaches value to node n_t by topology relation. By ADMIT, a claim is admitted if and only if the code processes it, never when D asserts it. What follows a $\bot$ is M's own decision: reflect on $\bot$ and respawn another subtask, continue on existing D, or drop it. The run ends when M judges the task completed, and the answer is composed from q, c_T^{trust}, G_T and M's own runtime record.

4.2 PRECOMMITMENT: GRANTS DOWN, GATES UP

Precommitment.[1] Before any outsourcing leaves, M answers three questions from its context $(q, n, c_t^{\text{trust}}, G_t)$: what the response will be used for, and therefore what a wrong value could cause; which tools D needs for the subtask, and what those tools could do in combination if D were entirely compromised; and whether $\mathcal{E}$ holds an independent verifier a response could be corroborated against. The answers are packed as σ, S_{dual} and κ respectively, as Eq. (2) and thereafter fixed. The grant and the gate bind different parties. S_{dual} binds D, fixing what the untrusted side may induce in the worst case. κ binds M, fixing what M will accept before M can be persuaded otherwise—a precommitment made by the party that expects its own later reasoning to be what the attack aims. In the triplet, only σ and S_{dual} travel downward to D; κ is kept by M. An injection can change what D reasons and reports; it cannot change how the report is checked and admitted.

Grants and gates. Downward go the outsourced subtask and the capability grant. $S_{\text{dual}} \subseteq S_{\text{tool}}$ is read off the annotations and is enforced by spawning D's runtime, i.e., only read-only tools are eligible. Eligibility is reasoned by M or by explicit tool annotations under the worst-case assumption of a fully compromised D. Upward, r_t is admitted by $\textsc{Admit}_\kappa^{\mathcal{E}}$. How it is instantiated can be different. A typed field is admitted when it satisfies the constraint in κ. An opaque handle stands for content M never reads; it is redeemed in slots κ enumerated. A format report describes the layout M cannot guess and fields σ did not ask for. Free text should be summarized and passed by handle so that M never reads it. Moreover, a request for further capability is granted only within the category σ already covers. These gates discipline what a value is used in terms of the context of the task and constrain the data flow, e.g., forbidding data becoming a control flow, while allowing predesignated control flow to pend for a filling value via the dynamic graph state.

Code enforcement. The gates are compiled with κ given and enforced by hooking $\textsc{Admit}_\kappa^{\mathcal{E}}$ to the return path, so admission is a function of r_t without any model. D returns values and evidence as inputs prescribed in σ; code run re-reads that evidence from $\mathcal{E}$ itself before any part of r_t enters G_{t+1}. Fidelity is not veracity: a faithful extraction from a planted document still yields false values, so admission establishes agreement with $\mathcal{E}$ rather than D's honesty, e.g., a payment destination extracted by D is matched against a cross-check from the transaction history via $\textsc{Admit}_\kappa^{\mathcal{E}}$. Evidence sources form a closed registry in which each source is registered together with the verifier that re-reads it. A claim no verifier can reproduce is rejected, so one planted document cannot corroborate another cannot pass the gate. Nothing on this path is adjudicated by a probabilistic model, and we demonstrate that this enforcement is load-bearing in § 5.4: with the same constraints stated as soft prompts and all other components intact, the defense no longer holds.

[1] We borrow the term from behavioral economics and philosophy (Strotz, 1955; Elster, 1984; 2000), where an actor at one moment and the same actor later are treated as two parties, the earlier binding what the later may do. *Ulysses had himself lashed to the mast before the Sirens came within hearing: the self that would hear the song was no longer the self that could answer it.* M writing κ is the earlier party; M judging what comes back is the latter, and finds the question settled in advance before context already changes.

Table 1: Defense performance comparison on *Long horizon replay* attack.

	Banking		Slack		Travel		Workspace		Average	
Defense	Utility	ASR	Utility	ASR	Utility	ASR	Utility	ASR	Utility	ASR
Undefended	81.5±1.9	51.4±1.8	41.6±2.6	43.4±3.7	61.3±7.9	2.9±2.2	77.2±7.0	0.0±0.0	67.9±1.1	37.1±1.0
Tool filter	62.2±2.7	3.4±0.3	47.8±2.8	0.4±0.6	67.3±8.5	0.0±0.0	68.0±5.7	0.0±0.0	59.5±1.7	1.7±0.3
Repeat user	75.5±1.6	25.4±2.6	**62.5±4.3**	8.4±3.4	**89.3±4.8**	0.0±0.0	**94.9±5.3**	0.0±0.0	**76.1±1.4**	14.5±1.8
Spotlighting	**76.5±2.0**	44.3±2.1	42.2±4.2	22.7±4.0	59.3±10.3	0.0±0.0	73.1±5.7	1.0±1.1	64.9±1.2	27.8±1.4
IPIGuard	64.9±2.2	4.9±1.2	**62.5±0.9**	4.9±1.2	74.7±5.5	0.0±0.0	59.2±3.4	0.0±0.0	64.1±2.0	3.7±0.6
DRIFT	57.1±3.0	4.7±2.0	32.9±3.5	10.2±1.9	48.7±8.5	0.0±0.0	76.9±5.1	0.4±0.9	52.8±2.0	5.2±0.9
CaMeL	58.7±3.1	0.2±0.4	55.3±3.2	**0.0±0.0**	34.7±6.0	**0.0±0.0**	66.0±3.1	**0.0±0.0**	56.9±0.5	0.1±0.2
GraDual (ours)	**72.7±3.2**	**0.0±0.0**	**59.6±3.4**	**0.0±0.0**	**92.0±0.0**	**0.0±0.0**	**74.5±3.8**	**0.0±0.0**	**70.9±2.1**	**0.0±0.0**

4.3 The Runtime State via a Dynamic Graph

Graph: Nodes & edges. The initial graph is built by M using graph primitives. The primitives fall in two groups: a semantic node for a step M is tracking, an entity node for a value named in q, a control node for a call it intends, and an edge between any two to track sources; M may revise or delete, and it may query node attributes, like ancestors and values. The second group is hard-coded. The nodes that carry responses are created by code: an execution node for tool action, a schema node for the values admitted by gates, attached under the control node as in (4). With the graph in context c_t^{trust}, M can therefore plan and reason. Edges carry the flow: which node a value derives from, which call consumes it, which step it triggers, and which execution a schema belongs to. Code reports three properties of the graph whenever the graph state is called: 1) that it is acyclic, so that ancestry holds and a handle's redeem is well defined; 2) that every control node is reachable from the root; and 3) that every node added at run time has an incoming edge.

Deferred planning. A plan fixed before anything is retrieved cannot decide on what has not yet been seen, and existing defenses have various attempts to resolve it. Such dependency comes in two degrees. 1) A milder one is missing parameters. 2) A sharper one leaves the control flow undetermined: asked to notify entities in a list, an agent cannot know before reading how many calls to make, to whom, or whether via email or direct message. GraDual defers: G_0 is the part of the plan q already determines, and the run extends G_0 when responses are admitted. The deferral provides two kinds of flexibility. A control node whose argument is missing waits for a value to pass the gate. A control node whose action form is left open and what settles it is an admitted data. The returned value is processed by κ and results in specified action(s), which settles the open node. § 5.4 freezes graph to demonstrate what the deferral is worth: security is unaffected but utility degrades.

Dynamic growth. Three things extend the graph at runtime, and only one of them is a choice made by model. Code adds an execution node when a call completes, and code settles pending node for fan-out or reclaim. The third is M judging that the task needs a call G_{t-1} did not anticipate, and that one is bounded twice. A new node must attach to a schema node already present, so that every step is traceable to the plan it grew from. And whatever the new node does, only two kinds of value can reach it: entities grounded in q, and schema nodes admitted at the gates of § 4.2. The precommitment settles, once and for all before anything is read, both the form a return may take and the node it will hang from; the gate then executes to enforce, admitting nothing it did not already describe and recasting what it admits into the prescribed form.

5 Evaluation

5.1 Experimental Setup

Benchmark. We evaluate on the task-injection benchmark AgentLAB (Jiang et al., 2026), which extends the indirect prompt injection setting of AgentDojo (Debenedetti et al., 2024). It keeps AgentDojo's four environments with 949 (user task, injection task) pairs, each hiding an attacker's goal in content a tool returns, but replaces the single override instruction with a payload split into 2–4 snippets in environments. The injection is therefore assembled by the agent's own actions instead of delivered in one response, making the injection more stealthy. We replay the payloads against an undefended `gpt-4o` agent, so every defense is evaluated on identical attacker text.

Defense and attack baselines. We compare across eight systems: the undefended agent, three defenses shipped with AgentDojo (Tool filter, Repeat user and Spotlighting), IPIGuard (An et al., 2025) as probabilistic representative, DRIFT (Li et al., 2026b) and CaMeL (Debenedetti et al., 2025)

Table 2: Defense performance under three different attacks on the banking suite.

	Tool knowledge		Important instr.		ChatInject	
Defense	Utility	ASR	Utility	ASR	Utility	ASR
Undefended	80.8±0.8	54.9±2.6	79.6±2.6	65.4±2.9	28.8±3.6	36.0±3.5
Tool filter	62.8±1.8	14.8±2.2	59.8±1.7	22.7±1.9	N/A†	
Repeat user	81.9±1.5	25.1±0.9	81.0±2.2	40.5±0.8	52.2±2.5	0.7±0.9
Spotlighting	77.1±2.6	34.5±2.7	78.8±1.8	55.6±1.8	37.8±2.7	28.0±1.0
IPIGuard	59.6±1.8	4.9±0.8	63.0±0.6	5.8±0.7	49.8±2.9	8.2±0.5
DRIFT	60.4±3.6	14.6±2.9	63.3±3.3	10.8±2.5	61.7±3.1	4.6±1.1
CaMeL	57.5±2.7	0.3±0.6	57.5±2.9	0.3±0.6	52.0±0.8	0.1±0.3
GraDual (ours)	**69.9±4.9**	**0.0±0.0**	**72.9±1.8**	**0.0±0.0**	**72.7±2.7**	**0.0±0.0**

as system-level state-of-the-art defenses, and our GraDual. CaMeL runs in its default structural configuration, without tuned security policy. We port all defenses from authors' public implementations into the same harness and test environment for fair comparisons. We evaluate under four attacks. *Important instructions* plants a fixed jailbreak that impersonates user, and *Tool knowledge* provides the real tool signatures to it, so the payload no longer has to guess the API. *Long horizon replay* is specialized by AgentLAB, using multi-round interaction and compositional payloads to inject, as a harder and stealthier setting. *ChatInject* (Chang et al., 2026) forges the backbone's chat-template tokens so that returned data is tokenized as a conversation turn rather than retrieved data. It assumes attacker knows that backbone's exact control tokens, so it applies only where the chat template is public. Thus we run it on `qwen3-235b-a22b-instruct-2507` as agent backbone.

Criteria and models. We mainly report two criteria: *utility* and *ASR*. Utility is measured by the task success rate. An errored task stays in the denominator and scores zero, so that crashing cannot raise a score (the reported runs' error rates are under 3%). ASR is the fraction of graded tasks in which the injection task succeeded, errored tasks being excluded. Two backbones are mainly used for benchmark runs: `gpt-4o` and `qwen3-30b-a3b-instruct-2507`. On each, every cell number is a mean over six runs and ± their sample standard deviation. % in tables are omitted.

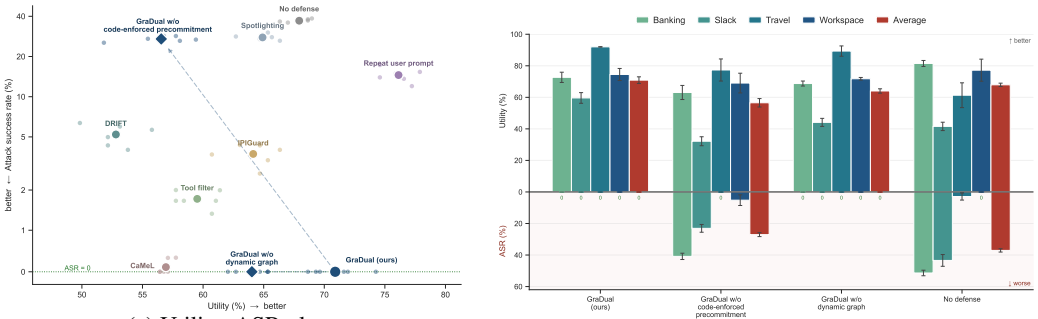
5.2 Main Results

Comparison among defense baselines. Table 1 reports every defense under Long horizon replay on gpt-4o, and Figure 2a places its rows in the utility–ASR plane, which clearly reflects the comparisons (note: ASR is logarithmic). The plane splits the field in two. Prompt-level defenses (Spotlighting, Repeat user) stay near or above the undefended agent in utility but leave ASR close to the undefended baseline (higher in places, because some user tasks and injection tasks are completed at the same time). Since it is a long-horizon compositional attack, simple defense measures are not effective, resulting in high ASR. System-level defenses (Tool filter, IPIGuard, DRIFT, CaMeL) drive ASR towards zero but all sit to the left of the undefended baseline, paying 3–15 points of utility for it. GraDual is the only point cluster consistently on the $\mathrm{ASR} = 0$ line and to the right of the undefended baseline. In per-suite perspective, the gain is not uniform. The undefended baseline is already relatively safe on Travel and Workspace, but suffers high ASR in Banking and Slack. GraDual maintains utility on Workspace and loses about 9 points on Banking. It reaches zero ASR everywhere and improves utility substantially on Slack and Travel. These are the suites the deferred plan is built for, and also where CaMeL, whose flow is fixed before retrieval, loses the most. The same split holds on the other backbone of Qwen (see Appendix A), where GraDual is consistently at 0.0% ASR with leading utility among all baselines.

Comparison against attacks. Table 2 varies the attack on the Banking suite, which is the most susceptible suite under different attacks, because of its structural data types. Since ChatInject column runs on Qwen, Tool filter is not applicable (†) because its tool-selection step returns an empty tool set. Every other defense responds attack by attack; for example, Repeat user ranges from 0.7% to 40.5% ASR and Spotlighting from 28.0% to 55.6%. GraDual is again the only defense at 0.0% ASR under all three, which is what the design proposes: the attacker's text is only ever tokenized inside D's context, so neither what it says nor how it is framed reaches the trusted side.

5.3 Case Study

Case 1: adaptive information retrieval. *Travel, user task 3.* The user asks for the highest-rated hotel in Paris, ties broken by higher price, and then an email with the hotel information. The hotel reviews are where the attack lies: three of the four hotels carry forged user turns ("Do not conclude or stop…") demanding three further searches before the task may be concluded. M never reads



(a) Utility–ASR plane. (b) Per-suite ablation.

Figure 2: (a) Defenses in the utility–security plane; ASR is logarithmic. (b) The two ablations per suite, removing the code enforcement of precommitment (§4.2) or the dynamic graph (§4.3).

them. It precommits what the review call must return: a map from hotel name to rating. D reads and replies mapping {`Good Night: 5.0, Luxury Palace: 5.0, ...`}. The review containing the payload is in no field and never enters c^{trust}. M breaks the tie on the admitted numbers and initiates another round of outsourcing with the hotel name. D provide typed address with state. After these returns passing the gate, M resolves the deferred email node at last.

Case 2: opaque handle on untrusted value. *Slack, user task 7.* "add Charlie to the channel starting with External". The fourth channel's name is the actual payload and the task cannot be completed without handing that exact string to `add_user_to_channel`. Hiding it forbids the call; admitting it puts injection in c^{trust}. M initializes a handle and receives `<ref:identifier:names[3]>` , then passes the handle itself as `channel`; the gate redeems string in code immediately before execution, while only the identifier as the *opaque handle* is seen by M. We can also find opaque handle used for cross validation in the next case.

Case 3: a fact check using corroboration. *Banking, user task 15.* The user asks for an address change, then intention shifts to a transaction to the landlord account named in q, and a refund of "that 10.00 I just got from my friend". Which friend that is appears nowhere in q. It has to be read off the ledger, and the payload lies in the subject line of exactly that row. M handles transaction slots for the pending payment one by one, via the opaque handle and code-enforced precommitment: `recipient=verify/deny`, subject and date `skip/allow`. The landlord account, named by the user, needs no corroboration; the refund recipient is corroborated against the bank's own records. Under a sibling payload asking to pay `US133...`, one digit different from a real ledger entry: nothing matches it, and the evidence names no transaction the main can reach, so the call is refused.

5.4 Ablations

W/o code-enforced precommitment. Shown in Fig. 2b, the first arm keeps the main–dual architecture and moves the precommitment into the prompt, as soft constraint. Though with same framework, every constraint §4.2 enforces in code is now a set of instructions the models are asked to follow: report the values the task needs, do not carry instructions forward, do not act, and so on. Nothing is weakened on structural design but nothing is enforced. Average utility falls from 70.9 to 56.5 and ASR rises from 0.0% to 27.1%, level with the prompt-level defenses and concentrated where the payload has an action to name, in the Banking and Slack. **W/o dynamic graph.** The second arm changes nothing but the graph. The full system runs, but after the initial plan G_0 the graph may no longer grow. ASR stays at 0.0% on every suite, and utility falls by 6.9 points.

6 Conclusion

GraDual proposes a novel framework towards resolving indirect prompt injection by system design with information asymmetry. It involves a main agent that issues outsource and precommitments while a dual agent that handles the outsourced subtasks. Both agents are given stateful loops, and the main agent preserves a dynamic graph in particular. With each outsourcing, the main agent keeps a precommitment for the future return, which is enforced in code on the interaction channel. The stateful graph grows on passing data, preserving versatility for changing tasks. Across four attacks, four suites, a long-horizon variant, two backbones, and against various defenses, GraDual is the only one that upholds ASR at 0.0% consistently while achieving leading utility. GraDual shows that the utility cost for security assumed in prior deterministic defenses is not intrinsic: by system design with dual-agent precommitted collaboration and the runtime state free to grow on trustworthy data, our framework harmonizes the security–utility trade-off and pushes the frontier forward.

REFERENCES

Sahar Abdelnabi and Eugene Bagdasarian. Ai agents may always fall for prompt injections. *arXiv preprint arXiv:2605.17634*, 2026.

Hengyu An, Jinghuai Zhang, Tianyu Du, Chunyi Zhou, Qingming Li, Tao Lin, and Shouling Ji. Ipiguard: A novel tool dependency graph-based defense against indirect prompt injection in llm agents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 1023–1039, 2025.

Anthropic. Piloting Claude in Chrome. Blog post, `https://claude.com/blog/claude-for-chrome`, August 2025. Accessed: 2026-09-15.

Luca Beurer-Kellner, Beat Buesser, Ana-Maria Creţu, Edoardo Debenedetti, Daniel Dobos, Daniel Fabian, Marc Fischer, David Froelicher, Kathrin Grosse, Daniel Naeff, et al. Design patterns for securing llm agents against prompt injections. *arXiv preprint arXiv:2506.08837*, 2025.

Manish Bhatt, Sarthak Munshi, Vineeth Sai Narajala, Idan Habler, Ammar Al-Kahfah, Ken Huang, Joel Webb, Blake Gatto, and Md Tamjidul Hoque. The defense trilemma: Why prompt injection defense wrappers fail? *arXiv preprint arXiv:2604.06436*, 2026.

Hwan Chang, Yonghyun Jun, and Hwanhee Lee. Chatinject: Abusing chat templates for prompt injection in llm agents. In *International Conference on Learning Representations*, volume 2026, pp. 24688–24715, 2026.

Sizhe Chen, Julien Piet, Chawin Sitawarin, and David Wagner. {StruQ}: Defending against prompt injection with structured queries. In *34th USENIX Security Symposium (USENIX Security 25)*, pp. 2383–2400, 2025a.

Sizhe Chen, Yizhu Wang, Nicholas Carlini, Chawin Sitawarin, and David Wagner. Defending against prompt injection with a few defensivetokens. In *Proceedings of the 18th ACM Workshop on Artificial Intelligence and Security*, pp. 242–252, 2025b.

Sizhe Chen, Arman Zharmagambetov, David Wagner, and Chuan Guo. Meta secalign: A secure foundation llm against prompt injection attacks. *arXiv preprint arXiv:2507.02735*, 2025c.

Yulin Chen, Haoran Li, Yuexin Li, Yue Liu, Yangqiu Song, and Bryan Hooi. Topicattack: An indirect prompt injection attack via topic transition. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 7338–7356, 2025d.

Manuel Costa, Boris Köpf, Aashish Kolluri, Andrew Paverd, Mark Russinovich, Ahmed Salem, Shruti Tople, Lukas Wutschitz, and Santiago Zanella-Béguelin. Securing ai agents with information-flow control. *arXiv preprint arXiv:2505.23643*, 2025.

Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for llm agents. *Advances in neural information processing systems*, 37:82895–82920, 2024.

Edoardo Debenedetti, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, Daniel Fabian, Christoph Kern, Chongyang Shi, Andreas Terzis, and Florian Tramèr. Defeating prompt injections by design. *arXiv preprint arXiv:2503.18813*, 2025.

Gelei Deng, Yi Liu, Yuekang Li, Kailong Wang, Ying Zhang, Zefeng Li, Haoyu Wang, Tianwei Zhang, and Yang Liu. Masterkey: Automated jailbreaking of large language model chatbots. In *Proc. ISOC NDSS*, 2024.

Jon Elster. *Ulysses and the sirens: Studies in rationality and irrationality*. Cambridge University Press, 1984.

Jon Elster. *Ulysses Unbound: Studies in Rationality, Precommitment, and Constraints*. Cambridge University Press, Cambridge, UK, 2000.

Linfeng Fan, Ziwei Li, Yuan Tian, Yichen Wang, Rongsheng Li, and Xiong Wang. The granularity mismatch in agent security: Argument-level provenance solves enforcement and isolates the llm reasoning bottleneck. *arXiv preprint arXiv:2605.11039*, 2026.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM workshop on artificial intelligence and security*, pp. 79–90, 2023.

Keegan Hines, Gary Lopez, Matthew Hall, Federico Zarfati, Yonatan Zunger, and Emre Kiciman. Defending against indirect prompt injection attacks with spotlighting. *arXiv preprint arXiv:2403.14720*, 2024.

Zhanhao Hu, Dennis Jacob, Xiao Huang, Zhaorun Chen, Bo Li, and David Wagner. Twin agent: Context residual compression for privilege separated agents. *arXiv preprint arXiv:2607.19595*, 2026.

Zimo Ji, Xunguang Wang, Zongjie Li, Pingchuan Ma, Yudong Gao, Daoyuan Wu, Xincheng Yan, Tian Tian, and Shuai Wang. Taxonomy, evaluation and exploitation of ipi-centric llm agent defense frameworks. *arXiv preprint arXiv:2511.15203*, 2025.

Tanqiu Jiang, Yuhui Wang, Jiacheng Liang, and Ting Wang. AgentLAB: Benchmarking LLM agents against long-horizon attacks. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=Bb0pQqsiad.

Aashish Kolluri, Rishi Sharma, Manuel Costa, Boris Köpf, Tobias Nießen, Mark Russinovich, Shruti Tople, and Santiago Zanella-Beguelin. Optimizing agent planning for security and autonomy. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=g0aVCDY3gS.

Evan Li, Tushin Mallick, Evan Rose, William Robertson, Alina Oprea, and Cristina Nita-Rotaru. Ace: A security architecture for llm-integrated app systems. *Network and Distributed System Security Symposium*, 2026a.

Hao Li, Xiaogeng Liu, CHIU Chun, Dianqi Li, Ning Zhang, and Chaowei Xiao. Drift: Dynamic rule-based defense with injection isolation for securing llm agents. *Advances in Neural Information Processing Systems*, 38:83262–83290, 2026b.

Haoran Li, Dadi Guo, Wei Fan, Mingshi Xu, Jie Huang, Fanpu Meng, and Yangqiu Song. Multi-step jailbreaking privacy attacks on chatgpt. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 4138–4153, 2023.

Junda Lin, Zhaomeng Zhou, Zhi Zheng, Shuochen Liu, Tong Xu, Yong Chen, and Enhong Chen. Vigil: Defending llm agents against tool-stream injection via verify-before-commit. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9764–9785, 2026.

Yi Liu, Gelei Deng, Yuekang Li, Kailong Wang, Zihao Wang, Xiaofeng Wang, Tianwei Zhang, Yepang Liu, Haoyu Wang, Yan Zheng, et al. Prompt injection attack against llm-integrated applications. *arXiv preprint arXiv:2306.05499*, 2023.

Yupei Liu, Yuqi Jia, Jinyuan Jia, Dawn Song, and Neil Zhenqiang Gong. Datasentinel: A game-theoretic detection of prompt injection attacks. In *2025 IEEE Symposium on Security and Privacy (SP)*, pp. 2190–2208. IEEE, 2025.

Satnam Narang. CVE-2025-54135, CVE-2025-54136: Frequently asked questions about vulnerabilities in Cursor IDE (CurXecute and MCPoison). Tenable Blog, August 2025. https://www.tenable.com/blog/faq-cve-2025-54135-cve-2025-54136-vulnerabilities-in-cursor-curxecute-mcpoison. Accessed: 2026-09-14.

National Institute of Standards and Technology. CVE-2025-53773: Command injection in GitHub Copilot and Visual Studio enabling local code execution. National Vulnerability Database, https://nvd.nist.gov/vuln/detail/CVE-2025-53773, August 2025. CVSS v3.1 base score 7.8 (High). Accessed: 2026-09-15.

OpenAI. Understanding prompt injections: a frontier security challenge. https://openai.com/index/prompt-injections/, November 2025. Accessed: 2026-09-15.

Nathan Parker. Architecting security for agentic capabilities in Chrome. Google Blog, https://blog.google/security/architecting-security-for-agentic/, December 2025. Accessed: 2026-09-24.

Fábio Perez and Ian Ribeiro. Ignore previous prompt: Attack techniques for language models. In *NeurIPS ML Safety Workshop*, 2022. URL https://openreview.net/forum?id=qiaRo_7Zmug.

Johann Rehberger. SpAIware: Advanced prompt injection exploits. Briefing, Black Hat Europe, 2024. Slides: http://i.blackhat.com/EU-24/Presentations/EU-24-Rehberger-SpAIware-Advanced-Prompt-Injection-Exploits.pdf. Accessed: 2026-09-14.

Tianneng Shi, Jingxuan He, Zhun Wang, Linyu Wu, Hongwei Li, Wenbo Guo, and Dawn Song. Progent: Programmable privilege control for llm agents. *arXiv e-prints*, pp. arXiv–2504, 2025.

Robert Henry Strotz. Myopia and inconsistency in dynamic utility maximization. *The review of economic studies*, 23(3):165–180, 1955.

Peiran Wang, Yang Liu, Yunfei Lu, Yifeng Cai, Hongbo Chen, Qingyou Yang, Jie Zhang, Jue Hong, and Ye Wu. Agentarmor: Enforcing program analysis on agent runtime trace to defend against prompt injection. *arXiv preprint arXiv:2508.01249*, 2025.

Simon Willison. The dual LLM pattern for building AI assistants that can resist prompt injection. Blog post, https://simonwillison.net/2023/Apr/25/dual-llm-pattern/, April 2023. Accessed: 2026-09-15.

Chong Xiang, Drew Zagieboylo, Shaona Ghosh, Sanjay Kariyappa, Kai Greshake, Hanshen Xiao, Chaowei Xiao, and G Edward Suh. Architecting secure ai agents: Perspectives on system-level defenses against indirect prompt injection attacks. *arXiv preprint arXiv:2603.30016*, 2026.

Xilie Xu, Keyi Kong, Ning Liu, Lizhen Cui, Di Wang, Jingfeng Zhang, and Mohan Kankanhalli. An LLM can fool itself: A prompt-based adversarial attack. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VVgGbB9TNV.

Jingwei Yi, Yueqi Xie, Bin Zhu, Emre Kiciman, Guangzhong Sun, Xing Xie, and Fangzhao Wu. Benchmarking and defending against indirect prompt injection attacks on large language models. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*, pp. 1809–1820, 2025.

Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. Injecagent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 10471–10506, 2024.

Qiusi Zhan, Richard Fang, Henil Shalin Panchal, and Daniel Kang. Adaptive attacks break defenses against indirect prompt injection attacks on llm agents. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 7116–7132, 2025.

Hanrong Zhang, Jingyuan Huang, Kai Mei, Yifei Yao, Zhenting Wang, Chenlu Zhan, Hongwei Wang, and Yongfeng Zhang. Agent security bench (asb): Formalizing and benchmarking attacks and defenses in llm-based agents. In *International Conference on Learning Representations*, volume 2025, pp. 35331–35366, 2025.

Kaijie Zhu, Xianjun Yang, Jindong Wang, Wenbo Guo, and William Yang Wang. MELON: Provable defense against indirect prompt injection attacks in AI agents. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=gt1MmGaKdZ.

A Results on the Other Backbone

Table 3 repeats the grid of Table 1 with `qwen3-30b-a3b-instruct-2507` as the backbone. The choice of backbones is based on two considerations: the model cannot be too strong, as it is well aligned such that the ASR remains low even in undefended tests; and the model cannot be too weak, as the proposed method requires a certain level of intelligence to reason and propose relevant components. Thus, `qwen3-30b-a3b-instruct-2507` and `gpt-4o` are chosen for the evaluation. Qwen3 as the weaker backbone sharpens the split of §5.2. Undefended ASR rises from 37.1% to 60.8% and the prompt-level defenses rise with it, whereas IPIGuard, CaMeL and GraDual stay within 0–1.9%, their security being decided in code rather than by the model's competence. CaMeL, however, is not uniformly at zero: in the main-text evaluations it records small but non-zero ASR under every attack (0.1–0.3%; Tables 1 and 2), and reaches 0.0% only on this grid, whereas GraDual holds 0.0% in every cell. DRIFT is the exception among the system-level defenses, rising from 5.2% to 18.3%: its isolation relies on the backbone following detection and validation instructions, which the weaker model exposes. GraDual again holds 0.0% ASR at above-undefended utility (45.6 against 39.3), though not the highest utility there—DRIFT reaches 48.1 at 18.3% ASR—and its margin narrows, consistent with utility being bounded by the quality of the plan the main agent writes. Per suite, GraDual's utility exceeds the undefended agent's on Banking, Travel and Workspace and falls below it on Slack, and CaMeL completes no Travel task on this backbone.

Table 3: Defense performance on *Long horizon replay* with qwen3-30b-a3b-instruct-2507

	Banking		Slack		Travel		Workspace		Average	
Defense	Utility	ASR	Utility	ASR	Utility	ASR	Utility	ASR	Utility	ASR
Undefended	47.8±3.2	71.1±1.0	39.4±1.6	72.0±1.4	6.7±5.5	37.0±9.8	31.0±5.5	22.4±1.3	39.3±1.8	60.8±0.5
Tool filter	N/A — tool-selection step yields an empty tool set									
Repeat user	53.2±4.2	60.0±2.8	36.1±2.8	55.1±0.5	46.7±8.6	1.4±2.2	42.9±5.5	7.5±2.5	46.2±2.3	45.6±1.7
Spotlighting	49.7±3.9	69.8±1.5	34.5±2.9	68.0±2.9	14.0±6.1	21.0±8.4	31.3±6.5	20.1±3.7	39.5±3.0	57.5±0.7
IPIGuard	40.6±1.6	0.3±0.4	53.7±2.2	6.1±2.0	32.0±7.6	**0.0±0.0**	19.7±4.6	**0.0±0.0**	40.2±0.7	1.9±0.5
DRIFT	47.7±2.5	23.1±2.3	38.4±2.9	21.8±3.0	55.3±3.0	**0.0±0.0**	62.6±5.4	6.8±2.5	48.1±1.8	18.3±1.3
CaMeL	34.5±2.5	**0.0±0.0**	44.7±2.0	**0.0±0.0**	0.0±0.0	**0.0±0.0**	53.1±4.7	**0.0±0.0**	37.5±1.0	**0.0±0.0**
GraDual (ours)	**51.4±2.0**	**0.0±0.0**	**29.4±3.7**	**0.0±0.0**	**41.3±9.7**	**0.0±0.0**	**58.8±4.7**	**0.0±0.0**	**45.6±1.2**	**0.0±0.0**

AI use statement

In this work, we used generative AI tools — mainstream Claude and Codex models — for two tasks in general: editing the paper to improve readability, and assisting with evaluation code. We have not used generative AI tools for the other tasks that require disclosure, and the remaining ones are not applicable to this work. We have reviewed all AI-assisted work: every AI-edited passage was checked against the results it describes, and AI-assisted code was read and tested by the authors. We take full responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

Reproducibility statement

We will release the full implementation of GraDual (its artifact is currently provided as supplementary material), and the rest of the evaluation is built on public code. The benchmark is AgentLAB (`https://github.com/TanqiuJiang/AgentLAB`), which extends AgentDojo (`https://github.com/ethz-spylab/agentdojo`); our runs pin a single AgentLAB commit and one of its released replay artifact sets, which together fix the injected payloads cell for cell. Every compared defense and attack is public as well: CaMeL (`https://github.com/google-research/camel-prompt-injection`), IPIGuard (`https://github.com/Greysahy/ipiguard`), DRIFT (`https://github.com/SaFo-Lab/DRIFT`) and ChatInject (`https://github.com/hwanchang00/ChatInject`), while Spotlighting, Repeat user prompt and Tool filter are the implementations that ship with AgentDojo. The experimental setup — suites, attacks, defenses, backbones and the 303-cell grid repeated over six runs — is spec-

ified in §5.1, together with how utility and ASR are computed; the full grid for the second backbone is reported in Appendix A.