WARDENGUARD AND WARDEN-BENCH: A MULTI-CHANNEL DEFENSE AND BENCHMARK FOR INDIRECT AND AGENTIC PROMPT INJECTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Prompt injection is the leading security risk for language-model agents, yet existing benchmarks cover narrow attack classes and existing detectors either flatten the agent's context into a single string or fine-tune on one attack distribution, causing high false positives or poor generalization. The proposed work makes two contributions. First, Warden-Bench, a 23,601-row benchmark consolidating twelve attack tiers (T0-T11), direct overrides, indirect injection via tool outputs and retrieved context, multi-turn fragmentation, and encoding evasion, plus a 5,899-row benign split for false-positive measurement. Second, WardenGuard, a defense exposing six trust-annotated context channels to a two-stage judge cascade: a local open-weight model settles clear cases, deferring to a frontier judge when an input is unfamiliar or escalation is needed, behind an attestation gate and a write-time memory-integrity monitor, preceded by a deterministic $\approx$2 ms decoder covering thirteen of fourteen encoding schemes, with every decision hash-chained for audit. The proposed work evaluates across Warden-Bench and four external benchmarks (41,949 cases total), WardenGuard attains mean F1 0.911 and precision 0.981 externally, and is the only defense meeting F1$>$0.90, FPR$<$5%, and FNR$<$15% on Warden-Bench. Across a 5×5 chained residual attack matrix (442,550 combinations), it reaches the lowest mean residual ASR (1.95%), significantly below every competitor on all twenty cells at 11.4$\times$ fewer false positives, a margin that persists under an open-weight judge backbone. Against a query-budgeted adaptive attacker, it admits 34.0% of a stratified held-out sample versus 68.7% and 70.0% for the strongest baselines, a separation that survives excluding every case the attestation gate refuses, indicating that trust-annotated context-channel separation is a practical, backbone-agnostic path toward deployable prompt-injection defenses. The code is available at: `https://anonymous.4open.science/r/WardenGuard-07F4/`

1 INTRODUCTION

Large Language-model (LLM) agents now read tool outputs, retrieve documents, and consult prior memory before acting on a user request [1, 2, 3]; consequently, each input channel constitutes a path through which an attacker can insert instructions into a region the agent treats as data. This attack class, *prompt injection* [4], ranks as the top-listed risk in OWASP's catalog of LLM application threats [5] and succeeds through tool responses [6], retrieved documents [7, 8], planning chains [9], and encoded payloads [10, 11]. Furthermore, published defenses consist of detector models [12, 13] or LLM-judge methods [14, 15]; both approaches flatten the agent's context into an unstructured string, thereby discarding information about which substring originated from the user, a tool, or memory, and both additionally exhibit either high false-positive rates (PromptSleuth at 48.6% on Warden-Bench) or recall collapse out of distribution (DataSentinel at 4.6% on InjecAgent).

The proposed work introduces WardenGuard, a defense that exposes the agent's context as six named channels with explicit trust labels and subsequently judges them via a two-stage cascade: a locally served open-weight model settles the clear cases it has standing to settle, while a frontier judge decides the remainder. Additionally, the cascade sits behind an attestation gate over the principals

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.

a case claims and a write-time monitor over the agent's memory; furthermore, it is preceded by a deterministic encoding decoder and appends every decision to a hash-chained audit log.

Contributions. The specific contributions of the proposed work is as follows:

1. The proposed work introduces *Warden-Bench*, a twelve-tier prompt-injection benchmark comprising 23,601 rows; 17,702 injection-positive across tiers T0 -T11 along with a 5,899-row benign set.

2. The proposed work presents *WardenGuard*, a trust-labeled multi-channel defense in which six named channels reach the judge with explicit UNTRUSTED, SEMI-TRUSTED, or TRUSTED labels, with the implicated `attack_channel` returned as a first-class JSON field; furthermore, the cascade is preceded by a deterministic encoding decoder covering thirteen of fourteen encoding families, yielding 19.1 pp of recall on the encoded tier.

3. A measured chained residual ASR matrix at scale is reported: specifically, the first 5×5 matrix over five defenses and five target LLMs, spanning $442{,}550$ combinations, on which WardenGuard's mean residual ASR of 1.95% is lowest on every target model and significantly lower across all twenty cells.

4. Additionally, replacing GPT-4.1-mini with the open-weight Qwen3-30B-A3B-Instruct [17] leaves the ranking unchanged and widens WardenGuard's MCC margin from $+0.233$ to $+0.264$, thereby demonstrating the robustness of the evaluation to judge choice.

Section 2 reviews related work, Section 3 describes Warden-Bench, Section 4 the WardenGuard architecture, Section 5 the experiments and Section 6 the limitations. The appendices carry the full tables, figures and protocols.

2 Related Work

The literature contains related work in several directions. Detector models [12, 35, 13, 36] and activation-based task-drift detection [54] score a text field or the protected model's own state, whereas StruQ, SecAlign, and Jatmo [37, 38, 53] instead train the agent to ignore embedded instructions. Additionally, LLM-judge defenses [14, 15] prompt a frontier model over a concatenated context, while Spotlighting [16], the closest prior work on provenance, carries a single bit and rewrites the protected model's own prompt; consequently, its efficacy remains a property of that model.

CaMeL [56] and information-flow control [55] place the separation outside the model, albeit at the cost of rewriting the application, and Zverev et al. [52] measure the premise on which both lines rest. Furthermore, SCOUT [33] treats defense as per-input detector allocation, a premise that WardenGuard's cascade shares. BIPIA, InjecAgent, AgentDojo, and PoisonedRAG [7, 6, 9, 8] target indirect injection, whereas Warden-Bench additionally introduces twelve tiers, including encoding evasion, across 23,601 rows.

Multi-channel input [18], holistic judging [34], and explicit trust labels [16] each appear in earlier work; however, no prior defense combines multi-channel trust-labeled input with a deterministic decoder, attestation before detection, write-time memory gating, and a local-first judge cascade. Appendix E extends the discussion further.

3 Warden-Bench

Constructed for this work, Warden-Bench is a 23,601-row prompt-injection benchmark designed to fill three gaps in existing public benchmarks: (i) consolidated coverage of direct, indirect, and encoded attacks in a single split; (ii) a balanced benign tier large enough to measure false-positive rate at single-percentage-point resolution; and (iii) tier-level decomposition that lets a defense be inspected attack class by attack class.

Table 1: Warden-Bench tier taxonomy.

Family	Tier	Attack	Benign	Total	Attack class
Direct	T0	360	31	391	override / system forgery
Indirect	T1	1,736	285	2,021	retrieved-context override
Multi-turn	T2	1,039	231	1,270	memory-poisoning temporal
Indirect	T3	1,519	87	1,606	tool-output directive
Direct	T4	183	140	323	multi-task injection
Indirect	T5	2,000	0	2,000	code-embedded payload
	T6	2,498	0	2,498	multi-channel simultaneous
	T7	2,500	0	2,500	inter-agent directive
	T8	2,000	0	2,000	tool-permission escalation
Multi-turn	T9	1,958	0	1,958	cross-turn fragmentation
Technical	T10	509	0	509	interpreter-layer exploit
Encoded	T11	1,400	465	1,865	14 encoding families
Benign	BEN	0	4,660	4,660	benign NLP tasks
Total		**17,702**	**5,899**	**23,601**	

T11 covers 14 encoding families: base64, hex, ROT13, Caesar, leetspeak, mixed_case, reversed, unicode_escape, url_encoding, zero_width, base85, base32, atbash, ROT47.

3.1 Threat model

We adopt the agentic-LLM threat model used by InjecAgent and BIPIA [6, 7], wherein an agent A executes on behalf of a user U and consumes a structured context $\mathcal{C}$ comprising the union of six channel types:

$$\mathcal{C} = \{c_{\text{UI}}, c_{\text{TOOL}}, c_{\text{MEM}}, c_{\text{HIST}}, c_{\text{RAG}}, c_{\text{FILE}}\}, \tag{1}$$

denoting user input, tool output, agent memory, conversation history, retrieved context, and file metadata. The adversary $\mathcal{A}$ controls a strict subset $\mathcal{C}_{\mathcal{A}} \subset \mathcal{C}$; notably, tool outputs and retrieved context are the most exposed, since a malicious search result, a poisoned email, or a crafted attachment lands in $\mathcal{C}$ under the agent's reading. Furthermore, the user prompt c_{UI} is adversarial in the direct-injection setting [4, 19] but trusted in the indirect-injection setting [18]. An injection succeeds when an instruction $\iota \in c_i$ for some $c_i \in \mathcal{C}_{\mathcal{A}}$ causes A to deviate from the user task τ_U and instead execute ι's task τ_ι. Positioned between $\mathcal{C}$ and A, the defense gate returns a binary verdict $v \in \{0, 1\}$. We adopted three assumptions consistent with prior work [13, 15]: the defense judge is not itself the adversary's target; the payload resides in the channel data passed to the defense rather than in side channels such as timing or network state; and the defense and agent share the same success criterion, namely a behavior-overriding instruction that causes deviation from τ_U. Section 6 discusses the BIPIA definitional gap.

3.2 Tier Taxonomy

Warden-Bench contains twelve injection tiers (T0 through T11) plus a benign tier (BEN); Table 1 lists each tier with its row count and a one-line description. At a higher level, the twelve tiers group into five families: *direct* (T0, T4) places the override in the user turn; *indirect* (T1, T3, T5–T8) places it in a non-user channel—retrieved context, tool output, embedded code, several channels at once, a spoofed inter-agent message, or a tool-permission grant; *multi-turn* (T2, T9) splits it across conversation turns; *technical* (T10) carries an interpreter-layer exploit—SQL, deserialization, XXE, or command syntax—in a single user turn; and *encoded* (T11) carries the override through any of fourteen encoding families: base64, hex, ROT13, Caesar, leetspeak, mixed-case, reversed, Unicode-escape, URL-percent, zero-width, base85 (RFC 1924/Z85), base32 (RFC 4648), atbash, or ROT47. These families group by primary delivery channel and are not disjoint on the temporal axis, since every T3 attack row and half of T11's also carry `temporal_mode = multi_turn`.

3.3 Construction and Use

Every attack payload was written for Warden-Bench by a tier-specific generator, seeded in part from five public prompt-injection collections and recorded per row in the `seed_source` column; furthermore, labels are assigned at construction by those generators rather than post hoc, and an

independent re-classification by `gemini-3.7-flash` agrees with them on 91.9% of rows (Cohen's κ=0.789). Consequently, the 17,702 injection-positive and 5,899 benign rows serve as the denominators for every rate reported below; additionally, Appendix G reports the construction, the mutation layer, the label audit, and the benign accounting in full.

4 WARDENGUARD

WardenGuard comprises a pipeline of six mechanisms applied to every case in a fixed order (Fig. A1): a memory-integrity monitor over the writes the agent would persist (Sec. 4.7); an attestation gate over the principals the case claims (Sec. 4.6); a deterministic encoding decoder (Sec. 4.2); a two-stage judge cascade in which a locally served open-weight model settles the clear cases while a frontier judge decides the rest (Sec. 4.4), both stages receiving the six context channels as labeled regions and returning a verdict, a confidence score, and a localized attack channel; a standing check that withholds the local stage's authority to clear a case whose context does not resemble the traffic it was calibrated on (Sec. 4.5); and a hash-chained audit log with content-addressed rollback (Sec. 4.8). Notably, the order is not incidental: the monitor decides which memory writes persist before any judge reads the state, and the gate refuses unverifiable principals without spending a judge call at all, so both run ahead of detection. Additionally, the decoder and the judge prompt are described first since the cascade's two stages share them.

4.1 SIX-CHANNEL TRUST TAXONOMY

The agent runtime is required to expose its working context as six named regions, and Table A4 lists the channels along with their default trust levels. It should be noted that trust levels reflect typical agent topology rather than ground truth. Specifically, the `agent_memory_state` channel carries scratchpad state that the agent itself wrote and read, whereas `retrieved_context` and `file_metadata` carry data whose contents are determined by parties outside the trusted boundary.

4.2 DETERMINISTIC ENCODING DECODER

The encoding-evasion class (T11 in Warden-Bench) appends an attack string in one of fourteen encodings: base64, hex, ROT13, Caesar with non-13 shift, leetspeak, mixed-case, reversed, Unicode-escape, zero-width, URL-percent, base85 (RFC 1924/Z85), base32 (RFC 4648), atbash, and ROT47. While modern frontier LLMs read some of these natively (URL-percent in particular), encodings such as base64, hex, base85, base32, atbash, and ROT47 nonetheless evade most LLM-judge defenses by pushing the malicious string out of the model's training distribution [10, 11].

4.3 HOLISTIC JUDGE PROMPT

Decoded channels are concatenated into a single prompt (detailed in appendix B) and submitted to the frontier model with τ=0, max_tokens=100; each channel is truncated to 1,500 characters and the agent task description to 500. The model emits JSON of the form:
{`"verdict":"injection"|"benign", "confidence":` 0.0–1.0,
`"attack_channel":"name"|"none"`}
A single prompt per stage avoids the OR-aggregation FPR penalty that arises when six independent calls, each with FPR p, are aggregated to $1-(1-p)^6 \approx 6p$. Both stages of the cascade receive this prompt, and a stage's score is the probability mass the model places on the injection value of the `verdict` field, read from the top-five token log-probabilities at that position; consequently, the same prompt yields a calibrated number rather than a bare label, while the JSON fields are retained for the attack channel.

4.4 TWO-STAGE JUDGE CASCADE

Two judges run the decoder and the prompt above and differ only in backbone. Stage 1 is Qwen3-30B-A3B-Instruct-2507, an open-weight mixture-of-experts model with 3B active parameters, served locally by Ollama in 4-bit quantization, whereas Stage 3 is GPT-4.1-mini accessed through an API. Specifically, Stage 1 settles a case when its score is confidently low, in which case the case is allowed, or confidently high, in which case it is blocked; a score in between, however,

escalates the case to Stage 3, whose verdict is final, since there is no stage after it. Furthermore, an errored Stage 1 call escalates, while an errored Stage 3 call blocks the case and marks the record invalid, so a transport failure can never read as an allow, and invalid records are re-run rather than scored. Cost is counted as one unit per Stage 1 call and one hundred per judge call, reflecting the ratio of the two models' per-token prices. On Warden-Bench, 76% of cases settle at Stage 1, 15% are refused by the gate before any judge runs, and 9.1% reach GPT-4.1-mini (Sec. 5.2); this is where the frontier-judge cost saving originates, namely from the local model rather than a smaller prompt. The defense's first design sent every case to the frontier judge directly; Appendix D reports what that cost and why it was replaced. The cascade outperforms that single-frontier-judge design on every detection metric ($+0.0245$ F1) at $10.0\times$ lower mean cost per case; Appendix D reports both arms.

4.5 Standing to Clear

Stage 1's score answers whether a case looks like an injection; however, it does not answer whether the model has any basis for that judgment on this particular input, and the two questions come apart out of distribution. Measured across the five benchmarks, the cases Stage 1 scores as most confidently benign are 10.4% attacks on Warden-Bench and 90.5% attacks on BIPIA: consequently, out of distribution, the stage's confident clears invert rather than merely degrading, while its confident blocks continue to transfer. Furthermore, no threshold on the score can separate the two populations, because the cases in question carry identical scores.

4.6 Agent Passport and Attestation Gate

Directives arriving through the inter-agent, tool, and memory channels claim a principal: an agent id and a capability such as `execute_tool`. WardenGuard verifies the claim before it reads the content. Specifically, a passport binds an agent id to an Ed25519 key and a monotone capability set, and the gate returns Refuse when any claimed principal fails verification, which blocks the case at zero judge cost and runs no detector; Pass when every claim verifies; or No-Claim when the case carries only plain content channels. Since a refusal does not depend on the text, rewriting a directive cannot mint a signature, and consequently residual attack success on claim-carrying classes is structural rather than a matter of attacker effort. Because Warden-Bench carries no signing infrastructure, every number involving the gate is disclosed as a simulation of attestation rather than a deployed one (Sec. 5.2).

4.7 Memory Integrity Monitor

An agent's persistent memory constitutes the one channel an attacker can write once and read many times, so WardenGuard gates writes rather than reads. Each write carries an integrity label from the lattice Trusted $>$ Semi-trusted $>$ Untrusted, namely the meet of its source channel's label and its writing principal's label. Specifically, the monitor enforces two rules: **NUIPM**, whereby no instruction-bearing record below the policy floor persists in memory— refused outright rather than sanitized, since a sanitizer is itself a second detector with its own error rate—and **BGAD**, which anchors on the system prompt and refuses a write whose post-commit drift exceeds a fixed threshold, thereby catching accumulations of individually innocuous writes. Because Warden-Bench assigns every write Untrusted—the benchmark records what the agent would write but not where it came from—the monitor's benign refusal rate reflects that of the strictest provenance setting and is accordingly reported beside the detection numbers rather than folded into them; additionally, Appendix G discuss the lattice, the instruction-bearing scorer, and the enforce/observe modes.

4.8 Hash-Chained Audit Log and Rollback

Every decision the pipeline makes is appended to an audit chain whose entries are linked by hash—SHA-256 over the previous entry's hash and the canonical JSON of the record—so an edit to any past entry breaks every later hash and is caught by a single verification pass. Furthermore, memory writes are transactional and states are stored content-addressed, so a rollback restores an earlier committed address and is itself recorded on the chain; because the monitor's rules run inside the transaction, every state to which a rollback can return satisfies the memory invariant. The audit log

Table 2: Warden-Bench overall performance, with confusion counts. WardenGuard is the only defense on the F1/FPR Pareto frontier.

Defense	F1	MCC	Bal. acc.	Prec.	Recall	FPR	TP	FP	FN	TN
ProtectAI	0.2573	−0.0302	0.4871	0.7204	15.7%	18.24%	2,773	1,076	14,929	4,823
DataSentinel	0.7549	0.0262	0.5131	0.7567	75.3%	72.69%	13,333	4,288	4,369	1,611
PromptArmor	0.8356	0.6047	0.8473	**0.9863**	72.5%	**3.02%**	12,831	178	4,871	5,721
PromptSleuth	0.8591	0.4051	0.6945	0.8438	87.5%	48.60%	15,489	2,867	2,213	3,032
WardenGuard	**0.9543**	**0.8381**	**0.9415**	0.9849	**92.5%**	4.25%	16,383	251	1,319	5,648

constitutes an engineering property that the other mechanisms rely on, rather than a detection claim. Additionally, Appendix G presents the transaction and rollback semantics.

5 Experiments

5.1 Evaluation Setup

WardenGuard is evaluated on Warden-Bench and, zero-shot, on four external benchmarks—PS-Bench, DS-Bench, InjecAgent, and BIPIA, comprising 18,348 cases—against ProtectAI, DataSentinel, PromptArmor, and PromptSleuth. Detection is reported as F1, MCC, recall, and false-positive rate, with MCC serving as the balanced measure under the benchmark's 75/25 class split. Additionally, Appendix G presents the benchmark descriptions, baseline configurations, and metric definitions.

5.1.1 Held-Out Protocol

Warden-Bench ships a frozen 70/30 development/test partition, stratified on tier × label × mutation under a fixed seed, and every configuration choice was made on the development split. Notably, WardenGuard's held-out F1 is within 0.1 pt of its development figure, the ordering of all five defenses is identical on the two splits, and no defense's development/test difference is significant after correcting for the five comparisons (WardenGuard: p=0.76). It should be noted that the split is not a train/test split: no component of any defense is trained or fine-tuned on Warden-Bench, on the development rows, or on any other, and every verdict is a fresh model call—consequently, since the development split was used only to choose WardenGuard's configuration, the held-out column carries no selection effect. Additionally, Appendix G details the protocol in full alongside Table A7.

5.2 Warden-Bench Results

5.2.1 Overall Detection

On Warden-Bench, WardenGuard leads on F1 (0.9543) and on MCC (0.8381, against 0.6047 for the next-best defense), with 92.5% recall, 98.5% precision, and a 4.25% false-positive rate. Specifically, a case counts as defended when a judge blocks it or when the monitor refuses the memory write that carried its payload, verbatim or in an encoding the decoder recognizes: the monitor stopped the payload at the write path in 2,713 attacks, 176 of which the judges had not blocked, whereas the judge verdict alone gives F1 0.9489 (recall 91.5%) at the same false-positive rate, since a refused write never blocks a case. Furthermore, the observe arm measures what enforcement is worth: with every write allowed to persist and the judges reading the poisoned state, the defense reaches F1 0.9532 (recall 92.4%) at the same false-positive rate. On the 4,290 attacks whose memory the monitor changed, enforcement defends 99.3% against 98.6% for the judges alone, gaining 54 attacks whose payload the judges had missed in memory and losing 20 whose refused memory entry had been the judges' evidence. Among benign cases that write to memory, 10.5% have at least one write refused; consequently, that rate is treated as the monitor's cost and is reported separately from the FPR. Meanwhile, PromptArmor leads on FPR (3.02%) but loses 11.9 pp of F1 by missing 3,552 more attacks, and PromptSleuth's recall (87.5%) is 5.0 pp below WardenGuard's, operating at 48.60% FPR—11.4× WardenGuard's which is the trade-off. Table 2 makes explicit.

Table 3: WardenGuard across five benchmarks.

Benchmark	n	F1	Prec.	Recall	FPR	FNR
Warden-Bench	23,601	0.9543	0.9849	92.5%	4.25%	7.45%
PS-Bench	12,700	**0.9703**	0.9587	**98.2%**	4.46%	**1.78%**
DS-Bench	1,200	0.9296	0.9851	88.0%	4.00%	12.00%
InjecAgent	2,108	0.9444	0.9816	91.0%	**1.71%**	9.01%
BIPIA	2,340	0.7994	**0.9980**	66.7%	3.33%	33.33%
Mean (4 ext.)	—	0.9109	0.9808	86.0%	3.38%	14.03%

5.2.2 PER-TIER RECALL

WardenGuard leads on six of the twelve attack tiers and is weakest on T9 (cross-turn fragmentation, 69.9%) and T10 (interpreter-layer exploits, 89.8%); additionally, Appendix G gives per-tier recall for all five defenses and the per-tier decomposition of where WardenGuard's decisions are made.

5.2.3 BENIGN FPR

WardenGuard's combined benign false-positive rate is 4.25% (Table 2), representing a 44.35 pp gap to PromptSleuth; furthermore, Appendix G decomposes it by slice.

5.3 CROSS-BENCHMARK GENERALIZATION

All six mechanisms of Sec. 4 are active; namely, the memory-integrity monitor, the attestation gate, the deterministic decoder, the two-stage judge cascade, the standing check governing local clears, and the audit log in the configuration designed against Warden-Bench and used unmodified on four external benchmarks. Table 3 reports the per-benchmark performance. Specifically, WardenGuard achieves F1 ≥ 0.93 on three of the four external benchmarks (PS-Bench, DS-Bench, InjecAgent), for a mean of 0.911 across the four; furthermore, mean precision is 0.981, so the false-positive rate stays low across all four distributions without per-benchmark tuning.

5.3.1 CONTROLLED CROSS-DEFENSE COMPARISON

WardenGuard leads InjecAgent by 27 pp F1 over the next-best defense and is the only defense with FPR $\leq 5\%$ and recall $\geq 60\%$ on both InjecAgent and BIPIA. Additionally, Appendix G gives the per-benchmark tables, baseline provenance limits, and DataSentinel's failure modes.

5.4 CHAINED RESIDUAL ASR

The paper's primary empirical contribution is reported here: a fully measured 5×5 matrix of residual attack-success rate over five defenses and five target LLMs on Warden. Specifically, the matrix covers the full 17,702-attack subset of Warden, giving $17{,}702 \times 5 \times 5 = 442{,}550$ attack-defense-LLM combinations. Furthermore, the matrix is built from two independent measurements joined per `test_id`: the defense's binary verdict and the target LLM's binary attack-success judgment, and Eq. equation 6 is applied at the cell level. Notably, no new API calls are required, since each input is already collected as part of the per-defense and per-LLM evaluations. Additionally, the five target LLMs span a representative range of agent-class backbones: GPT-OSS-120B [25], Nemotron-3-Super-120B [26], Gemma-4-31B [27], Llama-3.1-70B [28], and Mistral-Small-3.1 [29]. Fig. A4 sketches the pipeline.

5.4.1 CELL-LEVEL RESULTS

Table 6 reports the full 5×5 matrix and the per-defense mean. WardenGuard's mean residual ASR is 1.95%, representing a 20.03 pp absolute reduction (91.1% relative) over the no-defense baseline of 21.98%, and correspondingly the lowest mean in the study on every one of the five target LLMs.

Table 4: Pairwise residual ASR, McNemar exact with Holm correction.

Competitor	FPR	WardenGuard lower	Tied	Competitor lower
PromptArmor	**3.02%**	**5**	0	0
ProtectAI	18.24%	**5**	0	0
PromptSleuth	48.60%	**5**	0	0
DataSentinel	72.69%	**5**	0	0
Total	—	**20**	0	0

Table 5: Component ablations against complete WardenGuard.

Arm	F1	MCC	Recall	FPR
Full benchmark (n=23,601)				
WardenGuard(complete)	**0.9543**	**0.8381**	**92.5%**	4.25%
Without encoding decoder	0.9475	0.8229	90.9%	**2.95%**
Without trust annotation	0.9486	0.8280	90.9%	**2.27%**
Neither	0.9371	0.8005	88.6%	**1.63%**
Encoding decoder, T11 only (n=1,865)				
With decoder	0.9480	0.8155	91.9%	5.81%
Without decoder	0.8390	0.6145	72.8%	2.15%

Table 6: Residual ASR per defense and target LLM.

Defense	GPT-OSS-120B	Nemotron-3	Gemma-4-31B	Llama-3.1-70B	Mistral-S-3.1	Mean
No defense	13.77%	18.45%	21.28%	22.12%	34.27%	21.98%
ProtectAI	11.30%	15.01%	18.56%	18.05%	29.76%	18.54%
DataSentinel	2.49%	3.21%	3.87%	3.91%	6.03%	3.90%
PromptArmor	5.01%	5.78%	7.52%	6.93%	11.00%	7.25%
PromptSleuth	2.78%	2.92%	3.67%	3.14%	4.16%	3.34%
WardenGuard	**1.45%**	**1.61%**	**2.15%**	**1.93%**	**2.62%**	**1.95%**

5.5 Ablations

Both components are ablated against the complete defender of Table 2, singly and together, on the same 23,601 rows, with the band, the standing check, the attestation gate, and the memory monitor active in every arm. Notably, each ablation changes the prompt that *both* judge stages read, since an arm that ablated only the frontier stage would leave the local stage reading a different document from the one it escalates. Table 5 summarizes the arms; specifically, removing either component costs F1 and MCC, removing both costs more than either alone, and the decoder's contribution concentrates on T11, where turning it off drops F1 from 0.9480 to 0.8390. Additionally, Appendix G gives the per-arm and per-tier breakdown.

5.5.1 Trust Annotation and Encoding Decoder

The trust annotation is ablated by deleting the three-level definition and the per-channel (TRUST) suffix from the prompt, leaving the six named channel regions, the ordering, the truncation, and the token budget unchanged; furthermore, the ablated prompt is derived from the canonical one by deletion rather than rebuilt, and a startup check re-derives it and fails if anything else moved. Furthermore, the decoder is ablated by withholding the decode pass, with an empty decode notice so the prompt shape remains unchanged and only the decoded content is withheld. On the 1,865 T11 rows, where the decoder is the difference between reading a payload and reading a blob, it is worth $+19.1$ pp recall and $+0.2010$ MCC (p=2.2×10^{-58}, 293 attacks corrected against 26 broken). Benchmark-wide, it contributes $+0.0068$ F1, $+0.0152$ MCC, and $+1.6$ pp recall (p=2.9×10^{-18}).

5.5.2 Backbone Portability

Replacing the frontier judge with the open-weight Qwen3-30B-A3B-Instruct [17] preserves the ranking: WardenGuard leads on F1 and on MCC under both backbones, and all four within-backbone margins are significant under McNemar's exact test with Holm correction. Consequently, the MCC margin over the nearest baseline widens from $+0.233$ to $+0.264$, and WardenGuard's false-positive rate moves least of the three defenses, by 2.8 pp against 4.9 pp for PromptArmor and 14.0 pp for PromptSleuth; additionally, Appendix G gives the tables, the per-tier movement, and the output-format compliance screen that governs backbone selection.

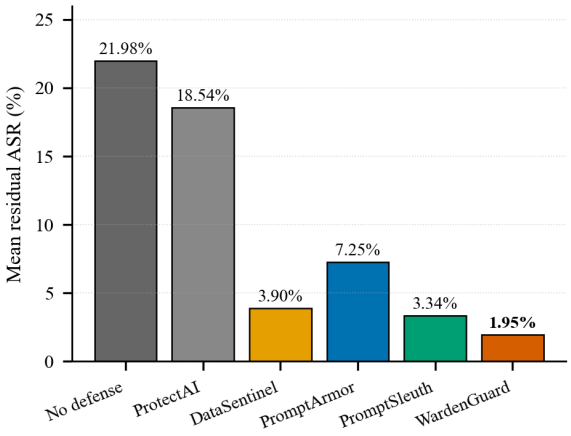

Figure 1: Mean residual ASR per defense.

6 Limitations and Discussion

Failure Modes WardenGuard's recall on T9 (cross-turn fragmentation) is 69.9%, the lowest of the twelve injection tiers. A per-case audit of the 697 misses (Appendix C) attributes 78.2% to a definitional mismatch; Warden-Bench labels under the deviation criterion of Sec. 1, whereas the holistic prompt flags only content that genuinely attempts to override agent behavior and 12.1% to genuine detector misses; the residual is thus definitional rather than architectural [30, 31]. Correspondingly, PromptSleuth's looser intent-based definition reaches 77.2% on T9, but at 48.6% FPR overall. The per-mutation decomposition of Table A10 separates this confound directly, and we recommend that benchmarks exposing a mutation layer report it. Similarly, WardenGuard's 60.1% recall on BIPIA and 54.7% on DS-Bench's grammar-correction task are definitional, since BIPIA counts style directives (e.g., "translate to Spanish") as injections and the DS-Bench task is semantically near-identical to its injection; Appendix G details both breakdowns.

Deployment Considerations The cascade bounds cost structurally: Stage 1 settles cases locally, so only 9.1% reach the frontier judge - 11$\times$ fewer calls than querying it on every case; though this share rises to 45 - 68% on external benchmarks, where accuracy is bought with frontier calls rather than saved. Component choice is a further lever: removing the decoder, the trust annotation, or both lowers benign FPR to 2.95%, 2.27%, or 1.63% against 4.25%, and all four configurations still clear the F1 >0.90, FPR $<5\%$, FNR $<15\%$ bar; yet the decoder yields $+19.4$ pp recall on T11 and the annotation $+10.9$ pp on T4, so a reduced configuration only narrows the published weakness. As strong adaptive attacks bypass most published defenses [32], App. A evaluates WardenGuard under that regime, leaving gradient-based attackers open. We thus claim a production-ready architecture, not a deployed one; Appendix G details the full cost accounting and trade-off.

7 Conclusion

The proposed work presented WardenGuard, a prompt-injection defense that combines a trust-labeled multi-channel prompt, a deterministic encoding decoder, an attestation gate, a write-time memory monitor, and a local-first judge cascade under a hash-chained audit log, together with Warden-Bench, the twelve-tier benchmark on which it is measured. The paper's primary empirical contribution is a measured 5×5 chained residual ASR matrix over five defenses and five target LLMs, on which WardenGuard is significantly lower than every competitor in all twenty cells while operating at a false-positive rate a deployment can absorb. Furthermore, the deterministic decoder covers thirteen of the fourteen encoding families end to end, and the ranking survives replacing the frontier judge with an open-weight backbone. Future work includes optimization-based adaptive attacks against the fixed prompt template, definitional alignment between benchmark labeling and defense criteria on mutated tiers, an output-style sub-detector for BIPIA, and direct AgentDojo evaluation.

AI USE DISCLOSURE

Large language models were used to assist with grammar refinement and manuscript editing, to generate synthetic attack payloads for Warden-Bench under author-written templates and labeling rules, and to support research execution (monitoring evaluation runs, reviewing code, and producing figures and tables from measured values). All scientific content, experimental results, and conclusions are the sole work of the authors.

ETHICS STATEMENT

Warden-Bench contains prompt-injection payloads and is released for defensive research. Payloads are generated for the benchmark, a portion of them seeded from public jailbreak and prompt-injection collections; none target a specific deployed system, and no payload was tested against a production service. The benchmark's purpose is to measure defenses, and the accompanying defense is released alongside it. We judge the disclosure risk to be low relative to the defensive value, since the attack classes covered are already public.

REPRODUCIBILITY STATEMENT

Warden-Bench, its generators, WardenGuard and every evaluation script are released. Labels are assigned at construction by deterministic generators rather than post hoc, so each is traceable to the code that produced it. The residual matrix is built from two independent measurements joined per case and requires no new API calls to reconstruct.

REFERENCES

[1] T. Schick *et al.*, "Toolformer: language models can teach themselves to use tools," in *Proc. NeurIPS*, 2023.

[2] S. Yao *et al.*, "ReAct: synergizing reasoning and acting in language models," in *Proc. ICLR*, 2023.

[3] X. Liu *et al.*, "AgentBench: evaluating LLMs as agents," in *Proc. ICLR*, 2024.

[4] F. Perez and I. Ribeiro, "Ignore previous prompt: attack techniques for language models," in *NeurIPS ML Safety Workshop*, 2022.

[5] OWASP Foundation, "OWASP Top 10 for Large Language Model Applications," version 2025 (released Nov. 17, 2024), genai.owasp.org/llm-top-10.

[6] Q. Zhan *et al.*, "InjecAgent: benchmarking indirect prompt injections in tool-integrated LLM agents," in *Findings of ACL*, 2024.

[7] J. Yi *et al.*, "Benchmarking and defending against indirect prompt injection attacks on large language models," *arXiv:2312.14197*, 2023.

[8] W. Zou, R. Geng, B. Wang, and J. Jia, "PoisonedRAG: knowledge corruption attacks to retrieval-augmented generation of large language models," in *Proc. USENIX Security*, 2025; *arXiv:2402.07867*.

[9] E. Debenedetti, J. Zhang, M. Balunović, L. Beurer-Kellner, M. Fischer, and F. Tramèr, "AgentDojo: a dynamic environment to evaluate prompt injection attacks and defenses for LLM agents," in *NeurIPS Datasets and Benchmarks*, 2024; *arXiv:2406.13352*.

[10] F. Jiang, Z. Xu, L. Niu, Z. Xiang, B. Ramasubramanian, B. Li, and R. Poovendran, "ArtPrompt: ASCII art-based jailbreak attacks against aligned LLMs," in *Proc. ACL Long Papers*, vol. 1, 2024, pp. 15157–15173.

[11] A. Wei, N. Haghtalab, and J. Steinhardt, "Jailbroken: how does LLM safety training fail?," in *Proc. NeurIPS*, 2023.

[12] ProtectAI, "deberta-v3-base-prompt-injection-v2," HuggingFace model card, 2024.

[13] Y. Liu, Y. Jia, J. Jia, D. Song, and N. Z. Gong, "DataSentinel: A Game-Theoretic Detection of Prompt Injection Attacks," in *Proc. 46th IEEE Symposium on Security and Privacy (S&P)*, 2025, pp. 2190–2208; *arXiv:2504.11358*.

[14] T. Shi *et al.*, "PromptArmor: simple yet effective prompt injection defenses," *arXiv:2507.15219*, 2025.

[15] M. Wang, Y. Zhang, and G. Gu, "PromptSleuth: detecting prompt injection via semantic intent invariance," *arXiv:2508.20890*, 2025.

[16] K. Hines, G. Lopez, M. Hall, F. Zarfati, Y. Zunger, and E. Kiciman, "Defending against indirect prompt injection attacks with spotlighting," *arXiv:2403.14720*, 2024.

[17] Qwen Team, Alibaba Cloud, "Qwen3 technical report," *arXiv:2505.09388*, 2025. Checkpoint served as qwen/qwen3-30b-a3b-instruct-2507.

[18] K. Greshake *et al.*, "Not what you've signed up for: compromising real-world LLM-integrated applications with indirect prompt injection," in *Proc. ACM AISec*, 2023, pp. 79–90.

[19] A. Zou, Z. Wang, N. Carlini, M. Nasr, J. Z. Kolter, and M. Fredrikson, "Universal and transferable adversarial attacks on aligned language models," *arXiv:2307.15043*, 2023.

[20] X. Shen, Z. Chen, M. Backes, Y. Shen, and Y. Zhang, " "Do Anything Now": Characterizing and Evaluating In-The-Wild Jailbreak Prompts on Large Language Models," in *Proc. ACM CCS*, 2024, pp. 1671–1685, doi: 10.1145/3658644.3670388; *arXiv:2308.03825*.

[21] deepset, "Prompt injections dataset," Hugging Face Datasets, 2023. [Online]. Available: https://huggingface.co/datasets/deepset/prompt-injections

[22] TakSec, "Prompt injection everywhere: a collection of prompt-injection payloads," GitHub, 2024. [Online]. Available: https://github.com/TakSec/Prompt-Injection-Everywhere

[23] Cranot, "chatbot-injections-exploits," GitHub, 2023. [Online]. Available: https://github.com/Cranot/chatbot-injections-exploits

[24] yunwei37, "Prompt-injection payload collection," GitHub, 2023.

[25] OpenAI, "gpt-oss-120b & gpt-oss-20b model card," *arXiv:2508.10925*, Aug. 2025.

[26] NVIDIA, "Nemotron-3-Super-120B-A12B," served via OpenRouter as nvidia/nemotron-3-super-120b-a12b, 2025.

[27] Gemma Team, Google DeepMind, "Gemma 4 model card: gemma-4-31b-it," Google DeepMind, 2026. [Online]. Available: https://ai.google.dev/gemma

[28] A. Grattafiori *et al.* (Llama Team @ Meta), "The Llama 3 herd of models," *arXiv:2407.21783*, 2024.

[29] Mistral AI, "Mistral Small 3.1 model card," Tech. Rep., 2025.

[30] C. Anil, E. Durmus, N. Panickssery, M. Sharma, J. Benton, S. Kundu, J. Batson, M. Tong, J. Mu, D. Ford *et al.*, "Many-shot Jailbreaking," in *Advances in Neural Information Processing Systems (NeurIPS) 37*, 2024, OpenReview cw5mgd71jW; also Anthropic Tech. Rep., Apr. 2024.

[31] M. Russinovich, A. Salem, and R. Eldan, "Great, now write an article about that: the Crescendo multi-turn LLM jailbreak attack," in *Proc. USENIX Security*, 2025; *arXiv:2404.01833*.

[32] M. Nasr *et al.*, "The attacker moves second: stronger adaptive attacks bypass defenses against LLM jailbreaks and prompt injections," *arXiv:2510.09023*, 2025.

[33] S. Zhang, J. Li, Q. Cao, R. Zhang, and P. Xie, "Send a SCOUT first: pre-hoc reasoning for adaptive detector allocation in prompt-injection defense," *arXiv:2605.30837*, 2026.

[34] M. Phute *et al.*, "LLM self defense: by self examination, LLMs know they are being tricked," *arXiv:2308.07308*, 2023.

[35] Deepset, "`deberta-v3-base-injection`," HuggingFace model card, 2023.

[36] Y. Liu *et al.*, "Formalizing and benchmarking prompt injection attacks and defenses," in *Proc. USENIX Security*, 2024.

[37] S. Chen, J. Piet, C. Sitawarin, and D. Wagner, "StruQ: defending against prompt injection with structured queries," in *Proc. USENIX Security*, 2025; *arXiv:2402.06363*.

[38] S. Chen, A. Zharmagambetov, S. Mahloujifar, K. Chaudhuri, D. Wagner, and C. Guo, "SecAlign: defending against prompt injection with preference optimization," in *Proc. ACM CCS*, 2025, doi: 10.1145/3719027.3744836; *arXiv:2410.05451*.

[39] Y. Yuan *et al.*, "GPT-4 is too smart to be safe: stealthy chat with LLMs via cipher," in *Proc. ICLR*, 2024; *arXiv:2308.06463*.

[40] Y. Bai *et al.*, "Constitutional AI: harmlessness from AI feedback," *arXiv:2212.08073*, 2022.

[41] Y. Gao *et al.*, "Retrieval-augmented generation for large language models: a survey," *arXiv:2312.10997*, 2023.

[42] J. Ji *et al.*, "AI alignment: a comprehensive survey," *arXiv:2310.19852*, 2023.

[43] D. Kang *et al.*, "Exploiting programmatic behavior of LLMs: dual-use through standard security attacks," *arXiv:2302.05733*, 2023.

[44] P. Lewis *et al.*, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Proc. NeurIPS*, 2020, doi: 10.5555/3495724.3496517; *arXiv:2005.11401*.

[45] L. Ouyang *et al.*, "Training language models to follow instructions with human feedback," in *Proc. NeurIPS*, 2022.

[46] Y. Qin *et al.*, "ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs," in *Proc. ICLR (Spotlight)*, 2024, OpenReview `dHng2O0Jjr`; *arXiv:2307.16789*.

[47] S. Schulhoff *et al.*, "Ignore this title and HackAPrompt: exposing systemic vulnerabilities of LLMs through a global-scale prompt hacking competition," in *Proc. EMNLP*, 2023, pp. 4945–4977 (Best Theme Paper); *arXiv:2311.16119*.

[48] J. Xie *et al.*, "TravelPlanner: a benchmark for real-world planning with language agents," in *Proc. ICML*, 2024.

[49] S. Zhou *et al.*, "WebArena: a realistic web environment for building autonomous agents," in *Proc. ICLR*, 2024.

[50] S. Willison, "Prompt injection attacks against GPT-3," Sept. 2022. [Online]. Available: `https://simonwillison.net/2022/Sep/12/prompt-injection/`

[51] S. Toyer, O. Watkins, E. A. Mendes, J. Svegliato, L. Bailey, T. Wang, I. Ong, K. Elmaaroufi, P. Abbeel, T. Darrell, A. Ritter, and S. Russell, "Tensor Trust: interpretable prompt injection attacks from an online game," in *Proc. ICLR*, 2024; *arXiv:2311.01011*.

[52] E. Zverev, S. Abdelnabi, S. Tabesh, M. Fritz, and C. H. Lampert, "Can LLMs separate instructions from data? And what do we even mean by that?," in *Proc. ICLR*, 2025; *arXiv:2403.06833*.

[53] J. Piet, M. Alrashed, C. Sitawarin, S. Chen, Z. Wei, E. Sun, B. Alomair, and D. Wagner, "Jatmo: prompt injection defense by task-specific finetuning," in *Proc. ESORICS*, 2024; *arXiv:2312.17673*.

[54] S. Abdelnabi, A. Fay, G. Cherubin, A. Salem, M. Fritz, and A. Paverd, "Get my drift? Catching LLM task drift with activation deltas," in *Proc. IEEE SaTML*, 2025; *arXiv:2406.00799*.

[55] F. Wu, E. Cecchetti, and C. Xiao, "System-level defense against indirect prompt injection attacks: an information flow control perspective," *arXiv:2409.19091*, 2024.

[56] E. Debenedetti, I. Shumailov, T. Fan, J. Hayes, N. Carlini, D. Fabian, C. Kern, C. Shi, A. Terzis, and F. Tramèr, "Defeating prompt injections by design," *arXiv:2503.18813*, 2025.

[57] E. Wallace, S. Feng, N. Kandpal, M. Gardner, and S. Singh, "Universal adversarial triggers for attacking and analyzing NLP," in *Proc. EMNLP-IJCNLP*, 2019; *arXiv:1908.07125*.

A Adaptive Attacks

The results so far hold the attack fixed. An attacker who observes the defense and rewrites until it passes is the regime in which most published prompt-injection defenses fail [32]; consequently, we evaluate WardenGuard and the two strongest LLM-judge baselines under it.

A stratified sample of 300 held-out attacks is drawn, 25 from each of the twelve tiers, restricted to rows whose payload appears verbatim in a rewritable channel. Each attacker sees the defense's verdict after every query and may revise the payload, up to a budget of fifteen queries per case; a case counts as a success at budget b if any of its first b queries is allowed. Two attackers are run against each defense: the first, *random search*, composes the mutation engine of Sec. 3.3 with the fourteen encoding families and uses no model, while the second, *LLM rewrite*, gives GPT-4.1-mini the payload, the verdict, and the running history and asks for a revision. All three defenses see the same 300 cases and the same budget, with WardenGuard running in its full configuration, cascade and gate together.

Table A1 reports attack success at each budget over all 1,800 (defense, attacker, case) runs, none of which returned an invalid verdict. WardenGuard admits fewer successes than either baseline at every budget under both attackers, and the gap is widest where it matters operationally; at the first query, an attacker pays nothing for reconnaissance. Taking each case's worst outcome across the two attackers, the full-budget figures are 34.0% (95% CI 28.9 - 39.5) for WardenGuard against 68.7% (63.2 - 73.7) for PromptSleuth and 70.0% (64.6 - 74.9) for PromptArmor. Since all three defenses are evaluated on the same cases, the paired counts are tested directly: across the six comparisons, McNemar's exact test rejects equality with Holm-adjusted $p < 1.5 \times 10^{-8}$ throughout. Furthermore, the counts are lopsided rather than merely significant; under random search, 121 cases that PromptArmor allows are blocked by WardenGuard against 5 in the reverse direction.

A.1 Isolating the Attestation Gate

The attestation gate refuses 70 of the 300 cases outright, and a refusal is immune to rewriting by construction. Since those cases cannot be counted as detection, what remains without them is examined next. The refusals fall where the threat model predicts: every T3 (tool-output directive) and T8 (tool-permission escalation) case, 15 of 25 T2 cases and 5 of 25 T11 cases, and no case in the eight tiers that carry no principal claim. On the same 70 cases, the baselines, which have no notion of provenance, are breached at rates from 20.0% to 54.3%.

Restricted to the 230 cases the gate admits, where WardenGuard's advantage can come only from the cascade, random-search success is 37.0% against 70.4% for PromptSleuth and 73.9% for PromptArmor, while LLM-rewrite success is 28.7% against 41.3% and 43.9%. All four paired comparisons remain significant at Holm-adjusted $p < 2.1 \times 10^{-4}$. The separation is therefore not an artifact of the gate; attestation and detection each carry part of it, which is the behavior the layered design predicts and the reason the two are reported apart.

A.2 Cost Under Attack

The budget also prices the defense. Averaged over every run, WardenGuard spends 58.1 cost units per case against 866.3 for PromptSleuth and 868.8 for PromptArmor, since Stage 1 settles most queries in an adaptive campaign locally and only escalations reach the frontier model; this represents

Table A1: Adaptive attack success rate at query budget b.

Defense	b=1	b=2	b=5	b=10	b=15
Random search					
WardenGuard	**9.7%**	**15.0%**	**22.3%**	**27.7%**	**28.3%**
PromptSleuth	23.7%	35.3%	55.0%	64.7%	66.7%
PromptArmor	21.0%	42.0%	58.0%	65.0%	67.0%
LLM rewrite					
WardenGuard	**9.3%**	**12.7%**	**17.7%**	**19.3%**	**22.0%**
PromptSleuth	22.7%	28.3%	34.0%	38.0%	40.0%
PromptArmor	21.3%	26.0%	32.0%	34.3%	38.3%

a fifteen-fold reduction sustained while an attacker is actively probing, which is the condition under which a per-query frontier call is most expensive. The attacker's own costs run the other way - producing its successes against WardenGuard took the LLM rewriter 2,641 calls, against 5,561 for PromptSleuth and 8,590 for PromptArmor.

One result runs against expectation and is reported as measured: LLM rewriting is the weaker of the two attackers against all three defenses, not merely against WardenGuard. Feedback-guided revision converges on a narrow family of paraphrases, whereas random search keeps sampling the encoding families independently of the verdict; the rewriter's lower call counts above reflect earlier convergence rather than greater efficiency. Random search is therefore treated as the stronger adversary here, and the worst case over both is reported; coverage of optimization-based attackers is not claimed, and this remains open (Sec. 7).

B WardenGuard Holistic Prompt

```
You are a prompt injection detector for an AI agent system.
Analyze the COMPLETE interaction below across all data
channels.
The agent's intended task is: {system_prompt}
Each channel has a trust level indicating its data source:
- UNTRUSTED channels contain external/user-controlled data
- SEMI-TRUSTED channels contain system-generated but
externally
 influenced data
- TRUSTED channels contain internal system state
=== CHANNEL DATA ===
[USER INPUT] (UNTRUSTED):
{user_input}
[TOOL OUTPUT] (SEMI-TRUSTED):
{tool_output}
[AGENT MEMORY] (TRUSTED):
{agent_memory_state}
[CONVERSATION HISTORY] (SEMI-TRUSTED):
{conversation_history}
[RETRIEVED CONTEXT] (UNTRUSTED):
{retrieved_context}
[FILE METADATA] (UNTRUSTED):
{file_metadata}
=== TASK ===
Determine if ANY channel contains a prompt injection
attempt.  Consider the FULL CONTEXT: security discussions,
code examples, and technical documentation about attacks are
LEGITIMATE content when they align with the agent's intended
task.  Only flag content that genuinely attempts to override
the agent's behavior.
Respond ONLY with JSON:
{"verdict":  "injection" or "benign", "confidence":  0.0 to
1.0, "attack_channel":  "channel_name" or "none"}
```

Table A2: Classification of all 697 T9 misses. Percentages are rounded independently.

Category	n	%
Content benign after mutation	323	46.3
Benign payload in red-team scaffold	222	31.9
Genuine detector miss	84	12.1
Non-ASCII / zero-width content	41	5.9
Distractor dilution	27	3.9
Total	**697**	**100.0**

C T9 Miss Audit

WardenGuard's recall on T9 (cross-turn fragmentation) is 69.9%, the lowest of the twelve injection tiers. A per-case audit of the T9 misses (Appendix C) attributes the residual primarily to a definitional mismatch rather than to a failure to associate content across turns. T9 payloads pass through the twenty-transform mutation engine of Sec. 3. In the fragment-split and sleeper sub-types, the semantics-softening transforms (INDIRECT_PHRASING, REGISTER_SHIFT, NARRATIVE_EMBEDDING, SENTENCE_RESTRUCTURE) rewrite an override-shaped payload into a request whose concatenated content carries no behavior-overriding instruction; additionally, the progressive-build template wraps a benign payload in a red-team scaffold. Warden-Bench labels both as injections under the deviation criterion of Sec. 1 - any instruction that diverts the agent from τ_U - whereas the holistic prompt (Appendix B) directs the judge to flag only content that "genuinely attempts to override the agent's behavior." Of the 697 misses, 545 (78.2%) fall in this gap; 84 (12.1%) are genuine detector misses, concentrated in the callback-reference sub-type where an explicit directive-disclosure marker is present; 41 (5.9%) carry non-ASCII or zero-width content, whose recoverability under normalization was not tested; and 27 (3.9%) are distractor dilution.

Every T9 row on which WardenGuard predicted *benign* was classified by concatenating all user-role turns with the current `user_input`, attempting base64 and hexadecimal decoding, and testing the result against a marker list of override, disclosure, role-hijack, and encoding-execute phrases. Cases whose only marker is the phrase "Execute the following" *and* which carry the progressive-build scaffold prefix are reported separately, since that phrase belongs to the benign template wrapper rather than to the payload. Table A2 reports the outcome. Additionally, 49 of the 84 genuine misses fall in the callback-reference sub-type, accounting for every miss in that sub-type, where a directive-disclosure marker is present in the concatenated text. The content-benign category is led by the semantics-softening transforms of App. G.12.2 (SYNONYM_SWAP 32, INDIRECT_PHRASING 30, SENTENCE_RESTRUCTURE 24, VOICE_FLIP 24, REGISTER_SHIFT 23), indicating that the residual is concentrated where the mutation engine softens the payload's semantics rather than where it obfuscates the payload's surface: 206 of these rows carry a semantics-softening transform against 50 carrying surface obfuscation. The classifier is deterministic and its rules are released with the artifact, together with a stratified 125-case sample supporting independent replication of the categorization.

D Why the Cascade Replaced the Single Judge

The defense evaluated in this paper is the second design. The first sent every case to one frontier judge: the same six trust-labeled channels and the same deterministic decoder of Sec. 4, read by GPT-4.1-mini in a single call, with no local stage and no confidence rule. The appendix reports what that design measured and why it was replaced. Furthermore, both arms are scored on the same 23,601 Warden-Bench rows (17,702 attacks and 5,899 benigns) with the same decoder, prompt, and parser, so the comparison isolates the routing.

Table A3 reports both arms. The cascade is better on every detection metric and costs an order of magnitude less to run. It recovers 739 attacks the single judge missed while removing 55 of its false positives, so the gain is not a threshold trade in which recall is bought with benign traffic. Over the paired rows, it is correct on 1,120 cases the single judge got wrong against 326 in the reverse direction, which McNemar's exact test separates at $p < 10^{-100}$. The result may seem counter-

Table A3: Single frontier judge against the cascade on Warden-Bench.

	Single judge	Cascade	Δ
F1	0.9298	**0.9543**	+0.0245
MCC	0.7695	**0.8381**	+0.0686
Precision	0.9808	**0.9849**	+0.0041
Recall	88.4%	**92.5%**	+4.1 pp
FPR	5.19%	**4.25%**	−0.94 pp
Frontier calls	23,601	**2,152**	11.0× fewer
Mean cost per case	100.0	**9.96**	10.0× lower

intuitive: the cascade settles 75.5% of cases with a 30B open-weight model that the single-judge design never consulted, and the frontier judge sees only the 9.1% that escalate. The mechanism is the confidence rule rather than the second model. A single judge answers every case at the same effort whether or not the case is ambiguous, and its errors concentrate where a fixed prompt is asked to be decisive about inputs it finds marginal. Routing those cases, and only those, to the frontier judge lets the expensive model spend its budget where the local stage has no opinion, while the cases the local stage settles are the ones on which the two models agree anyway. The gate and the memory monitor contribute independently of the judge: both run ahead of detection in the current design, and the 3,630 cases the gate refuses are resolved at no judge cost at all, which the single-judge design had no way to express.

The cost figures use the paper's accounting throughout, one unit per local call and one hundred per frontier call. In absolute terms, a full Warden-Bench pass costs roughly $4.96 in frontier calls under the single judge against $0.45 under the cascade, and the local stage's median latency of 996 ms removes 91% of the network round-trips a deployment would otherwise make. The comparison is run on Warden-Bench, the benchmark against which both designs were developed, so it records the decision as it was actually made rather than a claim of uniform superiority; the cross-benchmark evidence for WardenGuard is Sec. 5.3. Furthermore, the single-judge arm is the design as it stood when it was replaced, not a version tuned afterwards with what the cascade later established about the band and the standing check.

E Extended Related Work

Detector defenses. ProtectAI [12] and Deepset's HuggingFace classifier [35] fine-tune encoder models on labeled prompt-injection corpora, while DataSentinel [13] fine-tunes Mistral-7B with LoRA on OPI [36]-style attacks. Task-drift detection [54] reads the target model's own activations before and after the untrusted span and flags a shift in the task being pursued, which requires white-box access to the model under protection. StruQ [37] and SecAlign [38] train the agent itself to ignore embedded instructions, a complementary direction to a runtime detector, whereas Jatmo [53] takes the training-side route without alignment data, finetuning a non-instruction-tuned base model on a single task so that embedded instructions have no instruction-following behavior to exploit.

LLM-judge defenses. PromptArmor [14] uses per-field structured prompts, whereas Prompt-Sleuth [15] asks a frontier judge to reason over task-level intent (semantic intent invariance) across a concatenated system-plus-user prompt. Spotlighting [16] is the closest prior work on provenance: it delimits, datamarks, or base64-encodes the untrusted span so the task model is less likely to obey instructions inside it, reporting attack success below 2% in its own setting. On the agentic threat model of Sec. 1, however, Spotlighting carries one bit, a span is untrusted or it is not, so it cannot express that a payload arrived through the memory store rather than the retrieval index, which is the distinction tiers T1-T3 are built to separate. It also operates by rewriting the protected model's own prompt, so it must be deployed inside every application's inference path, and its efficacy is a property of that model's instruction-following; the residual ASR matrix of Sec. 5.4 measures WardenGuard against five different target models without changing any of them. StruQ [37] and SecAlign [38] enforce a structured-query separation between instruction and data, and share this second property. Correspondingly, none of these methods annotates channel provenance with graded trust labels, and none emits the implicated channel as a structured output.

1. **System-level defenses.** A second line places the separation outside the model. CaMeL [56] extracts a control flow from the trusted user query and executes it in a restricted interpreter, so data returned by tools can influence values but not the program. Similarly, Wu et al. [55] give the corresponding information-flow-control account, propagating labels through an agent's execution and enforcing them where an action is taken. Both, however, buy their guarantee by rewriting the application: the agent's plan must be expressible in the restricted form, and a policy must be written per deployment. WardenGuard sits earlier in the pipeline and is orthogonal to both; it labels and screens the input a conventional agent is about to read, leaves the agent's execution untouched, and can be deployed in front of a system that also enforces flow control. Furthermore, Zverev et al. [52] measure the premise these lines rest on and find that instruction-tuned models do not reliably separate instructions from data on their own.
2. **Detector allocation.** SCOUT [33] treats defense as per-input detector allocation: a retrieval step recovers how each light detector behaved on structurally similar inputs, a small predictor estimates per-detector reliability and latency, and an LLM judge is consulted only when the light detectors' vote is uncertain. It shares the premise that the frontier judge is the accurate but expensive component; correspondingly, WardenGuard's cascade is such an allocation with one fixed rule - a local open-weight judge on the trust-labeled multi-channel prompt settles the clear cases, and the frontier judge is consulted only for the rest, at most one frontier call per input.
3. **Adversarial robustness.** Adversarial suffix attacks [19], which extend universal triggers for NLP models [57] to aligned chat models, produce universal strings that flip aligned model behavior; jailbreaking surveys [11, 20] catalog attack styles. Additionally, encoding-based bypasses are described in [10, 43, 39].
4. **Agentic-IPI benchmarks.** BIPIA [7] and InjecAgent [6] target indirect injections in tool responses, while AgentDojo [9] extends to interactive multi-step environments and PoisonedRAG [8] targets retrieval pipelines. Tensor Trust [51] collects attacks and defenses written by players of an online game, giving a corpus of injections optimized by people rather than by a generator. Warden-Bench (this work) consolidates twelve attack tiers including encoding evasion across a 23,601-row split (75% attacks / 25% benign). Wider agent benchmarks [3, 48, 49] test capabilities orthogonal to security.
5. **Foundations.** The attack class was named in [50], which drew the analogy to SQL injection and proposed separating instruction from data at the interface; demonstrations against deployed systems followed [4, 18]. HackAPrompt [47] is a global-scale red-team. Furthermore, tool learning [1, 46] and ReAct-style planning [2] provide the agent substrate, while retrieval-augmented generation [44, 41] supplies the `retrieved_context` channel. Constitutional AI [40] and RLHF [45] shape the agent's resistance to override; alignment surveys [42] place prompt-injection in the broader landscape of misuse.

Multi-channel input [18], holistic judging [34], and explicit trust labels [16] each appear in earlier work; however, no prior defense combines multi-channel trust-labeled input with a deterministic decoder, attestation before detection, write-time memory gating, and a local-first judge cascade. Furthermore, the standing check of Sec. 4.5 is not itself a new mechanism, and we do not claim it as one: nearest-neighbour distance in an embedding space is a standard out-of-distribution score, and selective prediction with a reject option is long established. What is reported instead is the measurement that motivates applying it to one side of a cascade only - that a local detector's confident clears invert out of distribution while its confident blocks transfer - and the consequence that conditioning only the clear recovers most of the zero-shot accuracy a cost-reducing cascade otherwise gives up.

Their joint use as a defense prompt format is, to the authors' knowledge, not previously published, and the composition is what delivers the measured result: removing either of the two ablated ingredients, the trust labels or the decoder, costs measurable performance, while removing both costs more than removing either alone (F1 0.9371, against 0.9475 without the decoder and 0.9486 without the annotation, from 0.9543 for the complete defender), so the two contribute separately rather than substituting for one another. Correspondingly, the accompanying empirical contribution, the measured chained residual ASR matrix - 177,020 executed measurements yielding a 442,550-cell join - is among the largest reported in the public prompt-injection *defense-evaluation* literature as of submission. It exceeds DataSentinel's 35,700-sample contaminated-target sweep [13] by $\approx 5\times$

Algorithm 1 WARDENGUARD-DETECT

Require: Channels $\mathcal{C} = \{c_{\text{UI}}, c_{\text{TOOL}}, c_{\text{MEM}}, c_{\text{HIST}}, c_{\text{RAG}}, c_{\text{FILE}}\}$
Require: System prompt s, judge LLM $\mathcal{M}$, trust map T
Ensure: Verdict $v \in \{0, 1\}$, confidence $\rho \in [0, 1]$, channel a
1: $\mathcal{C}' \leftarrow \emptyset$
2: **for all** $(k, c) \in \mathcal{C}$ with $|c| > 0$ **do**
3: $c \leftarrow$ STRIPZEROWIDTH(c)
4: **for** decoder $d \in$ [HEX, BASE85, BASE32, BASE64, UNICODE, ROT13, ROT47, ATBASH, LEET, REVERSED, MIXEDCASE, CAESAR] **do**
5: $\hat{c} \leftarrow d(c)$
6: **if** $\hat{c} \neq \perp \wedge \hat{c} \neq c \wedge$ GATE$_d(\hat{c}, c)$ **then**
7: $c \leftarrow \hat{c}$
8: **end if**
9: **end for**
10: $\mathcal{C}'[k] \leftarrow c$
11: **end for**
12: $p \leftarrow$ BUILDPROMPT$(s, \mathcal{C}', T)$
13: $J \leftarrow \mathcal{M}(p;\ \tau{=}0,\ \text{max}{=}100)$
14: **return** $(J.\text{verdict}, J.\text{confidence}, J.\text{attack_channel})$

The decoder list on line 4 contains 12 entries; together with STRIPZEROWIDTH (line 3) and the natively-handled URL-percent (§4.2, not decoded), the architecture covers all 14 encoding families of T11. Pipeline order places BASE85 and BASE32 before BASE64 to avoid the alphabet-superset collision (`base32` alphabet $\subset$ `base64` alphabet), and HEX first, since `base64` also accepts a hex string and decodes it to Unicode garbage that the word-count gate can pass. The $+2$ word-gain constant inside GATE$_d$ for the six word-gain-gated decoders (line 6) prevents single-word coincidences from triggering a decoding swap. It is a fixed design parameter, set to $+2$ for zero false decodes on the full T11 benign set. Those 465 benigns, however, sit inside the reported false-positive denominator, so this constant is tuned on test; the frozen split postdates these runs (Sec. 5.1.1).

on executed measurements and PromptArmor's largest implicit sweep on AgentDojo ($\approx$ 28,000 raw runs from 11 defense configurations $\times$ 4 attacks $\times$ 629 scenarios $\times$ 1 backend [14, 9]) by $\approx 6\times$. Adversarial-collection corpora (HackAPrompt [47], etc.) are larger but are attack *datasets*, not defense-evaluation grids, and are therefore outside the comparison scope.

F DETECTION ALGORITHM

Algorithm 1 gives the detection procedure in full. Specifically, each detector is a small heuristic with a self-validation gate GATE$_d$, which differs by decoder family. The six word-gain-gated decoders (ROT13, ROT47, atbash, reversed, Caesar, leet) require the decoding to add at least two common English words; all but leet additionally require at least two in total, while leet requires at least three leetspeak characters. The five alphabet-validated decoders (base64, base32, base85, hex, Unicode-escape) require an alphabet match and a printable result, whereas mixed-case gates on an alternating-case ratio alone. Every gate, furthermore, requires the candidate to differ from the original string. The successful decodings are inlined into the channel as `[DECODED_BASE64: plaintext]`. The alphabet-validated detectors splice the marker in place, preserving surrounding tokens, except that base64 returns the bare plaintext when the whole channel is a single encoded blob; the word-gain-gated decoders replace the channel with the marker, and mixed-case normalization rewrites it without one.

G DETAILED RESULTS AND MECHANISM NOTES

This subsection holds the per-tier, per-slice and per-cell breakdowns behind the summary results of Section 5.

G.0.1 SIX-CHANNEL TRUST TAXONOMY

The trust label is a textual tag inside the prompt. Each channel is introduced as `[CHANNEL_NAME] (TRUST_LEVEL):` followed by the content. Alongside the deterministic decoder stage (Sec. 4.2),

this is what separates WardenGuard from a flat-concatenation LLM judge. WardenGuard's combined benign FPR is 4.25% against PromptSleuth's 48.60% (Table A11); the two systems differ in channel segmentation, trust annotation, decoder stage, and output schema, so the gap is not attributed here to any single component. The mechanism the tag is intended to supply is that an imperative phrase under TRUSTED can be recognized as consistent with the agent's own task, whereas the same phrase under UNTRUSTED is suspicious.

G.0.2 Encoding Decoder: Coverage Detail

The decoder applies twelve active detectors in order (Algorithm 1, line 4) plus a zero-width character strip (line 3). The twelve-detector pipeline was run on all 1,865 T11 rows; the four extension decoders (base85, base32, atbash, ROT47) are exercised by the 532 T11-v3 rows. Each detector is a small heuristic with a self-validation gate that differs by decoder family and requires the candidate to differ from the original string; Appendix F gives the gate conditions and the marker-splicing rules. The decoder is deterministic, runs in approximately two milliseconds per row, and is model independent. URL-percent is left unmodified and read by the judge natively; at 74% recall, however, it is among the weakest families (Sec. 5.5), so a URL-percent decoder is a candidate addition. Together, the twelve active decoders, the zero-width strip, and the LLM-native URL handling cover the fourteen encoding families of T11.

G.0.3 Overall Detection: Cascade Walkthrough

Table 2 reports the primary detection results for the complete defense. WardenGuard runs as a deterministic two-stage cascade. Stage 1, Qwen3-30B-A3B-Instruct-2507 in 4-bit quantisation served locally by Ollama, reads the six annotated channels through the same decoder and prompt as the judge and settles a case when the probability it assigns to the injection verdict is confidently low or confidently high; the remaining cases escalate to GPT-4.1-mini, whose verdict is final. Before either stage runs, an attestation gate refuses any case whose principal claims a capability it has not attested, and a memory-integrity monitor scores every write the agent would make to its persistent memory, refusing writes that carry injected instructions or drift from the task goal; every decision is appended to a hash-chained audit log. Of the 23,601 cases, the gate refuses 3,630 (15.4%) at no judge cost, Stage 1 settles 17,819 (75.5%), and 2,152 (9.1%) reach GPT-4.1-mini. The locally served Stage 1 therefore makes the defense 11 times cheaper in frontier-judge calls than consulting the judge on every case, at a mean cost of 10.0 units per case on a scale where a Stage 1 call costs 1 and a judge call 100.

Two caveats attach to the gate and the monitor. Warden-Bench attests capabilities from its own claim tables, so the gate's refusals measure the mechanism against simulated attestation rather than a deployed attestation service. The monitor, meanwhile, treats every candidate write as untrusted, since the benchmark records what the agent would write but not the provenance of that content; its refusal rate is therefore that of the strictest provenance setting.

G.0.4 Cross-Benchmark Generalization Detail

Frontier-judge volume rises because a local clear requires standing (Sec. 4.5) and external traffic rarely resembles the development split, so the share of cases reaching the frontier judge climbs from 9.1% on Warden to a range of 44.7% to 67.7% across the four external benchmarks. Frontier-judge volume therefore tracks the distance from the calibration distribution rather than a fixed budget. Withholding standing on unfamiliar inputs costs 1.5 pp of frontier-judge calls in distribution and raises mean external F1 by 6.5 pp, from 0.845 to 0.911. The two quantities are not independently tunable; relaxing the standing requirement returns the judge rate toward its in-distribution value and gives back the external accuracy with it. Sweeping the distance at which standing lapses confirms the exchange is not free in either direction: at the loosest setting, where the local stage clears almost everything, the judge rate returns to 7.8% in distribution at a cost of 1.8 pp of mean external F1. Two of the four external benchmarks, notably, do not move at all under the sweep, their cases sitting beyond even the loosest setting of the requirement.

The fourth benchmark, BIPIA, reaches F1=0.80, and the shortfall is definitional rather than a detection failure (Sec. 6). BIPIA labels any instruction embedded in retrieved content as an attack, irrespective of what the instruction asks for, whereas WardenGuard blocks instructions that subvert

the principal's task. Splitting BIPIA's attacks by category makes the gap visible: recall is 0.797 over the 900 attacks whose injected payload is itself harmful (keylogging, denial of service, operating system corruption) and 0.580 over the 1,350 whose payload is an innocuous request, falling to 0.07 for instructions that merely ask the agent to translate the retrieved text and 0.13 for ones that ask it to re-encode it. Fig. A3 shows the F1 bars.

G.0.5 CELL-LEVEL RESIDUAL ASR DETAIL

The margin is not bought with indiscriminate blocking, which is what the residual metric alone cannot settle. Residual ASR falls monotonically as a defense blocks more traffic, so a detector can drive it arbitrarily low by flagging everything: the nearest competitor on this axis, PromptSleuth at 3.34%, gets there while flagging 48.60% of benign traffic, 11.4× WardenGuard's 4.25%, and its MCC of 0.4051 against WardenGuard's 0.8381 shows how much of that is prevalence rather than discrimination. DataSentinel's 3.90% residual looks competitive when read on one axis alone; however, that impression holds only until its 72.69% FPR is read alongside it, which is precisely the point. WardenGuard is now lower than every competitor on every target LLM (Table 4) *and* operates at a false-positive rate a production deployment can absorb; it is the only defense that does both. Additionally, every defense is scored per cell on the same 17,702 attacks against the same target LLM, so the twenty pairwise comparisons are paired; these are tested with McNemar's exact test, Holm-corrected, with BCa bootstrap intervals (Table 4).

WardenGuard is significantly lower in all twenty cells: every comparison favors it, none is tied, and none favors a competitor. The smallest margin is 1.04 pp, against DataSentinel on GPT-OSS-120B, and the largest is 27.14 pp, against ProtectAI on Mistral-Small-3.1. Against PromptArmor, the only competitor that also operates below 5% FPR, the smallest margin is 3.56 pp. The result does not depend on tolerating a high false-positive rate elsewhere in the comparison: the two defenses closest to WardenGuard on residual ASR, PromptSleuth and DataSentinel, flag 48.60% and 72.69% of benign traffic respectively. Fig. 1 renders the per-defense mean as a bar chart. WardenGuard also has the lowest variance across target LLMs, ranging from 2.54% to 4.74% (Δ=2.20 pp) against 11.30% to 29.76% (Δ=18.46 pp) for ProtectAI, so a strong defense flattens the LLM-vulnerability landscape. Appendix G gives the heat-shaded matrix, the per-tier per-LLM cube, and the per-tier per-defense slice.

G.0.6 TRUST ANNOTATION ABLATION DETAIL

The annotation is worth $+0.0057$ F1 and $+0.0101$ MCC, raising recall 1.6 pp (Table 5). McNemar over the paired rows gives p=1.0×10^{-9}, with 491 cases the annotation gets right that the ablated arm gets wrong against 317 in the reverse direction. It does not, however, reduce false positives: the annotation *adds* 117 of them, 1.98 pp of FPR. The direction is mechanically unsurprising once stated, since marking a channel untrusted makes the judge readier to block what arrives through it, and the recall gain and the false-positive cost are the same effect measured on the two classes. The consequence for the comparison in Sec. 5.2 is worth being explicit about: WardenGuard's benign-FPR advantage over PromptSleuth is *not* attributable to the trust labels, which move that rate the wrong way. Which of the remaining differences (channel segmentation, holistic framing, the cascade's local stage, output schema) produces it is not isolated by any experiment reported here.

G.0.7 ENCODING DECODER ABLATION DETAIL

The decoder carries the same kind of cost as the annotation, and the earlier single-judge measurement did not show it: benchmark-wide it adds 77 false positives, 1.30 pp of FPR, and on T11 alone it adds 19 and removes 2 (p=2.2×10^{-4}). Decoding reveals payloads, and it also renders some benign encoded content in a form the judge reads as hostile. Under a single judge, every row reached the same model and decoding changed only what that model read; in a cascade, it also changes which stage settles the row, which is where the benign cost appears. Both components, therefore, buy recall with false positives, and both are justified on MCC rather than on FPR: the balanced measure already charges each of them for the false positives it adds, and rises in both cases. Reading the FPR column alone would recommend shipping neither, and would cost 19.1 pp of recall on the encoded tier.

G.0.8 TWO-STAGE JUDGE CASCADE DETAIL

The cascade is better on every detection metric ($+0.0245$ F1) than the single-frontier-judge design it replaced, while costing an order of magnitude less to run ($10.0\times$ lower mean cost per case, $11.0\times$ fewer frontier calls); Appendix D reports both arms.

G.0.9 SELECTIVE LOCAL CLEARANCE

WardenGuard therefore conditions the two sides of Stage 1's authority differently. A local *block* stands unconditionally. A local *clear* additionally requires the case to resemble the traffic the stage was calibrated on, measured as the distance from the case's channel content to a reference set of development-split cases in the same dense embedding space the memory monitor uses; when the case is too far from that reference, Stage 1's clear is withheld and the case escalates to the frontier judge instead. The reference set and the distance at which standing lapses are fixed on the development split before any external benchmark is read, so the zero-shot results in Sec. 5.3 remain zero-shot.

The asymmetry is the point of the mechanism: a cheap detector's positive evidence transfers, and its negative evidence holds only where it has been calibrated. Withholding standing costs 1.5 pp of frontier-judge calls in distribution and recovers 6.5 pp of mean external F1 (Sec. 5.3).

G.0.10 BIPIA AND DS-BENCH FAILURE MODES

WardenGuard's 60.1% recall on BIPIA is below every other benchmark. Two distinct mechanisms drive this. First, the BIPIA TableQA subtask embeds attack strings in table rows, which WardenGuard classifies as task content rather than instructions. Second, BIPIA counts "translate to Spanish", "encode response in base64", and "use emojis" as injections, while WardenGuard only flags behavior-overriding instructions. On BIPIA's classic override categories (Misinformation, Scams & Fraud, DoS, keylogging, ransomware), WardenGuard scores 64–80%, short of its 84–95% band on the other benchmarks. A single-line prompt extension targeting embedded imperatives in `tool_output` would close the definitional gap; the stricter override definition is retained for scope consistency. Additionally, recall on the DS-Bench grammar-correction task is 54.7%, below WardenGuard's DS-Bench mean (84.4%). The legitimate task "rewrite this text" and the injected task "rewrite this text differently" are semantically near-identical; the holistic judge cannot distinguish them without an auxiliary signal that does not exist in the OPI input format.

G.1 EVALUATION SETUP DETAIL

G.1.1 BENCHMARKS

Five benchmarks are evaluated, summarized in Table A5: Warden-Bench, introduced here, with twelve attack tiers (T0 through T11) plus benign rows, and four published external benchmarks used zero-shot. The WardenGuard prompt was designed against Warden and used unchanged on the others. For InjecAgent, the canonical 1,054 attack cases of Zhan *et al.* [6] (510 direct-harm, 544 data-stealing) are used and augmented with 1,054 matched-volume benign tool-response cases (synthesized by replacing the attacker instruction in each tool-response template with a benign product review string), yielding a balanced 2,108-row split. The augmentation enables a non-zero false-positive-rate denominator; the canonical InjecAgent split is attacks-only and supports recall but not FPR. PromptArmor, PromptSleuth, and DataSentinel were originally evaluated on a 1,404-row companion split (the same 1,054 attacks plus 350 benigns randomly sampled from the non-attack tool-response corpus); this was re-augmented to 1,054 matched-volume benigns for the WardenGuard measurement to balance the FPR denominator. Restricted to its own 1,054 attacks plus a volume-matched 350-benign subsample (the first 350 by identifier), the complete defender scores $\mathrm{F1} = 0.9500$ at the same 1.71% false-positive rate it reaches on the full 2,108-row split, against $\mathrm{F1} = 0.9444$ on that split, so the ranking in Table A13 is unaffected by the volume difference; since the benign sets are disjoint, the baselines' companion split was not itself re-scored.

G.1.2 DEFENSES COMPARED

We compare WardenGuard against four published baselines, each reimplemented locally. ProtectAI and DataSentinel are detector models that score a single text field rather than a six-channel schema, so each is given the input it was designed for: ProtectAI scores `user_input` alone on T0–T10 and BEN, and DataSentinel scores the task instruction concatenated with three of the six channels (`user_input`, `retrieved_context`, `tool_output`) on the same tiers. Their T11 cells are the exception: because an encoded payload may occupy any channel, both are scored there on all six channels concatenated. This applies throughout Tables A8 and A11. **ProtectAI** [12] fine-tunes a DeBERTa-v3 base classifier head for prompt-injection detection. **DataSentinel** [13] fine-tunes Mistral-7B with LoRA adapters on OPI-style attacks. **PromptArmor** [14] prompts a frontier LLM with per-field structured queries. **PromptSleuth** [15] prompts a frontier LLM to reason over task-level intent (semantic intent invariance) across a concatenated system-plus-user prompt. The three LLM-judge defenses (PromptArmor, PromptSleuth, WardenGuard) all use GPT-4.1-mini as frontier backbone in the controlled comparison. For the two baselines that is the only model; for WardenGuard it is the cascade's second stage, the first being Qwen3-30B-A3B-Instruct served locally, so a like-for-like comparison holds the frontier model fixed and lets each defense spend it as its own design dictates. That frontier backbone is replaced with the open-weight model in Appendix G.16.1.

G.1.3 METRICS

Standard binary-classification metrics are used with TP, FP, FN, TN counted per row:

$$\text{Precision} = \frac{\text{TP}}{\text{TP}+\text{FP}}, \quad \text{Recall} = \frac{\text{TP}}{\text{TP}+\text{FN}}, \tag{2}$$

$$\text{F1} = \frac{2\,\text{TP}}{2\,\text{TP}+\text{FP}+\text{FN}}, \tag{3}$$

$$\text{FPR} = \frac{\text{FP}}{\text{FP}+\text{TN}}, \quad \text{FNR} = \frac{\text{FN}}{\text{FN}+\text{TP}}. \tag{4}$$

We report the Matthews correlation coefficient

$$\text{MCC} = \frac{\text{TP}\cdot\text{TN}-\text{FP}\cdot\text{FN}}{\sqrt{(\text{TP}+\text{FP})(\text{TP}+\text{FN})(\text{TN}+\text{FP})(\text{TN}+\text{FN})}} \tag{5}$$

alongside F1 throughout. Warden-Bench is 75% attacks, and on a split of that composition F1 rewards a classifier that answers "attack" almost always: a detector that flags every row scores F1=0.857 while contributing nothing. It may be noted that MCC uses all four cells of the confusion matrix and is zero for such a detector, so it separates discrimination from prevalence. Balanced accuracy, $\frac{1}{2}(\text{TPR}+\text{TNR})$, is reported for the same reason.

For the chained-residual analysis (Section 5.4), the following is also measured for each attack t, defense d, and target LLM L:

$$\text{residual}(t, L, d) = \big(1 - \text{pred}_d(t)\big) \cdot \text{asr}_L(t), \tag{6}$$

the indicator that the attack passed the defense and succeeded against the target LLM. Cell-level residual ASR is the mean of equation 6 over the Warden attack set. Furthermore, the success term $\text{asr}_L(t)$ is decided by a deterministic lexical rule over the target model's response, not by a second LLM judge. A response scores asr=0 if it matches a refusal lexicon or completes the row's stated NLP task, and asr=1 if it reproduces the injected directive, matches an affirmative-bypass pattern, or abandons the task in the direction of the payload. The rule is deliberately reproducible: it returns the same verdict for the same response on any machine, so the matrix carries no judging variance of its own, and every cell can be recomputed from the released response text. It is also independent of the defenses by construction, sharing no model, prompt, or inductive bias with any defense under evaluation, so a defense cannot be credited for agreeing with the judge that scores it. The lexicon is English-language; on the 949 attack rows whose mutation translates the payload, 4 of the 1,177 responses scored as successes across all five target models contain a non-English refusal, bounding any resulting overstatement at approximately 0.02 pp.

G.2 AGENT PASSPORT AND ATTESTATION GATE

Directives that arrive through the inter-agent, tool, and memory channels claim a principal: an agent id and a capability such as `execute_tool` or `write_memory`. WardenGuard verifies the claim before it reads the content. Specifically, a passport binds an agent id to an Ed25519 key and

a capability set; furthermore, issuance is hierarchical and monotone, a child's capabilities being a subset of its issuer's, so no delegation step widens authority, and a registry verifies a passport by walking its chain to a root. Additionally, capability tokens attenuate a passport in the style of macaroons, and each directive travels in an envelope carrying a nonce and a timestamp that the registry checks against replay. The gate, accordingly, returns one of three outcomes: REFUSE when any claimed principal fails verification, which blocks the case at zero judge cost and runs no detector; PASS when every claim verifies; and NO-CLAIM when the case carries only plain content channels, in which case the gate has no opinion. Since a refusal does not depend on the text, rewriting a directive cannot mint a signature, and consequently the residual attack success on claim-carrying attack classes is structural rather than a matter of attacker effort. Because Warden-Bench contains no signing infrastructure, moreover, the benchmark expresses a row's claims as (principal, capability) pairs checked against the set of principals the row's environment attests; the gate runs on that representation through the same decision code, and every number that involves the gate is disclosed as a simulation of attestation rather than a deployed one (Sec. 5.2). Finally, the envelope verifier is an adapter, so an external capability-token scheme can replace the built-in registry without touching the gate.

G.3 MEMORY INTEGRITY MONITOR

An agent's persistent memory is the one channel an attacker can write once and read many times, so WardenGuard gates writes rather than reads. Each write carries an integrity label from the lattice TRUSTED $>$ SEMI-TRUSTED $>$ UNTRUSTED, the meet of its source channel's label and its writing principal's label: an attested principal writing through a semi-trusted channel yields SEMI-TRUSTED, whereas an unattested one yields UNTRUSTED regardless of channel. The monitor enforces two rules. **NUIPM** (no untrusted instruction persists in memory) is an invariant on the stored state: a record whose content is instruction-bearing must carry a label at or above the policy floor, SEMI-TRUSTED by default. Instruction-bearing content from below the floor is refused outright; the gate does not sanitize, since a sanitizer is a second detector with its own error rate. Data from below the floor, however, may still persist, labeled as data, so nothing downstream can mistake it for an operator instruction. Whether a write is instruction-bearing is decided by a small local scorer, the verdict-token logits of Qwen2.5-1.5B-Instruct under a fixed prompt, thresholded at 0.4. **BGAD** (behavioral goal-anchor drift) is the second rule and catches what the first cannot, namely an accumulation of individually innocuous writes. The system prompt is embedded once as the anchor (BGE-M3); accepted writes move a windowed centroid, and a write whose post-commit drift from the anchor exceeds 0.9 is refused. Since both rules run inside a memory transaction (Sec. 4.8), a refused write never reaches memory, and every committed state satisfies NUIPM by construction. On Warden-Bench, each case is one episode: the leaves of the row's memory state are offered to the monitor as writes, and a refused leaf is removed from the state the judges see (*enforce* mode); an *observe* mode, meanwhile, lets every write persist and reports what the monitor would have refused, which constitutes the ablation of the monitor's effect. Because the benchmark records what the agent would write but not where the content came from, every write is assigned the UNTRUSTED label, so the monitor's refusal rate on benign memory is that of the strictest provenance setting, and it is accordingly reported beside the detection numbers rather than folded into them.

G.4 HASH-CHAINED AUDIT LOG AND ROLLBACK

Every decision the pipeline makes is appended to an audit chain whose entries are linked by hash: the hash of an entry is SHA-256 over the previous entry's hash and the canonical JSON of the record, so an edit to any past entry breaks every later hash and is detected by a single verification pass. The chain records the start of each episode, each gate and judge verdict with its scores and cost, and each memory commit, abort, and rollback. Furthermore, memory states are stored content-addressed, the address of a state being the SHA-256 of its canonical serialization, so identical states share one snapshot and a snapshot's address is its own integrity check. Writes, accordingly, are transactional: a transaction snapshots the current state, stages the write, and either commits, publishing the staged state and recording both addresses on the chain, or aborts. Rollback restores any earlier committed address and records the rollback as its own entry, so recovery from a poisoning is itself auditable; since the monitor's rules run inside the transaction, every state a rollback can return to satisfies the memory invariant. Isolation is single-writer per episode, which is stated rather than solved. Rollback is exercised by the unit tests and not by the benchmark, whose cases are single-turn episodes;

additionally, the evaluation runs persist their chains beside their results and record the outcome of the verification pass in each run summary, and the backbone-ablation run's chain of 71,778 entries verified end to end. The audit log, ultimately, is an engineering property that the other mechanisms rely on, not a detection claim.

G.5 Deployment Considerations

Structurally, the cascade bounds inference cost. Stage 1 runs on local hardware, Qwen3-30B-A3B-Instruct-2507 served by Ollama, and settles the clear cases it recognizes, so only 9.1% of Warden-Bench cases reach the frontier judge: 11 times fewer judge calls than a design that consults the judge on every case. That ratio is a property of traffic the local stage is calibrated for, and it is deliberately not constant: since a local clear requires the input to resemble the evaluated distribution, the share of cases reaching the frontier judge rises to 45–68% on the four external benchmarks, where the defense buys accuracy with frontier calls instead of saving them. Sec. 5.3 reports both. Each escalated verdict is one GPT-4.1-mini round-trip regardless of how many channels are populated, whereas a per-field judge issues one call per populated channel; Warden-Bench rows populate 2.20 channels on average and up to five. Stage 1's confidence rule is the per-input detector allocation of [33] made explicit, in that a fixed rule decides which inputs the frontier judge sees. Judge-LLM versioning, drift monitoring, and robustness to attacks targeting the channel-tagging convention or the deterministic decoder are, furthermore, deployment concerns rather than detection ones.

Which components to run is also a deployment choice, and the ablations of Sec. 5.5 price it in a way the aggregate figures obscure. Both the decoder and the trust annotation buy recall with false positives, and removing either lowers the benign false-positive rate: to 2.95% without the decoder, 2.27% without the annotation, and 1.63% without either, against 4.25% for the complete defender. All four configurations clear the F1 > 0.90, FPR $< 5\%$, FNR $< 15\%$ bar of Sec. 5.2, so a deployment that weighs false positives more heavily than the proposed work does could rationally ship a reduced configuration. What it would be buying, however, is visible only per tier. The decoder is worth $+19.4$ pp of recall on T11 and approximately nothing on any other tier; the annotation is worth $+10.9$ pp on T4, $+6.1$ pp on T5, $+4.1$ pp on T9, and $+4.0$ pp on T1, and approximately nothing on T11. The two are close to orthogonal, each carrying a different failure mode, and each contributes under two points to benchmark-wide recall only because the tiers it covers are a small share of the rows. An attacker selects the tier, so a configuration at 2.95% FPR that recovers 72.8% of encoded payloads is not a safer system than one at 4.25% that recovers 91.9%; it is instead a system with a narrower published weakness, which is the same reading Table A8 invites for the baselines. Since recent work [32] shows that strong adaptive attacks bypass most published prompt-injection defenses, App. A evaluates WardenGuard under that regime against the two strongest baselines, and gradient- and optimization-based attackers remain the open case. A production-ready architecture is therefore claimed, but not a deployed system; the operational engineering is outside the scope of the proposed work.

G.6 Construction

Each row carries an attack payload, an injected target task, an agent system prompt, and a binary label. Every attack payload was written for Warden-Bench by a tier-specific generator; the public corpora supply seed material, recorded per row in the `seed_source` column. Of the 17,702 attack rows, 8,956 are seeded from five public collections; verazuo/jailbreak_llms [20] (3,829 rows), deepset's prompt-injection corpus [21] (4,607 across three splits), and the taksec [22] (303), cranot [23] (143), and yunwei [24] (74) prompt-injection lists; whose entries the generators rewrite into the channel, encoding, and multi-turn forms the tier taxonomy requires. The remaining 8,746 rows carry no external seed: 509 are hand-written T10 seeds, and 8,237 are generated from tier-specific templates with generative-AI assistance. The 5,899 benign rows were sampled from clean NLP task corpora that share the agent contexts of the attack rows, so that surface features (tool names, file types, agent roles) cannot be used as a shortcut. Labels are assigned at construction: each row is emitted by a deterministic generator that places a known payload, so `is_injection` records which branch of the template produced the row rather than a post-hoc judgment, and the generators are released so every label is traceable to the code that wrote it. Additionally, the generator applies identifier-uniqueness, schema-conformance, and JSON-validity checks to tiers T0–T3; channels carrying code or prose payloads are non-JSON by design.

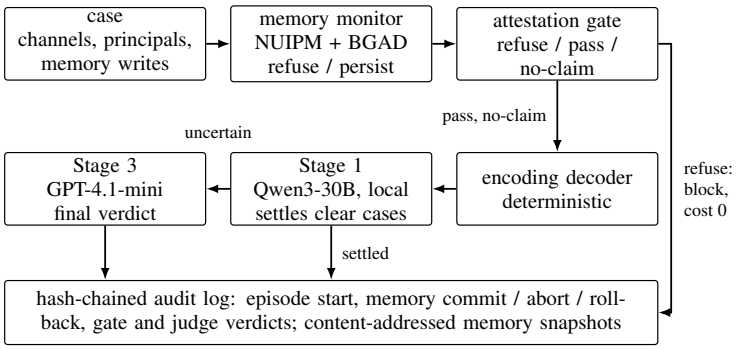

Figure A1: WardenGuard pipeline.

Furthermore, the label quality was audited independently: every row was re-classified by `gemini-3.7-flash`, blind to the assigned label and sharing no lineage with any evaluated defense, and over the 23,252 rows returning a parseable verdict, the auditor agrees with the benchmark on 91.9%, Cohen's κ=0.789. Agreement is lowest on T10 (76.0%), the benign tier (84.8%), and T9 (85.0%), and within each tier the disagreement runs almost entirely one way; this is the signature of a stricter deviation criterion rather than label noise. Appendix G gives the full breakdown.

The encoding tier T11 was assembled by sampling behavior-overriding payloads and re-encoding them in fourteen encoding families, with 465 matching benigns sampled from the same encoding distribution (33–34 per family) to prevent the encoding signature itself from acting as a shortcut feature.

G.7 Role in the Evaluation

Warden-Bench plays three roles in the proposed work. (i) It is the benchmark on which the five defenses are scored at the F1, FPR, and FNR level (Section 5.2). (ii) Its per-tier split supports the per-tier recall comparison (Table A8, Fig. A8) and the per-tier residual ASR comparison (Table A17, Fig. A12). (iii) Its 17,702-attack subset is the substrate of the chained residual ASR matrix (Section 5.4), where each attack is independently evaluated against five target language models in addition to the five defenses. Warden-Bench is also the home benchmark on which the WardenGuard prompt and decoder are tuned; Section 5.3 reports zero-shot transfer to four further benchmarks.

Label accounting The benchmark contains 17,702 injection-positive rows and 5,899 benign-labeled rows for a total of 23,601 (Table 1). The benign rows are distributed as follows: 4,660 in the BEN tier, 465 encoded benigns in T11 (33–34 per encoding family), and 774 control benigns interleaved across tiers T0–T4 (T0: 31, T1: 285, T2: 231, T3: 87, T4: 140). The control benigns are syntactically similar to their own tiers' attacks but do not contain a behavior-overriding instruction; consequently, they ensure that tiers T0–T4 cannot be solved by surface-level keyword filters. Furthermore, the 5,899 benigns are the false-positive-rate denominator used in Tables 2 and A11, the Warden row of Table 3, and the confusion counts in Fig. A7; the external-benchmark tables use their own benign denominators. The Combined column of Table A11, accordingly, pools over all 5,899 of them, as total false positives over total benigns, rather than averaging the per-slice rates.

subsectionPipeline, Threat Model, and Evaluation Scope

The order in which WardenGuard's six mechanisms run is what makes its cost profile possible: the attestation gate and the memory-integrity monitor both act before either judge, so a case claiming an unverifiable principal is refused, and a poisoned write withheld, at no detector cost. Fig. A1 gives that pipeline, and Fig. A2 gives the threat model it defends. Furthermore, two further constraints are fixed before any measurement is taken: what the judge can express at all is bounded by the channel taxonomy of Table A4, and the scope over which every number in Sec. 5 is computed is bounded by the benchmark inventory of Table A5. The remaining two figures, accordingly, condense results reported in full elsewhere; the cross-benchmark comparison in Fig. A3 and the chained residual ASR construction in Fig. A4.

Table A4: WardenGuard channel trust labels.

Channel	Trust	Typical content
`user_input`	UNTRUSTED	end-user prompt
`tool_output`	SEMI-TRUSTED	API or web response
`agent_memory_state`	TRUSTED	internal scratchpad
`conversation_history`	SEMI-TRUSTED	past dialogue turns
`retrieved_context`	UNTRUSTED	RAG snippets
`file_metadata`	UNTRUSTED	filenames, EXIF tags



Figure A2: Indirect prompt injection threat model.

G.8 Independent Label Audit

Label quality was audited independently. Every one of the 23,601 rows was re-classified by `gemini-3.7-flash`, a model sharing no lineage with GPT-4.1-mini or with any evaluated defense, from the channel contents alone and blind to the assigned label. The auditor sees the first 900 characters of each channel and the first 400 of the task instruction; T1 places its payload beyond that cut on 189 of 1,736 attack rows, so T1's agreement is a lower bound, and T1 is nonetheless included in the counts below. Over the 23,252 rows returning a parseable verdict, the auditor model agrees with the benchmark on 91.9%, Cohen's κ=0.789, with 1,887 rows disagreeing (Table A6, Fig. A5). Agreement exceeds 94% on eight of thirteen tiers, whereas it is lowest on T10 (76.0%), the benign tier (84.8%), and T9 (85.0%); these are the places where the deviation criterion of Sec. 1 is most demanding, and the same boundary Sec. 6 identifies as the source of T9's residual.

Within a tier, the disagreement runs almost entirely one way (Fig. A6), which distinguishes a criterion offset from label noise, since a mislabeling would scatter in both directions. Across the attack tiers, the auditor model declines to classify a constructed attack as an attack in 1,102 rows and makes the opposite call in 76 (71 of them in T11 and five elsewhere), and on the benign tier it labels 709 rows as attacks with no disagreement in the reverse direction. Eight of the twelve attack tiers show zero reverse disagreement. On T9 and the benign tier, the pattern reflects a stricter bar for deviation than the construction criterion applies, not an unreliable label set. T11 is the one genuinely two-way tier, at 89 against 71, the expected signature of encoded content being ambiguous in both directions.

T10, the lowest tier, is a separate case. Its declines concentrate on two techniques: SQL injection, 64.8% of its rows, and serialization attacks, 59.2%, while command injection and polyglot payloads, which carry overt executable directives, are declined at 0% and 1.4%. The concentrated pair are exploits of a downstream interpreter, a database engine, or an object deserializer, rather than attempts to redirect the agent, and they sit on the boundary of what counts as a prompt injection. At that boundary, the auditor model applies not a stricter rule but an unstable one: 118 of T10's 122 declines are on payload strings the same model labels as attacks on other rows, and 46% of its repeated payloads receive both labels, the highest within-payload inconsistency of any tier, against 22% on T9, 14% on T6, and at most 13% elsewhere. T10's agreement therefore measures definitional ambiguity at the interpreter-exploit boundary rather than a defect in its labels, and the tier is retained unchanged.

Additionally, a further 349 rows (1.48%) returned no parseable verdict and are excluded from the agreement figures rather than counted as agreement. The audit script, its per-row verdicts, and the disagreement set are released with the benchmark. The encoding tier T11 was assembled by sampling behavior-overriding payloads and re-encoding them in fourteen encoding families, with 465 matching benigns sampled from the same encoding distribution (33–34 per family) to prevent the encoding signature itself from acting as a shortcut feature.

Table A5: Evaluation benchmarks.

Benchmark	n	Source	Coverage
Warden-Bench	23,601	this work	T0–T11+BEN (T11 = 14 encodings)
PS-Bench	12,700	[15]	easy/med./hard
DS-Bench	1,200	[13]	6 NLP tasks
InjecAgent†	2,108	[6]	DH+DS+matched benigns
BIPIA	2,340	[7]	email/code/table
Total	41,949		

† InjecAgent: 1,054 attacks (510 direct-harm + 544 data-stealing, canonical attacks of [6]) plus 1,054 matched-volume benigns.

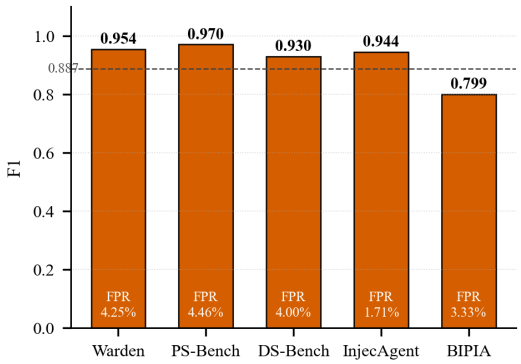

Figure A3: WardenGuard F1 across Warden and four external benchmarks.

G.9 Benchmark Mutation Layer

Tiers T6–T10 additionally pass each payload through a twenty-transform mutation engine, recorded per row in the `mutation_id` and `mutation_name` fields (9,465 rows); T11's 400 adaptive-encoding rows reuse `mutation_name` to record their encoding family, for 9,865 mutated attack rows in total. The transforms group into semantics-softening rewrites (SYNONYM_SWAP, VOICE_FLIP, REGISTER_SHIFT, SENTENCE_RESTRUCTURE, INDIRECT_PHRASING, GOAL_DECOMPOSITION, AUTHORITY_REFRAMING, NARRATIVE_EMBEDDING, SOCRATIC_ESCALATION), surface obfuscation (LEETSPEAK, HOMOGLYPH, ZERO_WIDTH_INSERT, MIXED_ENCODING, TEMPLATE_VARIABLE), language shifts (FULL_TRANSLATION, CODE_SWITCH), and format wrappers (JSON_WRAPPED, MARKDOWN_INJECTION, CODE_COMMENT_EMBED, MULTI_FORMAT_STACK). Row labels are assigned per tier at row construction, and the mutation is applied to the payload afterwards, so a label reflects the pre-mutation payload, and mutated rows are not individually revalidated. App. G.12.2 reports recall along this axis, and Sec. 6 discusses the consequence for T9. Furthermore, `mutation_name` is released with the benchmark so that this axis is separable by downstream users.

G.10 Overall Detection Detail

WardenGuard's verdicts on Warden-Bench resolve into 16,383 true positives, 251 false positives, 1,319 false negatives, and 5,648 true negatives. False positives are the smaller error mode by more than a factor of five, and that asymmetry is what the false-positive-rate comparisons of Sec. 5.2 turn on, since a defense can only be read as safer than WardenGuard if it trades away the recall those 1,319 misses already cost. The 5,899 benign rows are the denominator throughout. Fig. A7 renders the four counts.

G.11 Held-Out Split

The split exists to show that WardenGuard's configuration was not chosen on the rows it is scored on. Its held-out F1 lands within 0.1 pt of its development figure, the ordering of all five defenses is identical across the two splits, and no defense's development-to-test difference is significant, so

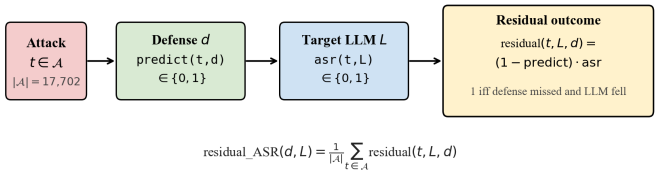

Figure A4: Chained residual ASR pipeline.

Table A6: Independent label audit by `gemini-3.7-flash`.

Tier	n	Agreement	Declined	Flagged
T0	383	95.3%	16	2
T1	2,021	90.6%	188	1
T2	1,246	99.0%	11	2
T3	1,606	98.6%	23	0
T4	323	100.0%	0	0
T5	2,000	96.7%	66	0
T6	2,497	94.4%	139	0
T7	2,500	95.1%	122	0
T8	1,998	98.3%	33	0
T9	1,954	85.0%	293	0
T10	508	76.0%	122	0
T11	1,556	89.7%	89	71
BEN	4,660	84.8%	0	709
All	23,252	91.9%	1,102	785

Declined: attack → benign. Flagged: benign → attack.

nothing in the comparison depends on which half a defense is read from. Table A7 gives each defense on both splits; every other Warden-Bench figure in the proposed work is computed on the full 23,601 rows.

G.11.1 Held-Out Protocol

Warden-Bench ships a frozen development/test partition. Rows are stratified on tier × label × mutation, giving 122 strata of at least seven rows each, and split 70/30 under a fixed seed: 16,501 development rows and 7,100 test rows. Both id sets are released with SHA-256 digests, and the partition rebuilds bit-for-bit from the seed. Every configuration choice was made on the development split. Furthermore, every Warden-Bench figure reported here other than Table A7's held-out column is computed on the full 23,601 rows; WardenGuard's configuration was chosen on the development split, and the held-out test split is reported alongside. No component of any defense is trained or fine-tuned on Warden-Bench, since every verdict is a fresh model call. The split, accordingly, separates the rows on which WardenGuard's configuration was chosen from rows on which nothing was chosen, so the held-out column carries no selection effect.

WardenGuard's held-out F1 is within 0.1 pt of its development figure, the ordering of all five defenses is identical on the two splits, and no defense's development/test difference is significant under Fisher's exact test after correcting for the five comparisons (WardenGuard: p=0.76); Table A7 gives each defense on both splits.

G.12 Per-Tier and Per-Mutation Results

This subsection details Per-Tier results and Per-Mutation results.

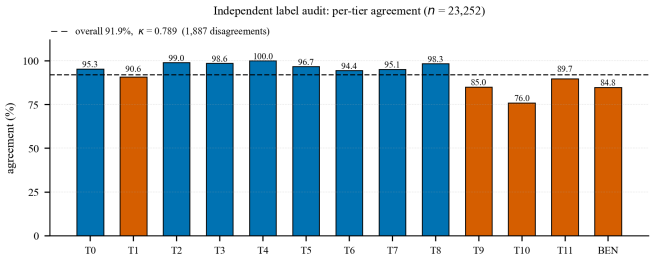

Figure A5: Per-tier agreement with the assigned labels.

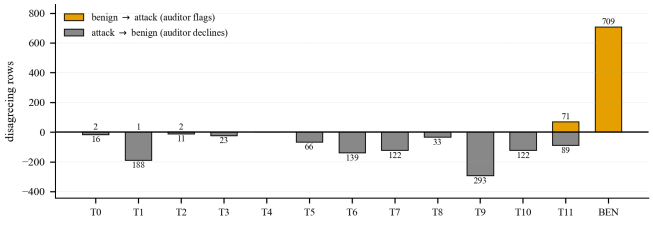

Figure A6: Disagreement direction by tier.

G.12.1 Per-Tier Recall

Table A8 breaks recall across the twelve attack tiers, and Fig. A8 overlays the same data graphically. WardenGuard leads on T1, T2, T5, T7, T8, and T11 (six tiers) and ties PromptSleuth on T3 (99.7%); it trails PromptSleuth by 7.8 pp on T0, 2.7 pp on T4, 0.8 pp on T10, and 0.4 pp on T6. PromptSleuth leads on those four tiers but operates at 48.6% FPR overall. T9 is the only tier where neither PromptSleuth nor WardenGuard leads recall (DataSentinel 81.3%), and DataSentinel's MCC of 0.026 over the full benchmark places it at chance, so that lead reflects prevalence rather than discrimination. The two tiers on which WardenGuard's measured recall is lowest are T9 (cross-turn fragmentation, 69.9%), discussed in Section 6, and T10 (interpreter-layer exploits, 89.8%); T11's benign controls carry the highest slice FPR (Table A11). For T9, the residual is predominantly definitional rather than a detection failure, and App. G.12.2 decomposes it by mutation.

Table A9 shows where each tier's decisions are made. Recall is taken over the tier's own attacks, and the gate, Stage 1, and judge columns are shares of those same attacks; the write-path column counts attacks defended only by a refused memory write, and the last two columns give the tier's benign rows and their false-positive rate, which tiers T5–T10 do not carry. Furthermore, the attestation gate absorbs all of T8 and the spoofed principals of T2 (48%) and T3 (59%) before any judge runs, and 16% of T11. On every other tier, the local Stage 1 settles at least 79% of the attacks; escalation to GPT-4.1-mini peaks on T5 (21.2%), followed by T9 (12.2%) and T6 (12.0%). Attacks defended only at the memory write path occur where the payload travels through memory: 57 in T6, 37 in T7, and 82 in T11.

G.12.2 Per-Mutation Recall

Tiers T6–T11 expose a mutation axis (Sec. 3) that the per-tier view of Table A8 conflates with attack class. Table A10 decomposes recall over the 9,865 mutated attack rows, sorted by WardenGuard recall: the upper block holds the twenty transforms of the mutation engine, the lower block the four additional encoding families of the T11 extension, and bold marks the best value in each column, with the All-mutated row bolded on its row maximum. To the authors' knowledge, no prior prompt-injection benchmark reports this decomposition.

Three observations follow. First, WardenGuard's recall spans 70.4–98.0% across mutations, a 27.6 pp range comparable to its 35.5 pp spread across tiers; the two axes are partially confounded, so reporting only one leaves a substantial source of variation unattributed. Second, grouping the transforms by the families of Sec. 3, WardenGuard averages 81.9% recall on the nine semantics-

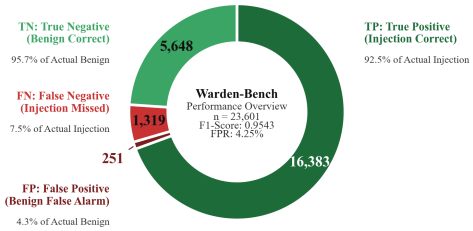

Figure A7: WardenGuard confusion counts on Warden.

Table A7: Development and held-out test performance.

	F1		MCC	
Defense	Dev	Test	Dev	Test
ProtectAI	0.2576	0.2568	−0.0310	−0.0283
DataSentinel	0.7538	0.7576	0.0240	0.0312
PromptArmor	0.8326	0.8426	0.5999	0.6159
PromptSleuth	0.8581	0.8614	0.3999	0.4171
WardenGuard	**0.9536**	**0.9525**	**0.8385**	**0.8332**

softening rewrites (4,241 rows), against 85.7% on surface obfuscation (2,404), 83.3% on language shifts (949), and 89.0% on format wrappers (1,871), while the four additional encoding families of the T11 extension reach 94–98% (96.2% pooled). The transforms that most reduce measured recall are therefore those that rewrite an override into a semantically milder request, not those that obfuscate its surface; Sec. 6 develops the consequence for T9. Third, the ordering is defense-specific: PromptSleuth leads on most engine transforms but falls to 49–58% on the four encoding families where WardenGuard reaches 94–98%, which is the per-mutation signature of the decoder ablation of Sec. 5.5.

G.13 Benign False-Positive Rate by Slice

G.13.1 Benign FPR

Table A11 decomposes the false-positive rate into the 4,660-case BEN tier, the 465 T11 benigns (33–34 per encoding family), and the 774 control benigns. WardenGuard's false positives concentrate in the BEN tier (4.74%) and among the T11 benigns (5.81%, 27 of 465), whereas on the 774 control benigns it records three (0.39%). PromptArmor leads every slice. Of WardenGuard's 251 false positives, 227 were confident Stage 1 blocks, and 24 were judge verdicts. Furthermore, the memory monitor's refusals on benign cases withhold a memory write without blocking the case and are therefore not part of these rates. The combined-FPR gap to PromptSleuth is 44.35.

G.14 Cross-Defense Comparison

G.14.1 Controlled Cross-Defense Comparison

The InjecAgent and BIPIA benchmarks were also run against three of the baseline defenses, PromptArmor, PromptSleuth, and DataSentinel, in Tables A13 and A14; ProtectAI was not evaluated on these benchmarks. BIPIA is scored on its full 2,340 rows, 2,250 attacks and 90 benigns, the latter carrying every false-positive rate in that table; on InjecAgent, the 1,054 attacks are shared across defenses, but the benign split is not, so precision, F1, and FPR there do not share a denominator.

Table A12 consolidates the F1 grid across all four external benchmarks. Two provenance limits apply to it. On PS-Bench and DS-Bench, the baseline entries are cited from the PromptSleuth Table 3 reconstruction [15] rather than re-run here, so differences in backbone snapshot, prompt formatting, or scoring rubric between that work and the proposed work are not held fixed; only the WardenGuard entries on those two benchmarks are measured in the proposed work. Additionally,

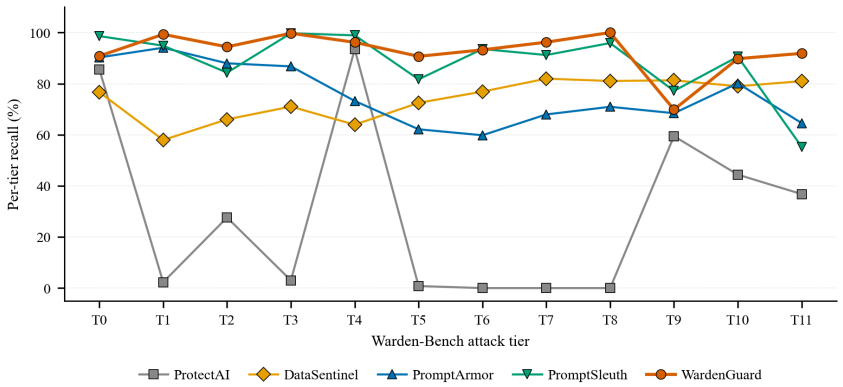

Figure A8: Per-tier recall, five defenses, twelve tiers.

Table A8: Warden-Bench per-tier recall (%).

Defense	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11
ProtectAI	85.6	2.3	27.6	3.0	93.4	0.8	0.0	0.0	0.0	59.4	44.4	36.8
DataSentinel	76.7	57.9	66.0	71.0	63.9	72.5	76.9	82.0	81.0	**81.3**	79.0	81.0
PromptArmor	90.3	94.1	88.0	86.8	73.2	62.2	59.8	68.0	71.0	68.4	80.2	64.6
PromptSleuth	**98.6**	94.9	84.3	**99.7**	**98.9**	81.7	**93.6**	91.2	95.9	77.2	**90.6**	55.3
WardenGuard	90.8	**99.3**	**94.4**	**99.7**	96.2	**90.7**	93.2	**96.2**	**100.0**	69.9	89.8	**91.9**

ProtectAI was not run on InjecAgent or BIPIA. The mean F1 of 0.911 quoted above is computed solely from WardenGuard's own measurements and is unaffected by the cited baselines.

LLM-judge defenses share GPT-4.1-mini; DataSentinel uses its native Mistral-7B+LoRA checkpoint. WardenGuard leads InjecAgent by 27 pp F1 over the next-best defense and is the only defense with FPR $\leq 5\%$ and recall $\geq 60\%$ on both benchmarks. Fig. A9 shows the grouped bars. Furthermore, DataSentinel collapses on InjecAgent to 4.6% recall, verified across two GPU runtimes, since the OPI-fine-tuned classifier does not generalize to the natural-language requests embedded in JSON tool responses that InjecAgent uses. The same brittleness appears on BIPIA in the opposite direction: 80.8% recall but 56.7% FPR, because the classifier flags any imperative-shaped retrieved content. WardenGuard's training-free prompt sidesteps both failure modes.

G.15 Residual ASR Detail

The matrix is composed, since residual is the product of the defense's verdict and the target LLM's undefended susceptibility, which is exact wherever the defense either stops a case, so that it never reaches the agent, or forwards it unchanged, so that the undefended measurement is the end-to-end outcome. WardenGuard is a filter and not a rewriter, so the only exceptions are the 28 attacks (0.16%) it allows through after the integrity monitor has sanitized their memory state; running those end to end through all five target LLMs moves no column by more than 0.04 pp, and downward in every case. The tighter constraint is reproducibility: re-running the identical undefended input through the same judge reproduces its verdict on 80% of cells, since these are sampled generations scored by lexical match. Aggregates over 17,702 attacks are stable to a few hundredths of a point; individual cells of Table A16 are not, and no conclusions are drawn from them.

Per cell, every defense is scored on the same 17,702 attacks against the same target LLM, so the 20 pairwise (defense, LLM) comparisons are paired, and these are tested with McNemar's exact test on the discordant counts, with Holm correction across all 20 and BCa bootstrap intervals (10^4 resamples) on the paired difference. Table 4 gives the outcome by competitor.

Table A9: WardenGuard on Warden-Bench by tier.

Tier	Attacks	Recall (%)	Gate (%)	Stage 1 (%)	Judge (%)	Write path (attacks)	Benigns	FPR (%)
T0	360	90.8	0.0	92.2	7.8	0	31	3.23
T1	1,736	99.3	0.0	97.9	2.1	0	285	0.35
T2	1,039	94.4	48.1	47.4	4.4	0	231	0.00
T3	1,519	99.7	59.2	40.0	0.8	0	87	1.15
T4	183	96.2	0.0	99.5	0.5	0	140	0.00
T5	2,000	90.7	0.0	78.8	21.2	0	—	—
T6	2,498	93.2	0.0	88.0	12.0	57	—	—
T7	2,500	96.2	0.0	93.5	6.5	37	—	—
T8	2,000	100.0	100.0	0.0	0.0	0	—	—
T9	1,958	69.9	0.0	87.8	12.2	0	—	—
T10	509	89.8	0.0	90.4	9.6	0	—	—
T11	1,400	91.9	16.4	74.8	8.8	82	465	5.81
BEN	—	—	—	—	—	—	4,660	4.74
All	17,702	92.5	20.5	71.5	8.0	176	5,899	4.25

Table A10: Per-mutation recall (%) on the mutated attack rows of T6–T11.

Mutation	n	ProtectAI	DataSent.	PromptArm.	PromptSl.	WardenGuard
INDIRECT_PHRASING	483	10.6	81.6	50.7	85.7	70.4
CODE_SWITCH	455	10.8	78.5	54.3	84.8	75.8
VOICE_FLIP	451	8.9	71.8	54.8	88.5	78.3
MULTI_FORMAT_STACK	466	14.2	90.6	68.7	84.3	79.2
SYNONYM_SWAP	456	10.5	82.5	55.0	85.7	79.8
LEETSPEAK	494	15.8	80.6	51.8	85.0	80.6
SOCRATIC_ESCALATION	465	17.4	51.4	63.4	94.6	80.9
ZERO_WIDTH_INSERT	463	11.4	81.2	65.4	87.3	81.9
HOMOGLYPH	490	12.2	80.8	66.7	86.5	82.0
REGISTER_SHIFT	486	14.0	80.0	61.1	82.1	82.9
SENTENCE_RESTRUCTURE	468	11.3	79.1	60.7	87.4	83.1
AUTHORITY_REFRAMING	484	12.8	80.6	65.5	89.9	84.1
NARRATIVE_EMBEDDING	483	16.4	81.8	60.2	90.7	84.9
CODE_COMMENT_EMBED	477	14.7	80.3	74.8	92.9	87.6
FULL_TRANSLATION	494	16.6	79.8	72.9	92.3	90.3
TEMPLATE_VARIABLE	483	19.7	78.1	78.1	92.5	91.7
GOAL_DECOMPOSITION	465	16.3	84.5	81.7	97.0	92.5
MIXED_ENCODING	474	17.5	**91.4**	88.2	96.4	92.6
JSON_WRAPPED	465	20.9	83.7	80.9	96.1	93.1
MARKDOWN_INJECTION	463	21.4	84.2	**89.0**	**98.1**	96.1
rot47	100	**33.0**	87.0	64.0	54.0	94.0
atbash	100	31.0	83.0	64.0	58.0	95.0
base32	100	32.0	79.0	64.0	49.0	**98.0**
base85	100	**33.0**	82.0	64.0	51.0	**98.0**
All mutated	9,865	15.4	80.2	67.1	**88.4**	84.9

G.15.1 Heatmap of the Marginal Matrix

Table A15 colors the five defense rows of Table 6 from green (low residual) to red (high residual), making the matrix scannable. Two facts emerge. First, WardenGuard has the lowest residual ASR of the LLM-judge defenses on the three most-resistant target LLMs, though DataSentinel is marginally lower on GPT-OSS-120B (2.49% vs. 2.54%), and none of the three WardenGuard-over-PromptSleuth margins survives Holm correction, while PromptSleuth is marginally lower than WardenGuard on the two most-vulnerable. Second, the variance across LLMs is small for WardenGuard (range 2.54–4.74%, Δ=2.20 pp) and large for ProtectAI (11.30–29.76%, Δ=18.46 pp). A strong defense, accordingly, flattens the LLM-vulnerability landscape.

Table A11: Benign false-positive rate by slice.

Defense	BEN (n=4,660)	T11-benign (n=465)	Control (n=774)	Combined (n=5,899)
ProtectAI	18.18%	43.2%	3.6%	18.24%
DataSentinel	72.42%	76.1%	72.2%	72.69%
PromptArmor	**3.43%**	**3.9%**	**0.0%**	**3.02%**
PromptSleuth	48.52%	10.8%	71.8%	48.60%
WardenGuard	4.74%	5.81%	0.39%	4.25%

Table A12: Cross-defense by cross-benchmark F1.

Defense	InjecAgent	BIPIA	PS-Bench	DS-Bench
DataSentinel	0.0871	**0.8825**	0.485	**1.000**
PromptArmor	0.6734	0.7611	0.911	0.978
PromptSleuth	0.6214	0.3707	0.928	**1.000**
WardenGuard	**0.9444**	0.7994	**0.9703**	0.9296

G.15.2 WARDENGUARD PER-TIER PER-LLM CUBE

Table A16 reports WardenGuard's measured residual ASR across all $12\times5=60$ (tier, LLM) cells. The cube exposes the conjunction of two independent factors, attack class and target LLM susceptibility, that the marginal matrix conflates. Fig. A11 renders the cube as a heatmap. Three features of it bear on the tier analysis. The row mean is dominated by T9 (5.47%), T11 (4.20%), and T5 (3.45%). T5 is the tier where WardenGuard is weakest on genuine overrides; T9's cell is inflated by the definitional gap of Sec. 6, since the defense applies a narrower override criterion than the deviation criterion used by both the row label and the attack-success judge, so task-substitution cases pass the defense and are then scored as successes. Column variance, moreover, is tier-driven rather than LLM-driven: T11 varies only 3.2–5.6% across the five LLMs, while T9 varies 3.5–7.9%. Thirty-eight of the sixty cells sit at or below 2% residual ASR, and the worst of them (T9 on Mistral-S) is 7.87%, well below the no-defense baseline of 34.27% on the same LLM.

G.15.3 PER-TIER, PER-DEFENSE COMPARISON

Sliced by Warden tier and marginalized across the five LLMs, the residual reduction tracks the undefended baseline (Table A17, Fig. A12). WardenGuard leads on T1, T5, T8, and T11. T5, the tier with the highest no-defense baseline (36.38%), receives the largest absolute residual reduction (30.82 pp), whereas T11, where the encoding decoder contributes most of the recall, yields a 27.89 pp reduction from its 33.00% baseline.

G.15.4 COMPUTED-VS-MEASURED RECONCILIATION

A simpler residual estimate is the product $(1 - \text{recall}_d) \cdot \text{ASR}_L$, treating defense miss events and LLM susceptibility as independent. For the LLM-judge defenses (PromptArmor, PromptSleuth, WardenGuard), this estimate is optimistic on 13 of 15 cells, by up to 1.67 pp, since the attacks they fail to detect are stealthier than the average attack and therefore fool the target LLM at a higher rate than the population mean. The effect shrinks as recall rises: the gap is 0.70–1.67 pp for PromptArmor, -0.12–1.06 for PromptSleuth, and 0.07–0.57 for WardenGuard, because a defense that misses less leaves a smaller and less atypical residue. DataSentinel sits below the independence estimate on every cell, whereas ProtectAI straddles it. The per-case matrix pairs each attack's own verdict with its own measured susceptibility; the independence formula over aggregates is retained only as a sanity check against it.

G.16 ABLATION AND BACKBONE PORTABILITY DETAIL

G.16.1 BACKBONE PORTABILITY

The judge backbone is a free parameter of every LLM-judge defense, so a comparison run on one backbone leaves open whether the ranking it produces is a property of the defenses or of that model.

Table A13: InjecAgent cross-defense.

Defense	F1	Prec.	Recall	FPR	FNR
WardenGuard	**0.9444**	0.9816	**91.0%**	1.71%	**9.01%**
PromptArmor	0.6734	**1.0000**	50.8%	**0.00%**	49.24%
PromptSleuth	0.6214	0.9737	45.6%	3.71%	54.36%
DataSentinel	0.0871	**1.0000**	4.6%	**0.00%**	95.45%

Table A14: BIPIA cross-defense.

Defense	F1	Prec.	Recall	FPR	FNR
DataSentinel	**0.8825**	0.9727	**80.8%**	56.67%	**19.24%**
WardenGuard	0.7994	0.9980	66.7%	3.33%	33.33%
PromptArmor	0.7611	0.9978	61.5%	3.33%	38.49%
PromptSleuth	0.3707	**1.0000**	22.8%	**0.00%**	77.24%

Consequently, the two baselines that use a judge were re-run over the full benchmark with the backbone replaced by Qwen3-30B-A3B-Instruct [17], an Apache-2.0 open-weight mixture-of-experts model with 3B active parameters; prompts, input builders, decoding parameters, and token budgets are byte-identical to the GPT-4.1-mini arm, and a digest gate over every prompt builder and parser aborts the run on drift. For WardenGuard, the swap replaces the final judge only: Stage 1 is that same open-weight model served locally and is unchanged, so the ablation asks what the frontier judge contributes on the cases Stage 1 escalates. The gate, the memory monitor, the decoder, and the audit log run identically in both arms; the ablation arm's chain of 71,778 entries verified end to end.

The ranking is preserved. Under both backbones, WardenGuard leads on F1 and on MCC (Table A18, Fig. A13), and all four within-backbone margins are significant under McNemar's exact test with Holm correction over the family of seven paired comparisons ($p < 10^{-300}$ in every case; 23,601 rows scored in every arm). WardenGuard's F1 advantage is $+0.0952$ (95% BCa CI $[+0.0914, +0.0991]$) over PromptSleuth and $+0.1187$ $[+0.1144, +0.1230]$ over PromptArmor with GPT-4.1-mini as the judge, and $+0.0749$ $[+0.0714, +0.0785]$ and $+0.1220$ $[+0.1177, +0.1264]$ with Qwen3-30B. On MCC, the margins are $+0.4330$ and $+0.2334$ under GPT-4.1-mini and $+0.4156$ and $+0.2644$ under Qwen3-30B; Fig. A14 plots them with their intervals.

Stage 1 settles the same cases in both arms, since the escalation decision is identical on all 23,601 rows, so the swap changes only the 2,152 escalated cases, 9.1% of the benchmark. On those, the open-weight judge blocks 1,004 of 1,421 attacks against 751 for GPT-4.1-mini, and 190 of 731 benign cases against 24. The whole-benchmark effect is the sum: recall rises from 92.5% to 93.9%, F1 by $+0.0025$ $[+0.0009, +0.0041]$ ($p_{\text{Holm}} = 2.9\times10^{-2}$), MCC by an insignificant $+0.0004$ $[-0.0049, +0.0058]$, and the false-positive rate from 4.25% to 7.07%. The open-weight judge is the more aggressive one, converting uncertain cases into blocks in both directions. PromptSleuth's F1 also rises ($+0.0227$, $p = 8.2\times10^{-21}$) while its MCC gains only $+0.0178$, the gain again bought by flagging more inputs, and its false-positive rate moves $+14.04$ pp; PromptArmor's F1 change is indistinguishable from zero (-0.0009, $[-0.0055, +0.0038]$) at $+4.86$ pp of false positives. WardenGuard's false-positive movement, $+2.81$ pp, is confined to the escalated cases; the 90.9% of the benchmark that the gate and Stage 1 settle does not move at all.

Per tier, the open-weight judge gains most where the escalated cases concentrate: recall on T9 rises from 69.9% to 74.1%, on T6 from 93.2% to 95.9%, on T7 from 96.2% to 97.8%, and on T2 from 94.4% to 96.7%, while T0 falls from 90.8% to 88.1% (Fig. A15). The cost lands on the benign side of the encoding tier: the T11-benign false-positive rate rises from 5.81% to 15.91%, and the BEN rate from 4.74% to 7.10%. The deterministic decoder is model-independent and unaffected; the judge is readier to read a decoded artifact as hostile.

Backbone selection for this class of defense is governed by output-format compliance rather than by parameter count. The canonical judge parsers recover a verdict by keyword when a reply is not valid JSON, so a backbone that disregards the output contract still returns verdicts, parser artifacts rather than judgments, and the resulting metrics describe the parser. Since the token budgets are themselves

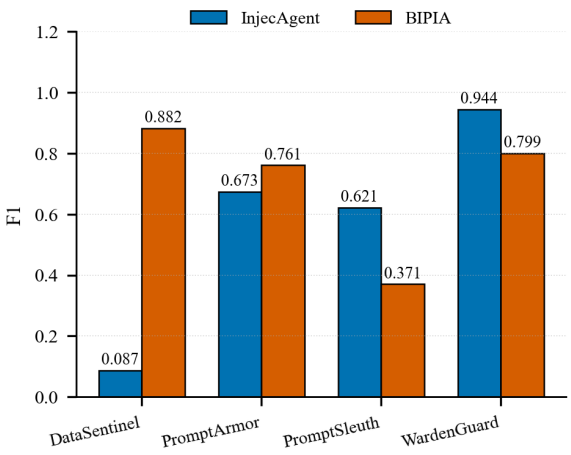

Figure A9: Cross-defense F1 on two benchmarks.

Table A15: Residual ASR, heat-shaded.

Defense	GPT-OSS	Nemotron	Gemma	L-70B	Mistral	Mean
ProtectAI	11.30%	15.01%	18.56%	18.05%	29.76%	18.54%
DataSentinel	2.49%	3.21%	3.87%	3.91%	6.03%	3.90%
PromptArmor	5.01%	5.78%	7.52%	6.93%	11.00%	7.25%
PromptSleuth	2.78%	2.92%	3.67%	3.14%	4.16%	3.34%
WardenGuard	1.45%	1.61%	2.15%	1.93%	2.62%	1.95%

part of the protocol being reproduced, a candidate must comply inside them. Table A19 reports compliance for four open-weight candidates at the canonical budgets over 195 stratified rows, one call per row for PromptSleuth and WardenGuard, one per populated field for PromptArmor, so n=195 and n=396 respectively. The spread is wide and is not ordered by size, with an 8B model at full compliance on the 16-token PromptArmor prompt yet none on the 100-token WardenGuard prompt, where it emits free-text analysis instead of the verdict object. Qwen3-30B returned a parseable verdict on every escalated case it judged in the ablation; two cases failed on transport, were marked invalid, and are excluded from every figure.

Table A16: WardenGuard residual ASR per tier and LLM.

	GPT-OSS	Nem.	Gem.	L-70B	Mis.	Mean
T0	0.56	2.50	2.22	1.94	1.39	1.72
T1	0.00	0.12	0.00	0.00	0.00	**0.02**
T2	0.39	0.48	0.39	0.87	0.48	0.52
T3	0.00	0.00	0.13	0.00	0.13	0.05
T4	1.09	0.00	0.00	0.55	2.19	0.77
T5	3.05	2.15	4.40	2.60	5.05	3.45
T6	1.64	1.56	2.76	2.04	3.52	2.31
T7	0.92	0.80	1.20	1.16	1.88	1.19
T8	0.00	0.00	0.00	0.00	0.00	**0.00**
T9	3.52	4.09	5.57	6.33	7.87	5.47
T10	0.98	1.57	1.77	1.77	2.55	1.73
T11	3.50	5.64	4.43	4.21	3.21	4.20
Mean	1.45	1.61	2.15	1.93	2.62	**1.95**

Table A17: Per-tier residual ASR (mean across LLMs).

Tier	No def.	ProtectAI	DataSent.	PromptArm.	PromptSl.	WardenG.
T0	18.94	2.94	1.78	1.56	**0.28**	1.72
T1	7.42	7.32	2.52	0.21	0.31	**0.02**
T2	7.55	3.29	1.54	2.00	0.54	**0.52**
T3	8.15	7.95	2.20	0.76	**0.04**	0.05
T4	8.20	0.98	3.28	2.08	0.55	**0.77**
T5	36.38	36.23	7.91	11.91	7.79	**3.45**
T6	29.42	29.42	5.71	13.02	2.46	**2.31**
T7	24.74	24.74	3.18	9.34	2.97	**1.19**
T8	21.26	21.26	3.21	6.77	1.17	**0.00**
T9	19.96	7.43	**2.12**	6.16	3.18	5.47
T10	23.03	13.01	**1.10**	3.54	1.38	1.73
T11	33.00	19.44	6.66	11.91	13.79	**4.20**

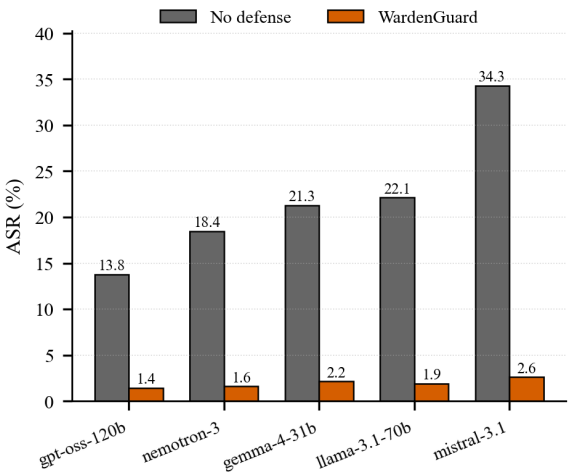

Figure A10: Per-LLM ASR with and without WardenGuard.

Table A18: Backbone portability on the full benchmark.

Defense	Backbone	F1	MCC	Prec.	Rec.	FPR
PromptArmor	GPT-4.1-mini	0.8356	0.6047	**0.9863**	72.5%	**3.02%**
	Qwen3-30B	0.8347	0.5742	0.9655	73.5%	7.88%
PromptSleuth	GPT-4.1-mini	0.8591	0.4051	0.8438	87.5%	48.60%
	Qwen3-30B	0.8818	0.4229	0.8204	**95.3%**	62.64%
WardenGuard	GPT-4.1-mini	0.9543	0.8381	0.9849	92.5%	4.25%
	Qwen3-30B	**0.9567**	**0.8386**	0.9755	93.9%	7.07%

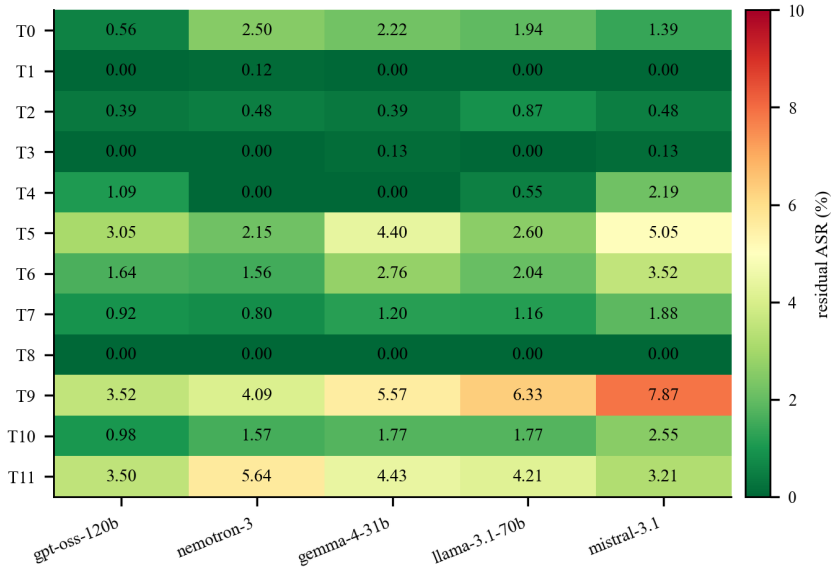

Figure A11: WardenGuard residual ASR, 60 cells.

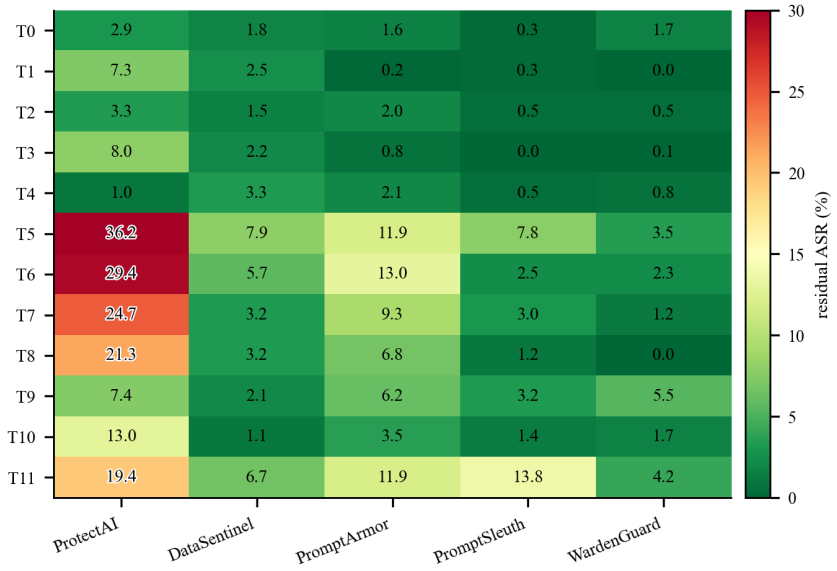

Figure A12: Per-tier per-defense residual ASR.

Table A19: Output-format compliance at the canonical budgets.

Backbone	Size	License	PA (16)	PS (20)	WG (100)
Qwen3-30B-A3B	30B/3B	Apache-2.0	100%	100%	98%
Ministral-14B	14B	MRL	100%	100%	69%
Ministral-8B	8B	MRL	100%	100%	9%
Granite-4.1-8B	8B	Apache-2.0	100%	17%	0%

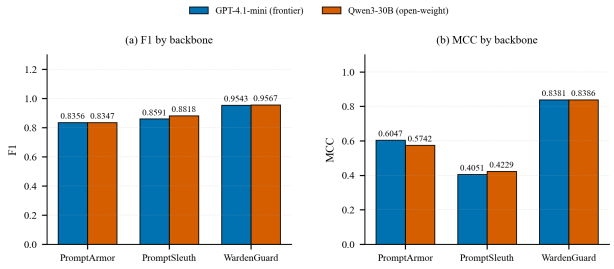

Figure A13: Ranking under both backbones.

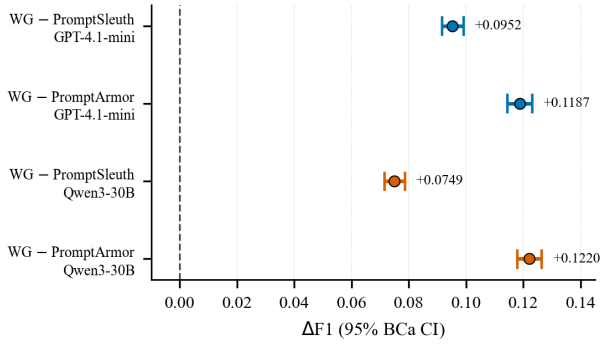

Figure A14: WardenGuard advantage with BCa intervals.

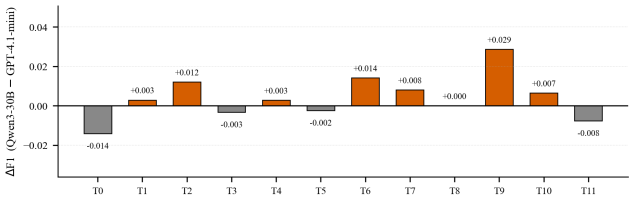

Figure A15: WardenGuard per-tier change under the open-weight backbone.