SECFLOW: LEARNING TO INTERPRET AND USE SECURITY EVIDENCE IN LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Understanding the surface meaning of security evidence does not ensure that models interpret its security implications or use it correctly. We investigate how reliably large language models (LLMs) understand security facts and use evidence to guide decisions. Using vulnerability information extraction and patch localization as complementary study settings, we identify limitations in both capabilities across seven commercial LLM systems. Our audit attributes more than 90% of incorrect localization runs to dominant reliance on surface cues and documents cases where systems inspect the correct patch but fail to select it. We introduce SECFLOW, which represents textual and program evidence as source-linked behavior flows encoding operations, conditions, and effects. It evaluates these flows against the vulnerability under analysis and injects the routed memory into later language-model layers through gated attention to guide answer generation. Built on Qwen2.5-Coder-7B, SecFlow improves both security-fact understanding and evidence-guided decision-making. Across the two study settings, it increases grounding F1 by 16.51 percentage points and achieves 65.24% Top-1 accuracy and 95.92% Hit@5 in candidate selection, surpassing the strongest evaluated commercial baseline by 40.13 percentage points.

1 INTRODUCTION

Cybersecurity is becoming an explicit focus of large language model (LLM) development, including GPT and Claude systems trained or adapted for vulnerability analysis (OpenAI, 2026; Anthropic, 2026). Benchmarks such as CyberGym and CyBench assess these capabilities through vulnerability reproduction and capture-the-flag challenges (Wang et al., 2026b; Zhang et al., 2025). However, task success combines security understanding with search, tool use, and execution. It does not separately establish whether a model correctly identifies the relevant security facts or bases its conclusion on the evidence (Ullah et al., 2024; Wang et al., 2026a). A failed attempt, for example, may reflect a misunderstood vulnerability condition or an execution failure.

This distinction matters because security implications depend on relationships among facts. Whether a program operation is unsafe depends on its inputs, execution conditions, and applicable safeguards (Guo et al., 2025). Recognizing identifiers, security terminology, or local code patterns can help locate relevant material, but cannot establish these relationships on its own (Li et al., 2025). Reliable analysis requires models to recover the supported facts, interpret their roles in the security problem, and determine what conclusions the evidence supports. We investigate how reliably current systems perform these forms of security-fact grounding and evidence-guided decision-making.

To study these capabilities, we curate a shared corpus of 15,365 vulnerability instances from the Open Source Vulnerabilities (OSV) database (OSV, n.d.) and evaluate seven commercial LLM systems in a common Hermes agent environment (Nous Research, n.d.). Vulnerability information extraction assesses recovery of annotated security facts and their roles from descriptions alone. Patch localization adds repository code and commit history, requiring models to connect the reported problem with candidate repair behaviors. We complement the task scores with audits of retrieved and inspected evidence, ranking signals, and final outputs. This design examines both what facts systems recover and how their observable decisions align with the available evidence.

The study reveals limitations in both capabilities. The best grounding F1 is 48.48%, while the best patch-localization Top-1 accuracy is 23.88%. Our audit identifies dominant reliance on surface

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.

cues in more than 90% of incorrect localization runs. These cues can promote related non-fixes or exclude the actual repair. In one case, a system inspects the correct patch but rejects it because its commit message fails a matching rule. The findings reveal a *security-event grounding gap* in recovering supported facts and an *evidence-to-decision gap* in selecting and using evidence for the final judgment.

To address these gaps, we introduce SECFLOW, a supervised model-adaptation method for improving how LLMs interpret and use security evidence. SecFlow keeps the pretrained model frozen and trains additional behavior-grounding, evidence-routing, and attention modules. Annotations of security roles, behavioral relations, and vulnerability–repair alignment supervise these modules to connect evidence with security judgments. At inference, SecFlow organizes textual and program evidence into source-linked behavior flows and assesses how each flow supports or challenges the conclusion. A gated cross-attention adapter then integrates the routed behavior memory into later decoder layers, allowing these relationships to influence the computation that generates the final answer.

We implement SecFlow on a Qwen2.5-Coder-7B backbone (Hui et al., 2024). It more than doubles security-fact recall from 29.39% to 59.38% while also improving precision. And it outperforms all seven evaluated commercial LLMs in patch localization. Compared with the strongest commercial baseline for each metric, SecFlow achieves $2.60\times$ the Top-1 accuracy (65.24%) and $2.50\times$ the Hit@5 (95.92%). Comparisons with standard supervised fine-tuning and component ablations further support the value of explicitly modeling and using security evidence.

We make three contributions.

- We develop a diagnostic study of security-fact grounding and evidence-guided decision-making, combining structured annotations with a controlled seven-system evaluation and trace-based failure analysis.
- We introduce SecFlow, which grounds security evidence into behavior flows and connects event-conditioned evidence routing with language-model prediction.
- We construct a manually verified patch-localization benchmark and demonstrate improvements over commercial systems and a same-backbone SFT baseline.

2 PRELIMINARY STUDY

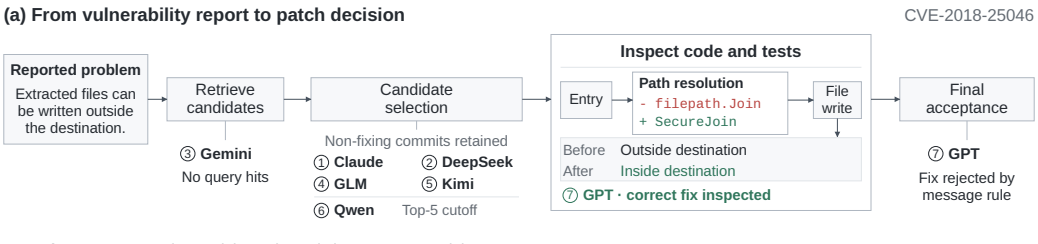
2.1 PROBLEM SETTING

To investigate whether LLM-based systems can go beyond surface relevance, we examine their ability to interpret evidence in its security context and use it to make decisions. Vulnerability information extraction examines the first ability through report interpretation, requiring models to recover supported facts and their security roles (Yitagesu et al., 2023). Patch localization examines the second by tracing a reported vulnerability to its repair (Li et al., 2024). This joint text–code setting requires connecting the reported problem with candidate code changes to distinguish an actual fix from a superficially related commit.

2.2 MOTIVATING EXAMPLE

Figure 1 illustrates a Cloud Foundry Archiver vulnerability that allows extracted files to be written outside the intended directory (GitHub Advisory Database, 2022). The accepted patch prevents these writes (Jolicoeur, 2018), but its commit message has little textual overlap with the later advisory.

Across the seven systems, identifier searches return no candidate, while title and entity matches favor changes unrelated to the repair. The figure shows the specific cues and resulting candidates. More strikingly, the GPT system ranks the correct patch first and inspects its diff, yet its acceptance criterion for this candidate requires both the commit message and the diff to meet advisory-term matching thresholds. Although the diff meets its threshold, the commit message does not, causing the correct patch to be rejected despite the available repair evidence. This case illustrates how surface-matching requirements can override repair evidence in the final decision.



(b) Surface cues and observed outcomes

System	Surface cue	Surfaced change / outcome
① **Claude**	“fix” / “path”	CI script paths
② **DeepSeek**	“file(s)” / “archiver.go”	Test-helper types
③ **Gemini**	Advisory IDs / phrases	No matching candidate
④ **GLM**	Permission wording / extractor file	File attributes (xattr)
⑤ **Kimi**	“cloud” / “foundry” / “archiver”	Dependency updates
⑥ **Qwen**	“paths”	Path separators / fix below cutoff
⑦ **GPT**	Hard message-matching rule	Correct fix inspected, then rejected

Figure 1: Motivation Example: CVE-2018-25046.

2.3 Study Design

Datasets. We examine both gaps using the same 15,365 vulnerability instances derived from OSV. Starting from the July 15, 2026 snapshot of 801,067 records, we retain 60,292 non-withdrawn vulnerabilities with commit, pull-request, compare, or pull-request patch references. Resolving their target repositories yields the shared set of 15,365 instances.

For the grounding gap, each system receives only the original vulnerability description and extracts the security facts it supports. The annotation schema covers triggering and prerequisite conditions, hazardous objects and behaviors, missing constraints, root causes, security impacts, and affected or fixed software information. For the evidence-to-decision gap, each system receives the same description together with access to the corresponding repository's source code and commit history.

Evaluation metrics. Grounding requires recovering a set of supported facts. We therefore use Precision, Recall, and F1 over normalized slot–value objects to capture both incorrect extractions and omissions. Localization instead requires choosing an accepted fix, not enumerating all acceptable commits. Set-level F1 would ignore candidate order and penalize omitted alternatives. We report Top-1 for the final primary choice, counting abstention as incorrect, and Hit@5 for accepted-fix coverage within a short candidate list.

Models and common agent environment. We evaluate the same seven flagship commercial LLMs on both tasks: `GPT-6-astra`, `Kimi-K3`, `Claude-fable-5.1`, `DeepSeek-v4-pro`, `Qwen-3.8-max`, `GLM-5.3`, and `Gemini-3.6-flash`.

All seven models run in Hermes agent framework with the same configuration and bundled skills, using the highest provider-supported reasoning effort. Within each task, they receive identical instructions, tool interfaces, and output requirements. Grounding exposes only the vulnerability description and output schema, while localization additionally provides repository code and commit history. External Web access is disabled, and ground-truth labels and audit records remain hidden during inference.

2.4 Security-Event Grounding Gap

Before repository evidence can be evaluated, a model must reconstruct the security event expressed in natural language. Each system extracts structured security facts from the original vulnerability description. The grounding block of Table 1 shows that no model achieves an F1 above 50%. GPT performs best at 48.48%. Recall remains below 49% for every model, showing that many triggering conditions, hazardous behaviors, missing constraints, and impacts are omitted. Precision is also limited, indicating frequent unsupported or incorrectly assigned security facts.

Table 1: Commercial LLM performance across the grounding and decision stages.

	Security-event grounding			Patch localization			
Model	Prec. ↑	Rec. ↑	F1 ↑	Top-1 ↑	Hit@5 ↑	Tokens ↓	Cost ↓
`GPT-6-astra`	**51.20%**	46.04%	**48.48%**	**23.88%**	**42.02%**	104.66M	$451.40
`Gemini-3.6-flash`	37.50%	38.85%	38.16%	22.40%	29.03%	**16.04M**	**$23.98**
`DeepSeek-v4-pro`	45.97%	46.09%	46.03%	21.84%	36.62%	30.10M	$100.80
`Kimi-K3`	44.36%	47.95%	46.09%	16.24%	23.74%	66.54M	$194.79
`Claude-fable-5.1`	45.77%	46.76%	46.26%	15.22%	24.19%	53.86M	$313.10
`Qwen-3.8-max`	45.60%	**48.74%**	47.12%	14.85%	26.39%	35.84M	$100.90
`GLM-5.3`	33.55%	40.79%	36.82%	13.92%	22.11%	64.26M	$128.52

Observation 1. *Security grounding requires understanding the role of evidence in a security context, not just its surface meaning.*

	F1 Identifier	F2 Security Narrative	F3 Entity / Tech Term	F4 Version / Time	F5 Local Code Shape	F6 Patch Role / Event	F7 Evidence Integration	F8 Fixed Gate	O1 Tool / Runtime	O2 Data / Mapping
GPT-6-astra	6.8	11.4	3.9	**65.7**	8.4	0	3.1	0	0.6	< 0.1
Gemini-3.6-flash	10.1	19.7	27.9	**41.6**	0	0	0	0	0.6	0
DeepSeek-v4-pro	2.2	**50.2**	29.0	4.0	0	13.6	0	0	1.0	0.1
Kimi-K3	8.2	**34.1**	25.2	27.6	0	< 0.1	0	0.2	4.6	< 0.1
Claude-fable-5.1	7.9	**42.4**	7.0	0.6	40.4	1.1	< 0.1	0	0.6	< 0.1
Qwen-3.8-max	4.2	31.8	5.0	**47.6**	0	10.1	0	0.7	0.5	0.1
GLM-5.3	4.4	**82.1**	0.4	1.2	0	1.4	0	0.7	5.6	4.2

Figure 2: Dominant observable failure mechanisms in the Preliminary Study.

2.5 Evidence-to-Decision Gap

The localization experiment adds repository evidence and asks each system to identify the commit that repairs the grounded security event. The localization block of Table 1 shows that the best Top-1 accuracy is 23.88%, while the best Hit@5 is 42.02%. Across all model–event runs, the micro-average Top-1 accuracy is 18.34% and the micro-average Hit@5 is 29.16%. We audit the execution traces to examine the queries, ranking signals, and acceptance rules behind these outcomes.

Failure patterns. We audit the incorrect localization records using their queries, candidate scores, filtering decisions, inspected code, and final outputs. The audit reveals a gap between finding relevant material and establishing what it means for the repair decision. Systems rely on surface matches, confuse related changes with actual repairs, or apply selection and acceptance rules that fail to reflect the available repair evidence. We group these evidence-interpretation and decision failures as F1–F8, distinguishing them from tool/runtime failures (O1) and data/mapping failures (O2). Figure 2 reports the distribution of these categories within each system's incorrect records. The dominant cues vary across systems, with Claude relying on security narratives and local code shapes, and GPT on temporal signals.

Observation 2. *Relevant evidence alone does not ensure a correct security decision. Systems must determine what it establishes for the problem and use that assessment to guide the final answer.*

3 Methodology

3.1 Overview

As shown in Figure 3, given a vulnerability description and the available textual or program evidence, SecFlow generates a structured answer through three connected components. Behavior grounding constructs a source-linked memory M encoding operations, conditions, effects, and their

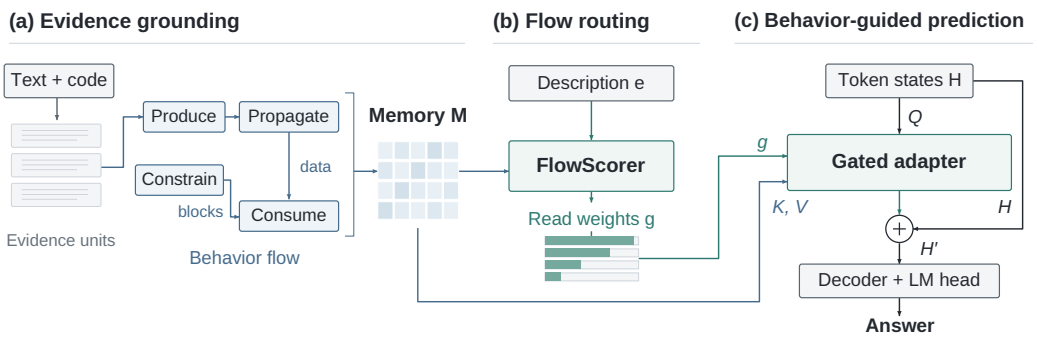


Figure 3: System overview of SecFlow.

relationships. Event-conditioned routing evaluates these representations against the vulnerability description and produces routing weights g, separating behavioral content from its problem-dependent reading priority. Behavior-guided prediction uses M as the content to be read and g as a prior over memory entries in gated cross-attention at later language-model layers. The updated token states pass through the remaining decoder layers and language-model head to generate the output.

3.2 Grounding Textual and Program Evidence

SecFlow constructs a source-linked behavior memory M from textual and program evidence. Evidence units preserve the source content, while behavior atoms and their relations represent its predicted security interpretation.

Evidence units and encoding. Textual artifacts are divided into claim-level units, while program artifacts are segmented into statements, call-condition pairs, or tightly coupled blocks. Constructed inputs and outcome assertions in regression tests form separate units. Each evidence unit u_i retains its original content and source location. Its representation z_i is obtained by pooling the corresponding input-span hidden states at a designated layer of the frozen language model. The source link provides a reference for subsequent behavioral interpretation without treating that interpretation as an observed fact.

Behavior interpretation. Each evidence representation z_i initializes a latent behavior atom a_i. A trainable tagger predicts its operation, security effect, and phase in the context of the objects and conditions expressed by the evidence. The shared phase vocabulary comprises PRODUCE, PROPAGATE, CONSUME, and CONSTRAIN, describing how values or states are introduced, transferred, used, or restricted. Using the same vocabulary for text and code makes their behavioral roles comparable. These attributes characterize individual behaviors, while relations between atoms describe how the behaviors interact.

Relations and flow memory. An edge scorer predicts typed relations between behavior atoms, covering data and state dependencies, control and branch relations, temporal order, and security effects. These relations express how values and states pass between operations, which conditions govern execution, and how one behavior constrains or affects another. A flow builder composes related atoms and edges into behavior flows, capturing interactions among operations, objects, and conditions rather than treating each operation independently.

Relational message passing uses the predicted relations to enrich atom representations with context from connected behaviors. The resulting memory $M \in \mathbb{R}^{N \times d_m}$ retains one entry per behavior atom, where N is the number of atoms and d_m is the memory dimension. Each entry combines its local evidence with relational context and preserves its supporting source links.

3.3 Event-Conditioned Flow Routing

Given the behavior memory M, SecFlow assesses how each entry relates to the security event under analysis and how it should be prioritized during prediction. The model-visible vulnerability description provides the event representation e.

Event-conditioned assessment. For each relation-enriched memory entry m_i, the FlowScorer combines m_i and e with their element-wise product and absolute difference. A shared feature network feeds separate heads predicting an event-relation distribution p_i^{rel} and a read-policy distribution p_i^{read}. The event relation characterizes whether the represented evidence supports or refutes the reported unsafe behavior, describes its mitigation, provides context, concerns another event, or is unrelated. Unlike the atom-to-atom relations in behavior grounding, these relations describe the role of the represented behavior in the current security problem.

Evidence prioritization. The read policy expresses which evidence should be emphasized when forming the answer. This priority is distinct from whether the evidence supports a particular conclusion. Evidence that challenges a conclusion can therefore receive high reading priority. We map the policy distribution to a continuous routing weight

$$g_i = \mathbf{w}^\top p_i^{\text{read}}, \qquad \mathbf{w} = (1, \tfrac{1}{2}, 0)^\top, \tag{1}$$

where the policy classes are ordered as MUST-READ, MAY-READ, and SUPPRESS, distinguishing evidence needed for the judgment, useful context, and irrelevant material. The fixed weights define a soft reading prior rather than a confidence score for evidence correctness. Event-relation and read-policy annotations supervise their respective predictions through L_{route}. The memory M, routing weights g, and a validity mask excluding unavailable entries are passed to the prediction module. There, g provides an event-conditioned prior over the behavior-memory entries.

3.4 BEHAVIOR-GUIDED PREDICTION

SecFlow integrates the behavior memory M and routing weights g into the language model through gated cross-attention at selected later decoder layers. The memory supplies behavior representations enriched with relational context, while the routing weights provide an event-conditioned prior over which entries to read.

Memory-guided generation. Let $H \in \mathbb{R}^{T \times d_h}$ denote the current token states and $M \in \mathbb{R}^{N \times d_m}$ the behavior memory. The adapter forms queries $Q = \text{Norm}(H)W_Q$, keys $K = MW_K$, and values $V = MW_V$, then computes

$$\begin{aligned} A &= \text{softmax}\left(\frac{QK^\top}{\sqrt{d}} + \beta \log(g + \epsilon) + B_\mu\right), \\ H' &= H + \text{LayerScale}\left((AV)W_O\right). \end{aligned} \tag{2}$$

Here, d is the key dimension, β controls the routing bias, $\epsilon > 0$ keeps the logarithm finite, and B_μ masks invalid memory entries. Softmax is applied over memory entries, with the routing bias broadcast across token positions. The content term measures compatibility with the current token state, while the routing term incorporates the reading priorities assigned for the security event. Their combination allows token-specific memory access without discarding the event-level evidence assessment. The residual update adds the retrieved behavior context to H.

The remaining decoder layers and language-model head use the updated states to generate an answer sequence $\hat{s}$ following the task-specific output schema. A task-specific parser $\mathcal{P}$ converts this sequence into the structured prediction $\hat{y} = \mathcal{P}(\hat{s})$. For vulnerability grounding, $\hat{y}$ contains the normalized slot–value objects extracted from the generated answer. For patch localization, it contains the final primary commit selection and the ordered candidate list specified in that answer. Task metrics are computed from these parsed predictions, rather than directly from internal routing weights or artifact-level scores.

Training objectives. SecFlow combines answer-generation supervision with auxiliary objectives for interpreting and using security evidence. L_{gen} supervises the output sequence; L_{ground} supervises behavior attributes, typed relations, and flow structure; L_{route} supervises event-relation and read-policy predictions; and L_{use} supervises the adapter's attention allocation using annotated read policies. Where candidate-preference annotations are available, L_{rank} additionally supervises the relative scores of candidate artifacts. The overall objective is

$$\begin{aligned} L = L_{\text{gen}} &+ \lambda_{\text{ground}} L_{\text{ground}} + \lambda_{\text{route}} L_{\text{route}} \\ &+ \lambda_{\text{rank}} L_{\text{rank}} + \lambda_{\text{use}} L_{\text{use}}. \end{aligned} \tag{3}$$

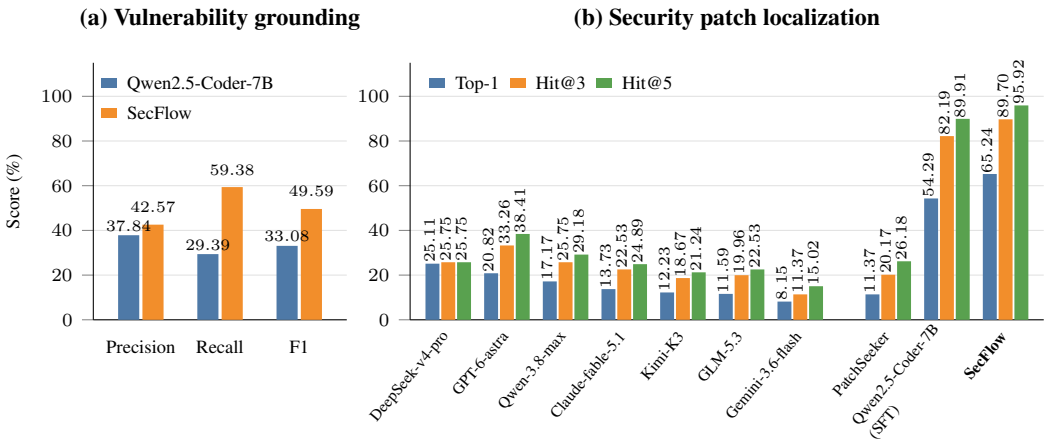


Figure 4: Performance on vulnerability grounding and security patch localization.

4 Evaluation

We evaluate whether SecFlow improves the recovery of security facts from reports and the use of repository evidence to identify security fixes. We further examine the contributions of its components and training objectives.

4.1 Experimental Setup

Evaluation datasets. Both tasks derive their training and development data from the same 15,365 OSV vulnerability instances used in the Preliminary Study. We augment the original vulnerability records and repository evidence with source-linked annotations of security roles, behavioral relations, and vulnerability–repair alignment. These annotations make explicit how the available facts and code changes support a security judgment, providing relational supervision beyond the original descriptions and fixing-commit labels. For vulnerability grounding, the vulnerability descriptions and their security-semantic annotations provide supervision for recovering security facts and their roles. We reserve a disjoint set of 1,000 vulnerability descriptions as the held-out evaluation set.

For patch localization, considering the fact that publicly available vulnerability–patch associations may have appeared in the pretraining data of the evaluated models, to reduce potential knowledge leakage from these associations, we construct a separate evaluation set from OSV records without explicit patch references. We manually inspect the corresponding repositories' commit histories and source diffs to identify the changes that repair the reported vulnerabilities. This process yields 466 vulnerability instances with manually verified fixing commits, reserved for final evaluation.

SecFlow implementation. To ensure a fair comparison, SecFlow follows the same evaluation environment as the Preliminary Study. SecFlow uses Qwen2.5-Coder-7B as the frozen backbone, while only the behavior-grounding, flow-scoring, ranking, and gated-adapter parameters are optimized. We train SecFlow with AdamW (Loshchilov & Hutter, 2019) on four NVIDIA A800 80GB GPUs for 3 epochs, using a learning rate of 2×10^{-4}, a global batch size of 32, and BF16 precision. For patch localization, we use model-hard negative sampling with two negative commits per positive fixing commit.

Baselines and Metrics. For vulnerability grounding, the evaluation data and protocol remain unchanged from the Preliminary Study. We therefore use the previously reported results of the seven commercial LLMs as reference and focus the additional evaluation on comparing SecFlow with its unadapted Qwen2.5-Coder-7B backbone. This same-backbone comparison measures the improvement introduced by SecFlow, as reported in Panel (a) of Figure 4.

For patch localization, we evaluate the same seven commercial LLMs on the new benchmark under the common agent protocol. We additionally include PatchSeeker (Nguyen et al., 2025) as a task-specific baseline. We also include a supervised fine-tuning (SFT) baseline based on Qwen2.5-Coder-

7B. It learns from training targets derived from the relational annotations described above, without SecFlow's dedicated behavior-memory, routing, or injection modules.

We follow the evaluation protocol of the Preliminary Study and additionally report Hit@3 to assess candidate coverage under a tighter shortlist cutoff. Together, Hit@3 and Hit@5 show how often an accepted fix remains within the highest-priority candidates as the shortlist size changes.

4.2 Security Facts Grounding Performance

Panel (a) of Figure 4 shows that SecFlow raises Recall from 29.39% to 59.38% and F1 from 33.08% to 49.59%, while Precision also increases from 37.84% to 42.57%. With the same vulnerability descriptions as input, SecFlow recovers more annotated security facts without sacrificing precision, addressing the omissions observed in the Preliminary Study.

4.3 Overall Patch Localization Performance

Panel (b) of Figure 4 reports patch-localization performance. SecFlow achieves 65.24% Top-1 accuracy and 95.92% Hit@5, exceeding the strongest commercial results by 40.13 and 57.51 percentage points, respectively.

PatchSeeker (Nguyen et al., 2025) matches vulnerability descriptions with commit messages augmented by code-diff summaries. Its 11.37% Top-1 accuracy and 26.18% Hit@5 illustrate the difficulty of establishing repair behavior through description-level similarity. A semantically related change need not prevent the unsafe behavior described in the report.

The same-backbone SFT baseline achieves 54.29% Top-1 accuracy and 89.91% Hit@5. Its targets explicitly encode security roles, behavioral relations, and vulnerability–repair alignment, and its performance supports the utility of this relational supervision. SecFlow further improves Top-1 and Hit@5 by 10.95 and 6.01 percentage points, respectively, supporting the benefit of explicit behavior modeling and evidence use beyond standard fine-tuning. The remaining gap between Top-1 and Hit@5 indicates that final selection remains challenging even when the shortlist contains an accepted fix.

Table 2: Ablation of SecFlow components and training objectives.

Variant	Top-1 ↑	Hit@3 ↑	Hit@5 ↑
SecFlow	**65.24%**	**89.70%**	**95.92%**
(a) Component ablations			
w/o Behavior Grounding	44.85%	78.11%	86.48%
w/o Event Routing	35.41%	65.88%	82.83%
w/o Behavior-Guided Prediction	39.06%	72.10%	86.05%
(b) Training-objective ablations			
w/o $\mathcal{L}_{\text{ground}}$	56.65%	82.62%	90.13%
w/o $\mathcal{L}_{\text{route}}$	46.14%	76.82%	86.05%
w/o $\mathcal{L}_{\text{rank}}$	28.97%	58.80%	74.03%
w/o $\mathcal{L}_{\text{use}}$	54.29%	80.90%	88.63%

4.4 Ablation Study

We examine SecFlow's architectural components and training objectives by removing one component or loss term at a time while keeping the remaining training and evaluation settings unchanged.

Contribution of Components. Table 2(a) shows that removing behavior grounding reduces Top-1 accuracy from 65.24% to 44.85%, supporting the value of explicitly representing security operations and their relationships. Removing event routing produces the largest component-level degradation, lowering Top-1 to 35.41%. This result highlights the importance of assessing grounded evidence against the vulnerability under analysis rather than representing behavior alone. Removing behavior-guided prediction lowers Top-1 to 39.06%, despite retaining grounding and routing, supporting the

additional contribution of the prediction component. All three removals also reduce Hit@3 and Hit@5, indicating that the degradation affects both final-answer correctness and shortlist coverage.

Contribution of Training objectives. Table 2(b) retains the full architecture. Removing L_{rank} causes the largest loss-ablation drop, to 28.97% Top-1, followed by L_{route} at 46.14%. Removing L_{ground} or L_{use} also degrades performance, supporting structural and attention supervision alongside candidate-preference and routing objectives.

5 Discussion

Broader applicability. Beyond vulnerability information extraction and patch localization, SecFlow also demonstrates utility in a vulnerability detection case. In a Piwigo case, GPT incorrectly treated allowlist rejection as blocking cookie propagation and `fatal_error` as terminating execution before HTML output, concluding that the path was unreachable. SecFlow's routing diagnostic ranked the call passing the raw cookie into `fatal_error` first, the cookie source and branch-entry condition second, and the concrete `echo $display` output third. These code locations reveal that the rejected input continues into an error message and reaches HTML output before termination. In the subsequent successful run, the model used this evidence to correctly identify the path as reachable. This case illustrates how prioritizing behaviorally decisive evidence can help correct a security judgment based on misleading surface cues.

Limitations. The current study is limited by the available computational budget and the capacity of the backbone model. Future work could explore larger, more capable backbones and increased training and inference budgets to improve SecFlow's performance. Another direction is agent reinforcement learning for multi-step tool use and evidence gathering, with the goal of extending SecFlow to more complex security workflows such as penetration testing.

6 Related work

Vulnerability information extraction and semantic modeling. Prior work transforms vulnerability reports into structured security knowledge through entity extraction (Dong et al., 2019), semantic aspect identification (Guo et al., 2021; 2022; Yitagesu et al., 2021; 2023), and knowledge graphs (Cheng et al., 2022). More recent approaches connect report statements to code (Gao et al., 2025), retrieve patch-derived security specifications (Zhu et al., 2026), or construct patch-behavior dependency graphs (Chen et al., 2026).

Security patch detection and localization. Security patch detection identifies security-related commits using learned code-change representations (Zhou et al., 2021; 2023) and patch graphs (Wang et al., 2023). Patch localization instead ranks commits for a given vulnerability report. Existing approaches use advisory–commit features (Tan et al., 2021; Wang et al., 2022), retrieval and reranking (Li et al., 2024), version constraints (Dunlap et al., 2024), or generated commit summaries (Nguyen et al., 2025).

Security-oriented LLMs and repository agents. Security-oriented models adapt LLMs through preference learning, continual pretraining, and instruction tuning, as illustrated by ProSec (Xu et al., 2025) and RedSage (Suryanto et al., 2026). Repository agents such as RepoAudit combine code exploration with validation of model-generated data-flow facts (Guo et al., 2025), while benchmarks including CyBench and SECODEPLT evaluate cybersecurity capabilities (Zhang et al., 2025; Nie et al., 2026).

7 Conclusion

In this work, we presented SecFlow, a framework that connects source-linked behavior modeling and event-conditioned evidence routing with answer generation. Experiments show that SecFlow improves grounding F1 from 33.08% to 49.59% and achieves 65.24% Top-1 accuracy in patch localization. These results highlight the value of explicitly modeling and using security evidence beyond surface relevance.

AI USE STATEMENT

In this work, we used generative AI tools for polishing grammar and writing. We have reviewed all AI-assisted material and take full responsibility for the accuracy, originality, and integrity of the final manuscript.

REPRODUCIBILITY STATEMENT

Code is available at `https://anonymous.4open.science/r/SecFlow/`.

REFERENCES

Anthropic. Making frontier cybersecurity capabilities available to defenders. `https://www.anthropic.com/news/claude-code-security`, 2026. February 20, 2026. Accessed: 2026-09-24.

Bofei Chen, Shuang Liao, Lei Zhang, Chibin Zhang, Mathias Payer, and Yuan Zhang. Patch-guided vulnerability detection: Extracting Java API security rules via attack–defense cross-analysis. In *35th USENIX Security Symposium (USENIX Security 26)*, Baltimore, MD, August 2026. USENIX Association.

Xingqi Cheng, Xiaobing Sun, Lili Bo, and Ying Wei. Kvs: a tool for knowledge-driven vulnerability searching. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1731–1735, 2022.

Ying Dong, Wenbo Guo, Yueqi Chen, Xinyu Xing, Yuqing Zhang, and Gang Wang. Towards the detection of inconsistencies in public security vulnerability reports. In *28th USENIX security symposium (USENIX Security 19)*, pp. 869–885, 2019.

Trevor Dunlap, Elizabeth Lin, William Enck, and Bradley Reaves. Vfcfinder: Pairing security advisories and patches. In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, pp. 1128–1142, 2024.

Amiao Gao, Zenong Zhang, Simin Wang, Liguo Huang, Shiyi Wei, and Vincent Ng. Teaching ai the 'why'and 'how'of software vulnerability fixes. *Proceedings of the ACM on Software Engineering*, 2(FSE):2006–2029, 2025.

GitHub Advisory Database. Cloud Foundry Archiver vulnerable to path traversal. `https://github.com/advisories/GHSA-32qh-8vg6-9g43`, 2022.

Hao Guo, Zhenchang Xing, Sen Chen, Xiaohong Li, Yude Bai, and Hu Zhang. Key aspects augmentation of vulnerability description based on multiple security databases. In *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*, pp. 1020–1025. IEEE, 2021.

Hao Guo, Sen Chen, Zhenchang Xing, Xiaohong Li, Yude Bai, and Jiamou Sun. Detecting and augmenting missing key aspects in vulnerability descriptions. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 31(3):1–27, 2022.

Jinyao Guo, Chengpeng Wang, Xiangzhe Xu, Zian Su, and Xiangyu Zhang. RepoAudit: An autonomous LLM-agent for repository-level code auditing. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 21083–21100. PMLR, 2025.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, Yunlong Feng, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. Qwen2.5-Coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

Stephane Jolicoeur. Refactored the path resolution to use secure-join. `https://github.com/cloudfoundry/archiver/commit/09b5706aa9367972c09144a450bb4523049ee840`, 2018.

Kaixuan Li, Jian Zhang, Sen Chen, Han Liu, Yang Liu, and Yixiang Chen. Patchfinder: A two-phase approach to security patch tracing for disclosed vulnerabilities in open-source software. In *Proceedings of the 33rd ACM SIGSOFT international symposium on software testing and analysis*, pp. 590–602, 2024.

Yansong Li, Paula Branco, Alexander M Hoole, Manish Marwah, Hari Manassery Koduvely, Guy-Vincent Jourdan, and Stephan Jou. Sv-trusteval-c: Evaluating structure and semantic reasoning in large language models for source code vulnerability analysis. In *2025 IEEE symposium on security and privacy (SP)*, pp. 3014–3032. IEEE, 2025.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Huu Hung Nguyen, Anh Tuan Nguyen, Thanh Le-Cong, Yikun Li, Han Wei Ang, Yide Yin, Frank Liauw, Shar Lwin Khin, Ouh Eng Lieh, Ting Zhang, et al. Patchseeker: Mapping nvd records to their vulnerability-fixing commits with llm generated commits and embeddings. *arXiv preprint arXiv:2509.07540*, 2025.

Yuzhou Nie, Zhun Wang, Yu Yang, Ruizhe Jiang, Yuheng Tang, Xander Davies, Yarin Gal, Bo Li, Wenbo Guo, and Dawn Song. Secodeplt: A unified benchmark for evaluating the security risks and capabilities of code genai. *Advances in Neural Information Processing Systems*, 38, 2026.

Nous Research. Hermes Agent, n.d. Software repository. Accessed: 2026-09-24.

OpenAI. Trusted access for the next era of cyber defense. `https://openai.com/index/scaling-trusted-access-for-cyber-defense/`, 2026. April 14, 2026. Accessed: 2026-09-24.

OSV. OSV: Open source vulnerabilities. `https://osv.dev/`, n.d. Continuously updated vulnerability database. Accessed: 2026-09-24.

Naufal Suryanto, Muzammal Naseer, Pengfei Li, Syed Talal Wasim, Jinhui Yi, Juergen Gall, Paolo Ceravolo, and Ernesto Damiani. RedSage: A cybersecurity generalist LLM. In *The Fourteenth International Conference on Learning Representations*, 2026.

Xin Tan, Yuan Zhang, Chenyuan Mi, Jiajun Cao, Kun Sun, Yifan Lin, and Min Yang. Locating the security patches for disclosed oss vulnerabilities with vulnerability-commit correlation ranking. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pp. 3282–3299, 2021.

Saad Ullah, Mingji Han, Saurabh Pujar, Hammond Pearce, Ayse Coskun, and Gianluca Stringhini. Llms cannot reliably identify and reason about security vulnerabilities (yet?): A comprehensive evaluation, framework, and benchmarks. In *2024 IEEE symposium on security and privacy (SP)*, pp. 862–880. IEEE, 2024.

Jialai Wang, Juncheng Lu, Songtao Yang, Bingyi He, Zeyu Gao, Chao Zhang, and Shuai Wang. Are reasoning traces faithful in LLM-based code vulnerability detection? In *Proceedings of the 41st IEEE/ACM International Conference on Automated Software Engineering*, 2026a. Accepted to the research track.

Shichao Wang, Yun Zhang, Liagfeng Bao, Xin Xia, and Minghui Wu. Vcmatch: a ranking-based approach for automatic security patches localization for oss vulnerabilities. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 589–600. IEEE, 2022.

Shu Wang, Xinda Wang, Kun Sun, Sushil Jajodia, Haining Wang, and Qi Li. Graphspd: Graph-based security patch detection with enriched code semantics. In *2023 IEEE Symposium on Security and Privacy (SP)*, pp. 2409–2426. IEEE, 2023.

Zhun Wang, Tianneng Shi, Jingxuan He, Matthew Cai, Jialin Zhang, and Dawn Song. CyberGym: Evaluating AI agents' real-world cybersecurity capabilities at scale. In *The Fourteenth International Conference on Learning Representations*, 2026b.

Xiangzhe Xu, Zian Su, Jinyao Guo, Kaiyuan Zhang, Zhenting Wang, and Xiangyu Zhang. ProSec: Fortifying code LLMs with proactive security alignment. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 69689–69704. PMLR, 2025.

Sofonias Yitagesu, Zhenchang Xing, Xiaowang Zhang, Zhiyong Feng, Xiaohong Li, and Linyi Han. Unsupervised labeling and extraction of phrase-based concepts in vulnerability descriptions. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 943–954. IEEE, 2021.

Sofonias Yitagesu, Zhenchang Xing, Xiaowang Zhang, Zhiyong Feng, Xiaohong Li, and Linyi Han. Extraction of phrase-based concepts in vulnerability descriptions through unsupervised labeling. *ACM Transactions on Software Engineering and Methodology*, 32(5):1–45, 2023.

Andy K. Zhang, Neil Perry, Riya Dulepet, Joey Ji, Celeste Menders, Justin W. Lin, Eliot Jones, Gashon Hussein, Samantha Liu, Donovan Julian Jasper, Pura Peetathawatchai, Ari Glenn, Vikram Sivashankar, Daniel Zamoshchin, Leo Glikbarg, Derek Askaryar, Haoxiang Yang, Aolin Zhang, Rishi Alluri, Nathan Tran, Rinnara Sangpisit, Kenny O. Oseleononmen, Dan Boneh, Daniel E. Ho, and Percy Liang. CyBench: A framework for evaluating cybersecurity capabilities and risks of language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

Jiayuan Zhou, Michael Pacheco, Zhiyuan Wan, Xin Xia, David Lo, Yuan Wang, and Ahmed E Hassan. Finding a needle in a haystack: Automated mining of silent vulnerability fixes. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 705–716. IEEE, 2021.

Jiayuan Zhou, Michael Pacheco, Jinfu Chen, Xing Hu, Xin Xia, David Lo, and Ahmed E Hassan. Colefunda: Explainable silent vulnerability fix identification. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pp. 2565–2577. IEEE, 2023.

Hao Zhu, Jia Li, Cuiyun Gao, Jiaru Qian, Yihong Dong, Huanyu Liu, Lecheng Wang, Ziliang Wang, Xiaolong Hu, and Ge Li. VulInstruct: Teaching LLMs root-cause reasoning for vulnerability detection via security specifications. *Proceedings of the ACM on Software Engineering*, 3(FSE): 2813–2834, 2026.