

# SemanticVul: Evidence-Grounded Structured LLM Explanations for Vulnerability Detection

**Abdul Razak**

National University of Sciences and Technology

**Ihtesham Ul Islam**

National University of Sciences and Technology

**Rabia Khan**

`rabia.khan@mcs.nust.edu.pk`

National University of Sciences and Technology

---

## Research Article

**Keywords:** vulnerability detection, large language models, explanation grounding, adaptive fusion, code semantics, class imbalance

**Posted Date:** October 15th, 2026

**DOI:** <https://doi.org/10.21203/rs.3.rs-10876072/v1>

**License:**   This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

**Additional Declarations:** No competing interests reported.

---

# SemanticVul: Evidence-Grounded Structured LLM Explanations for Vulnerability Detection

Abdul Razak<sup>1</sup>, Ihtesham Ul Islam<sup>1</sup>, Rabia Khan<sup>1\*</sup>

<sup>1</sup>Department of Computer Software Engineering, Military College of Signals, National University of Sciences and Technology (NUST), Islamabad, Pakistan.

\*Corresponding author(s). E-mail(s): [rabia.khan@mcs.nust.edu.pk](mailto:rabia.khan@mcs.nust.edu.pk);  
Contributing authors: [arazak.ms24mc@student.nust.edu.pk](mailto:arazak.ms24mc@student.nust.edu.pk);  
[ihtesham@mcs.edu.pk](mailto:ihtesham@mcs.edu.pk);

## Abstract

Large language models (LLMs) offer a promising source of semantic information for vulnerability detection, yet their use in security-critical settings is hindered by costly proprietary inference, limited reproducibility, and explanations that may contain unsupported or label-leaking claims. We present SemanticVul, a function-level C/C++ vulnerability detection framework that integrates code semantics with a locally generated, evidence-grounded explanation channel while explicitly separating explanation utility from adaptive multimodal fusion. SemanticVul replaces proprietary free-form explanations with an offline Qwen2.5-Coder-based generator that produces schema-constrained JSON descriptions of program purpose, data flow, risky operations, missing checks, and code evidence. A deterministic validation pipeline removes unsupported claims through whitespace-insensitive code matching and scrubs verdict-related vocabulary before explanations are encoded by RoBERTa. To systematically quantify the contribution of the explanation modality, we introduce a three-level fusion ladder comprising a code-only baseline, static attention fusion, and an adaptive per-sample gating mechanism. Experiments on the released Devign and Reveal validation splits show substantial gains over the corresponding published FuSE-Vul reference results under the base-comparable protocol, with improvements of 8.60 percentage points in accuracy and 7.31 points in F1 on Devign, and 1.23 and 15.09 points, respectively, on Reveal. On Reveal, minority-class recall increases from 39.52% to 63.00%, highlighting the potential of grounded semantic information under severe class imbalance. The adaptive gate achieves the highest F1 under the base-comparable setting; however, its advantage is not statistically resolved and does not remain consistent under held-out evaluation,

indicating that adaptive modality weighting should not yet be interpreted as a definitive contribution. A dedicated grounding audit increases the proportion of grounded API mentions from 26.2% to 55.4% while reducing hallucinated API references, and a dataset audit exposes duplicate and cross-split label contradictions in both benchmarks. Overall, the results demonstrate that locally generated, evidence-grounded explanations can provide a reproducible semantic signal for vulnerability detection, while also showing the importance of rigorous split auditing and held-out evaluation when assessing explanation-driven improvements.

**Keywords:** vulnerability detection, large language models, explanation grounding, adaptive fusion, code semantics, class imbalance

## 1 Introduction

Many exploited software vulnerabilities originate in C/C++ source code, and the volume of newly disclosed weaknesses exceeds what manual review can handle [1, 2]. Learning-based detectors that classify functions as vulnerable or benign automate part of that review. The field has progressed from supervised graph models [3, 4] to fine-tuned pre-trained code models [5, 6] and detectors that use large language models (LLMs) [7]. One approach keeps a trained classifier responsible for the verdict but enriches its input with an LLM-generated natural-language explanation of what the code does and where it may be risky. FuSEVul [8] applies this design by fusing CodeT5+ code features with ChatGPT-generated risk descriptions. It reports the strongest results among eighteen baselines on the Devign and Reveal benchmarks.

The explanation half of that recipe has two weaknesses. The explanations are free-form prose from a commercial API, so they are costly at corpus scale and can vary across accesses. They are also not audited: an explanation may hallucinate API calls that do not exist in the function or include verdict-like vocabulary that leaks the label into the input. Recent studies report both failure modes, reporting correct verdicts routinely paired with flawed reasoning [9, 10] and finding that fewer than six percent of surveyed works evaluate textual vulnerability explanations at all [11]. In parallel, fusion itself is usually a single fixed mechanism, which makes it impossible to tell whether reported gains come from the explanation content or from the extra parameters of the fusion block.

**SemanticVul** addresses these gaps in two ways. The first is an *evidence-grounded structured explanation pipeline*: a locally served, open-weight code LLM (Qwen2.5-Coder) emits a schema-constrained JSON explanation whose security claims must be linked to substrings of the analyzed function under whitespace-insensitive matching; a deterministic validator drops ungrounded claims, scrubs verdict vocabulary, and replaces self-reported confidence with a decode-time probability probe. The second is a *controlled fusion ladder with an adaptive-gate variant*. SemanticVul trains three configurations under matched conditions: code-only (L1), static attention fusion (L2), and a per-sample adaptive gate that learns how much to weight the two representation

streams (L3). This makes the explanation channel and the gate separately measurable; the weight is a diagnostic of feature mixing, not an explanation of an individual prediction.

This paper makes four contributions:

1. **A local, label-blind, evidence-grounded explanation pipeline** replacing FuSEVul’s commercial free-form prose: on matched Reveal functions it more than doubles API-mention grounding (26.2% to 55.4%), halves hallucinated call mentions, and provides explicit evidence anchors for 87% of functions.
2. **A controlled L1–L3 fusion ladder with a per-sample adaptive gate**, which makes the explanation channel and the fusion mechanism separately measurable. The explanation channel provides the largest and most durable gain over code-only, up to +17.5 F1 under both protocols. The gate reaches the highest base-comparable F1 on both datasets, but the five-seed comparison does not establish a reliable improvement over static fusion.
3. **Performance relative to the reported FuSEVul reference on the released validation splits**: +8.60 / +7.31 on Devign and +1.23 / +15.09 on Reveal, with minority-class recall rising from 39.52 to 63.00, using one RTX 3090 GPU and 12 training epochs. This is a reported-reference comparison rather than a local controlled rerun.
4. **A public data-quality audit of the distributed splits**, quantifying label-conflicting duplicates, cross-split leakage, and the conditional memorization bounds they imply, with a two-protocol design separating base-comparable from non-circular claims.

The rest of the paper reviews related work (Section 2), describes the methods (Section 3) and experimental setup (Section 4), reports the four research questions (Section 5), and discusses limitations and future work (Section 6) before concluding in Section 7.

## 2 Related work

### 2.1 Supervised-based methods

Early learning-based detectors encode code as token sequences or graphs that preserve syntax, control flow, and data dependence. Devign combines these views in a joint graph processed by a gated graph neural network [3]; Reveal builds on code property graphs and focuses on the rare-positive regime, with under ten percent of its functions vulnerable [4]. Later systems refine the structural signal, including control- and data-dependence-weighted attention that needs no natural language at all [12]. A parallel line questions what benchmark scores mean: PrimeVul documents noisy labels, duplication, and train–test leakage. A 7B model drops from 68.26 F1 on BigVul to 3.09 under stricter evaluation [13]; Real-Vul reports precision drops of up to 95 points in whole-codebase settings [14]; and Risse et al. [15] find 45.7% inaccurate labels in a widely used benchmark. These findings directly motivate the split audit of Section 4.2.

## 2.2 Pre-trained model-based methods

Pre-trained code models replaced task-specific representation learning: CodeBERT learns from paired code and natural language [5], GraphCodeBERT adds data flow during pre-training [16], UniXcoder unifies encoder, decoder, and cross-modal objectives [6], and CodeT5+ provides the encoder used by FuSEVul [17]. FuSEVul’s own encoder ablation shows that swapping among these models moves Reveal F1 by little more than a point [8]. This suggests that the main design space is not the code encoder. SemanticVul therefore keeps CodeT5+ for parity and directs its changes at the explanation source and the fusion mechanism.

## 2.3 LLM-based methods

LLMs enter vulnerability detection in two roles. As *detectors*, they are prompted or fine-tuned to output the verdict: chain-of-thought over threat slices [18], reasoning structured as a directed acyclic graph [10], retrieved in-context examples with graph structure [19], graph-guided slice selection [20], detector-and-static-analysis evidence embedded in prompts [21], and instruction tuning toward joint verdict-plus-explanation output [22]. Direct prompting remains mixed: across sixteen models, mean accuracy is 62.8% with better local reasoning than broad-context reasoning [23]. As *auxiliary modules*, LLMs supply semantic signals to a trained classifier: collaborative GNN-LLM training [24], summaries of called functions fed to an existing detector [25], refined vulnerability descriptions [26], and, closest to this work, FuSEVul’s fusion of code features with free-form ChatGPT explanations [8].

Table 1 compares these systems along the design axes used for explanation fusion, its final column recording whether each explanation’s grounding is checked before use. To our knowledge, none of the compared systems checks whether each explanation is grounded in the function before using it. Detector-role systems couple the verdict to the reasoning, so a hallucinated explanation silently corrupts the decision; studies that inspect reasoning quality find correct verdicts resting on flawed explanations [9, 10], and a review of 208 studies found only 5.28% evaluate textual explanations at all [11]. Auxiliary-role systems, including FuSEVul, fuse the LLM signal through one fixed mechanism that treats a grounded explanation and a hallucinated one as equally informative. SemanticVul addresses both gaps: the LLM produces only the explanation, while the verdict stays with a trained classifier; a validator enforces evidence grounding and verdict scrubbing before the text is encoded; automatic checks audit faithfulness; and the controlled ladder with a per-sample gate separates the explanation’s contribution from the fusion mechanism. The generator is open-weight and runs offline, unlike the commercial API or verdict-coupled designs used by the comparison systems.

**Table 1** Technique comparison on the design axes relevant to explanation fusion.

| System                    | Role                     | Explanation / context                | Fusion                   | Audit      |
|---------------------------|--------------------------|--------------------------------------|--------------------------|------------|
| FuSEVul [8]               | auxiliary explainer      | free-form Chat-GPT prose             | static atten-<br>tion    | no         |
| GPTVD [18]                | detector                 | CoT threat<br>slices                 | prompt-<br>internal      | no         |
| GRACE [19]                | detector                 | graph + ICL<br>examples              | prompt-<br>internal      | no         |
| LLMVulExp [22]            | detector +<br>explainer  | joint instruction-<br>tuned output   | LLM verdict              | no         |
| DLAP [21]                 | detector                 | DL + static-<br>analysis<br>evidence | prompt-<br>internal      | no         |
| LLMxCPG [20]              | detector                 | CPG slices                           | prompt-<br>internal      | no         |
| VulnSC [25]               | auxiliary sum-<br>maries | called-function<br>summaries         | existing<br>detector     | no         |
| CTVD [24]                 | semantic<br>module       | LLM semantic<br>features             | collaborative<br>GNN-LLM | no         |
| CosFormer [12]            | code only                | none                                 | CFG/PDG<br>attention     | n/a        |
| <b>SemanticVul (ours)</b> | auxiliary explainer      | grounded<br>structured<br>JSON       | L1-L3 + gate             | <b>yes</b> |

## 3 Method

### 3.1 Overview and design rationale

SemanticVul is a function-level binary vulnerability detector for C/C++ that fuses code semantic features with an LLM-generated explanation of the same function, following FuSEVul’s central idea. It changes the two parts that are hardest to trust and reproduce. The commercial free-form explanation source becomes a local, offline generator whose schema-constrained JSON output is grounding-validated and verdict-scrubbed before it reaches the detector (Section 3.3); the single fixed fusion block becomes a controlled three-rung ladder trained under matched seeds and hyperparameters, so the explanation channel and the adaptive gate are separately measurable (Section 3.4). Figure 1 shows the end-to-end architecture.

Three goals govern the explanation design: *consistency* (a fixed structure for every function), *grounding* (evidence-linked security claims must rest on whitespace-normalized code substrings; unsupported items are dropped), and *leakage safety* (verdict words are banned from the fed text). A fourth constraint is deployability:

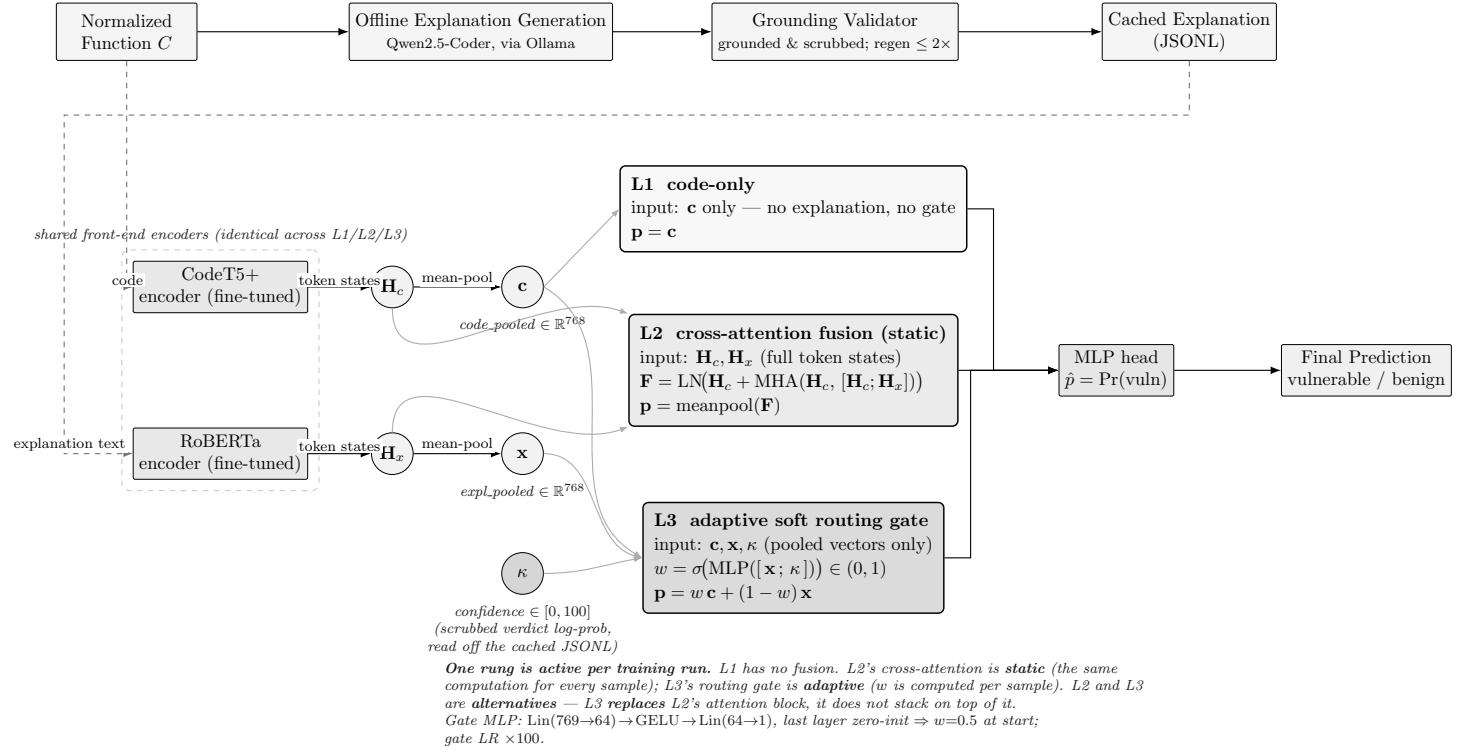


**Fig. 1** SemanticVul architecture. CodeT5+ encodes a normalized C/C++ function; RoBERTa encodes a locally generated, verdict-scrubbed, evidence-grounded JSON explanation of the same function; the L1–L3 ladder measures the code-only baseline, static attention fusion, and per-sample adaptive gated fusion under matched training conditions.

every component runs on a single consumer GPU with no external API access, keeping the pipeline reproducible and proprietary code on premises.

### 3.2 Code semantic encoding

The detector reads the `raw_code` representation distributed with the FuSEVul split files without recovering identifiers or adding external code features. Design is supplied in a FUN/VAR-normalized form; Reveal retains identifiers present in its distributed representation. CodeT5+ [17] (`codet5p-110m-embedding`), the base paper’s default and best-performing code encoder, encodes the function  $C$  with a 512-token window, yielding token states  $\mathbf{H}_c \in \mathbb{R}^{L_c \times d}$  with  $d=768$ .

### 3.3 Structured explanation generation and encoding

**Local generation.** Explanations are generated offline by Qwen2.5-Coder 14B [27] served through Ollama [28] with an 8,192-token context window. Ground-truth labels are never included in the prompt. Generation is a one-time corpus-level pass whose outputs are cached, so detector training never waits on the LLM.

**Schema-constrained output.** Instead of free-form prose, the model must emit a single JSON object with fixed fields: `purpose`, `data_flow`, `risky_operations`, `missing_checks`, `evidence_tokens`, `safety_indicators`, `risk_summary`, and `risk_level`. The pipeline then appends the measured `confidence` field. The `risk_level` field is ordered last in the schema so its value tokens are emitted after the evidence that should support them. The `confidence` scalar  $\kappa \in [0, 100]$  is *measured*, not self-reported: generation runs with decode-time log-probabilities enabled, and a probe sets  $\kappa$  to 100 times the geometric-mean token probability of the generated `risk_level` value tokens (falling back to the whole generation when that span cannot be located).  $\kappa$  thus reflects the generator’s internal certainty about its risk assessment, and it is the one channel through which that assessment reaches the detector at all, in compressed scalar form. Measuring rather than asking is deliberate: decode-time token probabilities track a model’s actual uncertainty better than verbalized self-reports [29], and because the detector consumes  $\kappa$  as a learned input feature rather than as a probability estimate, only its ranking information matters and no calibration guarantee [30] is required. The probe touches nothing outside the generation itself, so  $\kappa$  inherits the pipeline’s label-blindness by construction.

**Grounding validation and verdict scrubbing.** A deterministic validator enforces the grounding contract. A risky-operation claim must carry a matching code-evidence span; evidence tokens and safety indicators without matching code are dropped. Because the schema does not attach a separate span to each missing-check item, those items are retained only when grounded anchors survive and the claim names an identifier present in the function. Unsupported items are dropped, and when most evidence is invented the explanation is regenerated once. The validator also removes verdict vocabulary (*vulnerable*, *safe*, *exploit*, *attacker*, and similar) so the fed text carries description, not judgment. The categorical `risk_level` verdict never reaches the detector; the measured  $\kappa$  is its only surviving trace. The pipeline serializes  $\kappa$  into the restricted explanation text read by every fusion rung, and at L3 it also feeds

the gate network. It is not ground-truth leakage because the generator never sees the dataset label, but it is an auxiliary LLM risk signal rather than descriptive explanation content. Section 5.2 quantifies how strongly it correlates with the label and scopes the resulting claims. Figure 2 contrasts this pipeline with FuSEVul’s free-form generation.

**Encoding.** The restricted explanation text (the measured confidence  $\kappa$ , purpose, data flow, risky operations, missing checks, grounded evidence tokens, safety indicators, and risk summary) is serialized and encoded by RoBERTa [31] (`roberta-base`) with a 512-token window, yielding  $\mathbf{H}_x \in \mathbb{R}^{L_x \times d}$ .

### 3.4 Fusion: the L1–L3 ladder and adaptive gate

Masked mean-pooling gives the pooled views  $\mathbf{c} = \text{pool}(\mathbf{H}_c)$  and  $\mathbf{x} = \text{pool}(\mathbf{H}_x)$ , and a shared two-layer MLP head maps a pooled vector to  $\hat{p} = \Pr(\text{vulnerable} \mid C, X)$ . The three rungs differ only in what reaches that head (Figure 1):

**L1 (code-only).** The head reads  $\mathbf{c}$  alone. This is the within-system baseline; the explanation is deliberately ignored.

**L2 (static attention fusion).** Code tokens query the concatenation of code and explanation tokens through multi-head attention with a residual connection,

$$\mathbf{F} = \text{LayerNorm}(\mathbf{H}_c + \text{MHA}(\mathbf{Q}=\mathbf{H}_c, \mathbf{K}=\mathbf{V}=[\mathbf{H}_c; \mathbf{H}_x])), \quad \hat{p}_{L2} = \text{head}(\text{pool}(\mathbf{F})). \quad (1)$$

The mechanism is FuSEVul-style and identical for every sample; the L2–L1 difference therefore measures the utility of the explanation channel itself.

**L3 (adaptive fusion gate).** L3 replaces the attention block with a learned per-sample convex combination of the two pooled views. A small gate network reads what the explanation says ( $\mathbf{x}$ ) and how certain the generator was ( $\kappa$ ):

$$w = \sigma(\mathbf{W}_2 \phi(\mathbf{W}_1[\mathbf{x}; \kappa] + \mathbf{b}_1) + b_2) \in (0, 1), \quad \hat{p}_{L3} = \text{head}(w \mathbf{c} + (1 - w) \mathbf{x}), \quad (2)$$

with  $\mathbf{W}_1 \in \mathbb{R}^{64 \times (d+1)}$ ,  $\mathbf{W}_2 \in \mathbb{R}^{1 \times 64}$ , and  $\phi = \text{GELU}$ . The output layer is zero-initialized (training starts from an unbiased 50/50 blend) and the gate’s learning rate is scaled by 100 so it moves while the encoders fine-tune. As a single scalar per sample,  $w$  performs soft mixing between views. Since the pooled CodeT5+ and RoBERTa vectors come from different pretrained spaces, this is learned feature mixing rather than proof of modality routing or that the two representations are directly semantically aligned.

Twenty-two label-free quality and risk features (explanation content counts, reasoning flags, cross-modal similarity, code risk-surface counts) are computed for every sample but are *not* predictive inputs to the final ladder: a frozen-cache mechanism study (Section 5.3) showed quality-conditioned gating does not reliably beat simpler fusion. The features are used instead for the grounding, leakage, and faithfulness audits (Section 5.2).

### 3.5 Imbalance-aware training, thresholding, and ensembling

Reveal is heavily imbalanced (9.12% vulnerable in the evaluation split), so its runs use focal loss [32] with capped positive weighting ( $\alpha = 0.85$ ,  $\gamma = 2.0$ ); near-balanced

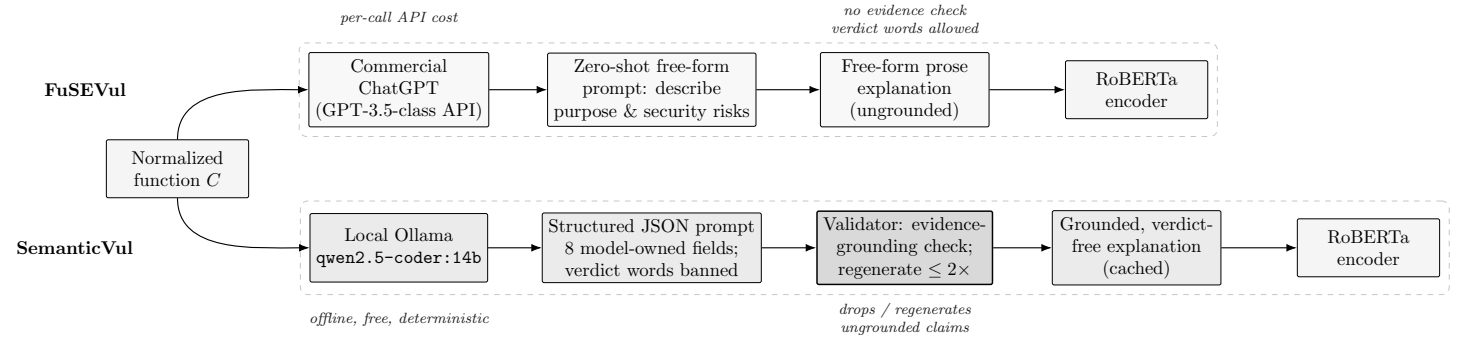


**Fig. 2** Free-form commercial explanation generation in FuSEVul (top) versus SemanticVul’s local, schema-constrained, evidence-grounded pipeline (bottom). The validator drops ungrounded claims, scrubs verdict vocabulary, and replaces self-reported confidence with a decode-time probability probe.

Devign uses plain cross-entropy. All headline results follow the FuSEVul-faithful operating point: the best-validation-accuracy epoch with a natural threshold of 0.5. Threshold tuning is reported only as a diagnostic, with the tuned threshold  $\tau^* = \arg \max_{\tau} \text{F1}(\tau)$  selected on a stratified 12% tune split carved from training data and never used for gradient updates. Finally, a multi-seed ensemble combines the five seeds’ predictions by weighted rank averaging,  $p_i^{\text{ens}} = \sum_k w_k r_k(p_i^{(k)})$  with  $w_k \geq 0$ ,  $\sum_k w_k = 1$ , where the weights are found by rank search over single members, dense pairs, and random mixtures of the strongest members. The search selects these weights on the same validation split on which the ensemble is then scored, so the ensemble rows are reported throughout as an optimistic diagnostic of what multi-seed combination can add, on a par with the validation-tuned threshold, and never as a deployment estimate.

## 4 Experimental setup

### 4.1 Research questions

**RQ1 (Overall):** How does SemanticVul compare with FuSEVul and representative baselines on Devign and Reveal? **RQ2 (Explanation technique):** To what extent do structured, evidence-grounded, label-blind local explanations improve detection utility over a code-only detector, and explanation grounding over the free-form LLM explanations distributed with the base system? **RQ3 (Fusion technique):** How does adaptive gated fusion compare with static attention fusion and code-only models? **RQ4 (Operating points):** How sensitive is minority-class detection to threshold selection and multi-seed ensembling under the fixed training recipe?

### 4.2 Datasets and data-quality audit

Experiments use two real-world datasets distributed with the base system: Devign [3], functions from Linux, QEMU, Wireshark, and FFmpeg, and Reveal [4], functions collected from Chrome and Debian issue trackers. Devign is near-balanced (43.45% vulnerable in the evaluation split) while Reveal is heavily imbalanced (9.12%), so the pair exercises both balanced and rare-positive regimes, the two settings probed by the imbalance-aware training and operating-point analysis in RQ4. The third base-paper corpus, TrVD, is synthetic: its vulnerabilities are template-injected, so its defect patterns are lexically stereotyped, nearly all base-paper systems already score close to or above 80 F1 on it, and a grounding audit is uninformative when the evidence to be grounded is templated by construction. The findings therefore do not establish generalization to synthetic benchmarks.

Because both systems train and evaluate on the distributed splits, we first audited them (Table 2, lower panel), where the reported split sizes are those of the distributed FuSEVul release and the audit hashes each normalized function body. Both datasets ship label-conflicting duplicates within splits, and functions that appear in both training and validation with reversed labels. Under the conditional assumption that a deterministic content-only model reproduces a leaked training label, these contradictions imply a train-label-memorization consistency bound (the table’s final row) of

**Table 2** Datasets and data-quality audit.

|                                                  | Devign           | Reveal            |
|--------------------------------------------------|------------------|-------------------|
| Training functions (released)                    | 21,854           | 18,187            |
| Validation functions (evaluation split)          | 2,732            | 2,269             |
| vulnerable / non-vulnerable                      | 1,187 / 1,545    | 207 / 2,062       |
| vulnerable share                                 | 43.45%           | 9.12%             |
| <i>Audit (SHA-1 of normalized function body)</i> |                  |                   |
| Duplicate codes with conflicting labels (train)  | 124              | 426               |
| Duplicate codes with conflicting labels (val)    | 12               | 8                 |
| Cross-split leaked codes with reversed labels    | 27               | 69                |
| Train-label memorization consistency bound       | $\approx 99.0\%$ | $\approx 96.96\%$ |

about 99.0% on Devign and 96.96% on Reveal; these are conditional bounds, not universal ceilings on validation accuracy. The audit still motivates reporting F1 and a cleaned secondary analysis.

### 4.3 Baselines

The primary baseline is FuSEVul as reported [8], with its default CodeT5+ encoder. It is the main comparison point because the paper reports it on the same distributed benchmarks. Within-system baselines complete the comparison: the L1 code-only rung isolates what the explanation adds, and a frozen-cache study compares static concatenation, gated fusion, and classical classifiers over identical cached features.

### 4.4 Evaluation metrics and protocols

Following the base paper, we report Accuracy, Precision, Recall, and F1 on the vulnerable class at threshold 0.5, adding threshold-free ROC-AUC and PR-AUC, which FuSEVul does not publish. All headline cells are five-seed means  $\pm$  one standard deviation; paired tests use the same five seeds across compared rungs. Five seeds give these tests limited power, so p-values are reported alongside effect sizes throughout, and a contrast that fails to reach significance is treated as unresolved rather than as evidence of no effect.

The model-selection procedure requires care because the public FuSEVul release ships only train and validation files; no test split exists on disk. The base paper itself partitioned each corpus 8:1:1 into train, validation, and test sets and computed its published numbers on the withheld 10% test partition [8]; that partition was never released. Every FuSEVul reference row in this paper is therefore a test-split figure, while every SemanticVul figure is computed on the released validation split, and the reported deltas are cross-partition and cross-paper. We report under two protocols. The *base-comparable* protocol reproduces FuSEVul’s own *selection rule*, choosing the best-validation-accuracy epoch on, and computing metrics over, the full validation split; because selection and scoring share that split, it is selection-circular by construction, and it is used only where parity with the published reference requires it. The *held-out* protocol selects the epoch on a stratified half of the validation split and

reports on the disjoint other half, fixed across seeds and rungs, which removes that circularity though it remains a split of the released validation data rather than an independent test set. Comparisons against FuSEVul use the base-comparable protocol for parity; every ablation claim is checked under both.

## 4.5 Implementation details

All runs use a single NVIDIA RTX 3090 (24 GB; peak use  $\approx 4.6$  GB): 12 epochs, seeds 1–5, effective batch 32 via gradient accumulation, AdamW at  $2 \times 10^{-5}$  with linear warmup/decay, 512-token windows on both channels, and the  $100\times$  gate learning-rate scale of Section 3.4. With cached explanations, the full ladder costs approximately 20.33 GPU-hours on Reveal and 83.53 on Devign in aggregate across all rungs and seeds; FuSEVul’s reported setup used 50 epochs and two RTX 3090 GPUs, so the comparison favors the baseline in training budget.

**Hyperparameter provenance.** The optimizer recipe is a standard pre-trained-model fine-tuning configuration adopted once and shared by every rung, dataset, seed, and protocol; no per-configuration hyperparameter search was run, and the recipe was not selected to maximize evaluation-split scores. Early stopping and the diagnostic threshold of Section 3.5 are computed on the train-carved tune split, which receives no gradient updates and is disjoint from the evaluation split. Two development-phase design choices, the serialized explanation field set and the focal positive weight  $\alpha = 0.85$ , were informed by exploratory validation-split probes during system development. Because the release ships no test split, this development-time exposure is shared by any system developed against these benchmarks; it is disclosed here rather than controlled, since the held-out protocol of Section 4.4 addresses epoch-selection circularity but not development-time selection.

## 5 Experimental results

### 5.1 RQ1: Overall performance

Table 5 gives the headline comparison under the base-comparable protocol on the released validation splits. The primary L2 configuration is +8.60 Accuracy and +7.31 F1 higher on Devign and +1.23 and +15.09 higher on Reveal than the reported FuSEVul reference. For the rare-positive Reveal dataset, recall rises from 39.52 to 63.00 while precision also improves (57.24 to 60.78), so the F1 jump is not bought with a precision collapse. The L3 adaptive-gate variant reaches the highest F1 on both datasets under this protocol (65.16 and 62.60), shifting the operating point toward recall. This is an exploratory result: L3–L2 contrasts are not significant across five seeds and are not consistent under held-out evaluation. We designate L2 primary because its improvement holds under both protocols (Table 7); the gate is analyzed in Section 5.3. Table 3 and Figure 3 provide a summary; the full field comparison follows in Table 4.

**Scope of the reported-reference comparison.** FuSEVul was not rerun in our environment, so differences in implementation, model versions, training choices, and development exposure cannot be excluded. The partitions also differ: FuSEVul’s published numbers are computed on its withheld 10% test split, which was never released,

**Table 3** Compact headline comparison on Devign and Reveal. SemanticVul L2 and L3 are five-seed means under the base-comparable protocol; FuSEVul is the published reference (not a local rerun).

| Dataset | Method                       | Acc   | Prec  | Recall | F1    |
|---------|------------------------------|-------|-------|--------|-------|
| Devign  | FuSEVul (reported reference) | 60.39 | 54.14 | 57.79  | 55.91 |
|         | SemanticVul L2 (ours)        | 68.99 | 65.28 | 61.52  | 63.22 |
|         | SemanticVul L3 (gate)        | 69.10 | 63.92 | 66.62  | 65.16 |
| Reveal  | FuSEVul (reported reference) | 91.68 | 57.24 | 39.52  | 46.76 |
|         | SemanticVul L2 (ours)        | 92.91 | 60.78 | 63.00  | 61.85 |
|         | SemanticVul L3 (gate)        | 92.87 | 60.46 | 65.51  | 62.60 |

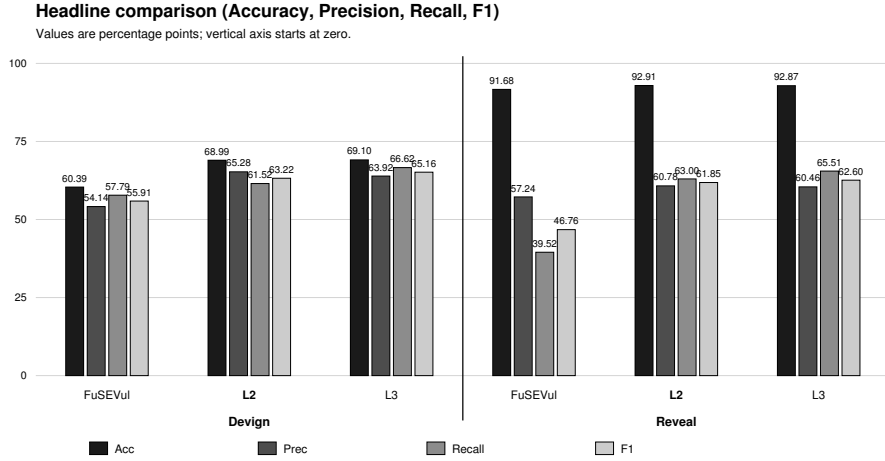


**Fig. 3** Compact black-and-white grouped bar chart for the six rows in Table 3. Grayscale fills identify Accuracy, Precision, Recall, and F1; the vertical scale starts at zero and all values are copied directly from the table.

while SemanticVul is scored on the released validation split (Section 4.4). The within-system L1 comparison isolates the explanation-bearing configuration under our own training recipe, but it does not eliminate those cross-paper confounds. The FuSEVul delta is therefore descriptive parity with its published protocol, not a controlled causal estimate or a statistical comparison against FuSEVul.

Table 4 places these results in the broader field. The upper block contains eighteen baselines plus FuSEVul, all evaluated by the base paper on the *identical* distributed splits. It is a source-attributed context table, not a rerun-based ranking. Scores vary across the supervised, pre-trained, and LLM-based groups. Supervised sequence and graph models (TokenCNN through S<sup>2</sup>FVD) plateau around 52–58 Accuracy and 42–54 F1 on Devign and top out at 33.6 F1 on Reveal: learned code representations alone struggle with the rare class. Fine-tuned pre-trained code models (UniXcoder through CAPTURE) show a modest improvement on Devign but still top out at 38.6 Reveal F1. FuSEVul’s own encoder ablation also shows little movement when these encoders

are swapped. Prompting LLMs for the verdict directly performs poorly (ChatGPT-3.5: 22.28 and 8.91 F1; CodeLlama-7B: 11.20 and 6.55). Explanation fusion (FuSEVul) is the strongest prior technique on both datasets, and SemanticVul extends that design: +7.31 to +9.25 F1 on Devign and +15.09 to +15.84 on Reveal over the best of the same-split field.

The † block reports recent systems’ *self-reported* numbers on their own preparations of the same corpora; their splits, deduplication, and class balance differ, so these rows are indicative rather than head-to-head. Two factors explain their Devign strength. First, all four augment the raw token view with *program structure*: SAHAN with syntax-aware hierarchical attention over the AST, GRACE with graph structure and retrieved examples inside GPT-4 prompts, CTVD with a collaboratively trained GNN, and CosFormer with control- and program-dependence-weighted attention. This is an input axis separate from SemanticVul’s explanation channel. Of these, GRACE also depends on a commercial GPT-4 backend, where SemanticVul runs fully local. Second, evaluation protocol: CosFormer’s 76.11 Devign F1 is reported on a *random re-split*, and the audit of Section 4.2 together with prior benchmark studies [13, 15] shows that random re-splits of these duplicate-laden corpora leak near-identical functions across partitions and inflate scores. This is why SemanticVul’s head-to-head claim is restricted to the split-identical block. On the imbalanced Reveal, the SemanticVul values are higher than the listed self-reported values, but this should not be read as a definitive ranking across incompatible preparations. Structural and inter-procedural context could complement grounded explanations; this extension is left for future work (Section 6).

## 5.2 RQ2: Contribution of grounded explanations

The full retained explanation channel carries the bulk of the gain. Moving from L1 to L2 (Table 7) adds 9.56 F1 on Devign and 17.16 on Reveal under the base-comparable protocol, and 11.31 and 17.51 under the held-out protocol, with  $p < .001$  on both datasets. For context, FuSEVul’s own explanation-ablation reports a drop of 8.18 F1 on Reveal when its ChatGPT explanations are removed. That value is reported only as cross-paper context: different implementations and protocols make its magnitude unsuitable for a direct effect-size comparison. Figure 4 shows the corresponding threshold-free view: the L1→L2 lift is visible across the full ROC and precision–recall curves. The PR gain is especially pronounced on Reveal, where it is the more informative metric for the rare vulnerable class.

The channel is useful and its outputs are more textually grounded (Table 6): on the 2,269 matched Reveal validation functions, API-mention grounding more than doubles, absent-call claims drop from 76.0% to 46.7%, and hallucinated call mentions halve. Here, API-mention grounding means the fraction of explanations whose named API is found in the normalized function; an absent-call claim names an API that is not found; and hallucinated call mentions count such unsupported call names per 100 functions. Evidence-span exactness is the fraction of emitted evidence anchors that match a normalized code substring exactly. SemanticVul also provides explicit evidence anchors for 87.0% of functions, 97.2% of which match verbatim code substrings. These evidence-anchor metrics (the middle block of Table 6) are defined only for a schema

**Table 4** Comparison with prior detectors on Devign and Reveal (Accuracy and F1). Rows above the † block are reported by the base paper [8, Table 2]; † rows are self-reported on the cited papers’ own preparations. SemanticVul rows are five-seed means under the base-comparable protocol.

| Group              | Method                | Devign       |              | Reveal       |              |
|--------------------|-----------------------|--------------|--------------|--------------|--------------|
|                    |                       | Acc          | F1           | Acc          | F1           |
| Supervised         | TokenCNN              | 54.25        | 45.27        | 87.42        | 24.34        |
|                    | TokenRNN              | 53.26        | 41.72        | 86.10        | 26.51        |
|                    | VulCNN                | 56.17        | 48.87        | 87.75        | 33.60        |
|                    | SySeVR                | 51.84        | 49.72        | 87.93        | 25.34        |
|                    | Reveal                | 54.37        | 43.48        | 89.07        | 26.97        |
|                    | TrVD                  | 56.19        | 49.13        | 83.47        | 32.37        |
|                    | DeepVD                | 57.50        | 51.02        | 89.85        | 25.59        |
|                    | S <sup>2</sup> FVD    | 56.70        | 54.05        | 89.16        | 30.18        |
| Pre-trained        | UniXcoder             | 57.43        | 54.96        | 90.14        | 33.44        |
|                    | CodeBERT              | 57.91        | 53.82        | 90.54        | 32.60        |
|                    | GraphCodeBERT         | 59.63        | 54.55        | 90.06        | 35.06        |
|                    | CodeT5                | 58.86        | 53.78        | 90.05        | 32.74        |
|                    | CodeT5+               | 58.96        | 54.30        | 90.05        | 38.58        |
|                    | LineVul               | 59.66        | 54.24        | 87.68        | 30.35        |
|                    | EPVD                  | 59.63        | 53.40        | 87.68        | 37.22        |
|                    | CAPTURE               | 58.57        | 53.53        | 86.32        | 30.11        |
| LLM as detector    | ChatGPT-3.5           | 53.85        | 22.28        | 81.77        | 8.91         |
|                    | CodeLlama-7B          | 53.44        | 11.20        | 83.49        | 6.55         |
| Explanation fusion | FuSEVul               | 60.39        | 55.91        | 91.68        | 46.76        |
| Self-reported†     | SAHAN [33]            | 61.34        | 62.77        | 87.55        | 48.26        |
|                    | GRACE [19]            | 59.78        | 65.11        | 89.73        | 43.13        |
|                    | CTVD [24]             | 87.3         | 34.4         | 85.1         | 39.7         |
|                    | CosFormer [12]        | 75.92        | 76.11        | 93.72        | 51.36        |
| This work          | <b>SemanticVul L2</b> | <b>68.99</b> | <b>63.22</b> | <b>92.91</b> | <b>61.85</b> |
|                    | SemanticVul L3 (gate) | 69.10        | 65.16        | 92.87        | 62.60        |

**Table 5** Main comparison under the base-comparable protocol; cells are five-seed mean  $\pm$  one standard deviation.  $\Delta$  columns are descriptive differences against the reported FuSEVul reference. L2 (bold) is the primary configuration; L3 is the exploratory adaptive-gate variant.

| Dataset | Method           | Acc                              | Prec             | Recall           | F1                               | $\Delta$ Acc | $\Delta$ F1   |
|---------|------------------|----------------------------------|------------------|------------------|----------------------------------|--------------|---------------|
| Devign  | FuSEVul          | 60.39                            | 54.14            | 57.79            | 55.91                            | —            | —             |
| Devign  | <b>L2 (ours)</b> | <b>68.99<math>\pm</math>0.24</b> | 65.28 $\pm$ 1.50 | 61.52 $\pm$ 4.58 | <b>63.22<math>\pm</math>1.74</b> | <b>+8.60</b> | <b>+7.31</b>  |
| Devign  | L3 gate (ours)   | 69.10 $\pm$ 0.61                 | 63.92 $\pm$ 1.35 | 66.62 $\pm$ 4.10 | 65.16 $\pm$ 1.40                 | +8.71        | +9.25         |
| Reveal  | FuSEVul          | 91.68                            | 57.24            | 39.52            | 46.76                            | —            | —             |
| Reveal  | <b>L2 (ours)</b> | <b>92.91<math>\pm</math>0.58</b> | 60.78 $\pm$ 3.04 | 63.00 $\pm$ 3.69 | <b>61.85<math>\pm</math>3.15</b> | <b>+1.23</b> | <b>+15.09</b> |
| Reveal  | L3 gate (ours)   | 92.87 $\pm$ 0.60                 | 60.46 $\pm$ 3.93 | 65.51 $\pm$ 6.07 | 62.60 $\pm$ 1.89                 | +1.19        | +15.84        |

that emits explicit evidence spans, so the FuSEVul cells are n/a by construction not losses; free-form prose provides no comparable anchors. Conversely, quoted-identifier grounding is not comparable in the other direction: it scores the identifiers a system quotes inline, and SemanticVul carries evidence in `evidence.tokens` rather than as inline quotes, so its denominator is near-empty. Verdict scrubbing works: direct verdict leakage in the fed text is 0.2% of Devign rows and 0.0% of Reveal rows.

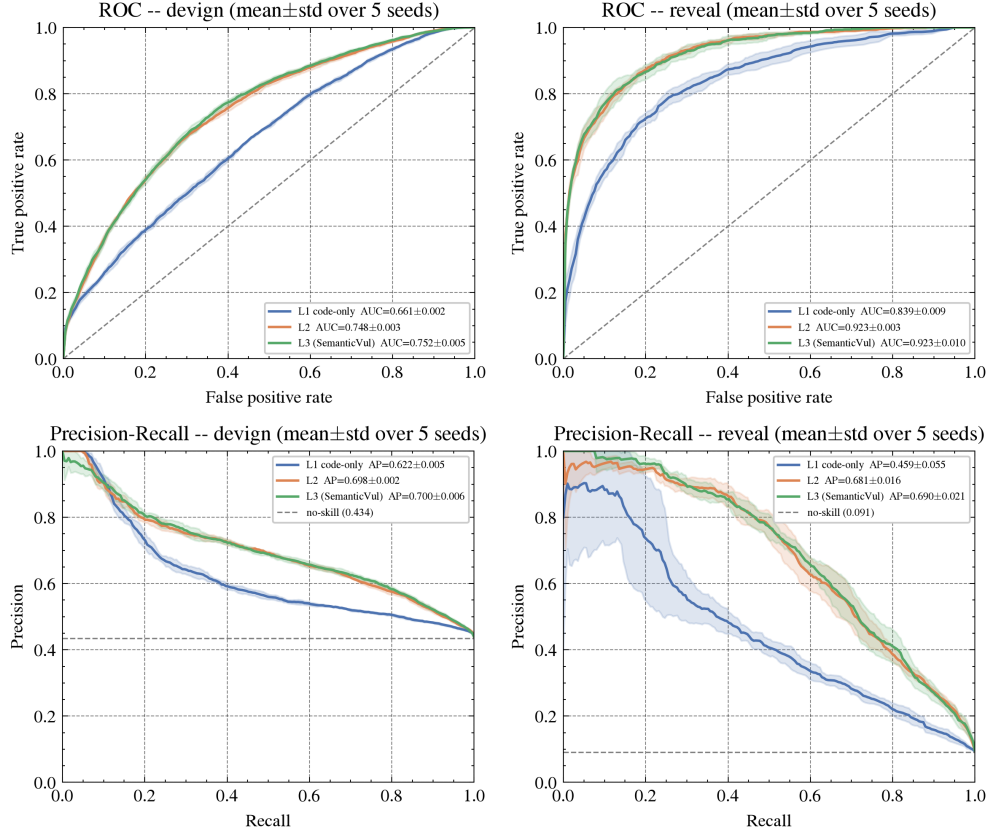


**Fig. 4** ROC (top) and precision-recall (bottom) curves for the L1-L3 ladder on Devign (left) and Reveal (right) at the best-validation-accuracy epoch. The dominant separation is between L1 and the two explanation-fusing rungs.

Three properties of this audit bound what it shows. First, it is lexical: it establishes that claims are anchored in the function’s text, not that they are semantically correct, and the guarantee is partly by construction, since the validator drops ungrounded claims before encoding. That is the mechanism working as designed rather than a confound, but it does mean the grounding figures describe the pipeline’s output contract, not an emergent property of the generator. Second, grounding quality is dataset-dependent: mean evidence-code overlap is 0.998 on Reveal against 0.415 on Devign, and correspondingly 99.7% of Reveal explanations have all emitted evidence anchors grounded against 33.5% on Devign. We do not isolate a cause for this gap, and identifying it is left to the generator sweep of Section 6. Its consequence is bounded only for evidence-linked fields: the validator drops unsupported evidence tokens, risky-operation claims, and guard indicators. Purpose, data-flow, and risk-summary prose is verdict-scrubbed but is not semantically verified, so the complete encoder input must not be described as fully grounded. That the smaller retained channel coincides with the smaller explanation-channel gain on Devign (+9.56 F1 against +17.16 on

**Table 6** Automatic explanation-grounding checks on the 2,269 matched Reveal validation functions. Only the upper block is a head-to-head comparison; bold marks the better value there.

| Faithfulness metric                                           | FuSEVul | SemanticVul  |
|---------------------------------------------------------------|---------|--------------|
| API-mention grounding ( $\uparrow$ )                          | 26.2%   | <b>55.4%</b> |
| API-naming explanations with an absent call ( $\downarrow$ )  | 76.0%   | <b>46.7%</b> |
| Hallucinated call mentions per 100 functions ( $\downarrow$ ) | 18.8    | <b>9.2</b>   |
| <i>Evidence-anchor metrics (schema-specific)</i>              |         |              |
| Functions with an explicit evidence anchor ( $\uparrow$ )     | 0%      | <b>87.0%</b> |
| Evidence-span exactness ( $\uparrow$ )                        | n/a     | <b>97.2%</b> |
| <i>Not comparable across systems (see caption)</i>            |         |              |
| Quoted-identifier grounding                                   | 99.2%   | n/a          |

Reveal) is consistent with this reading, though the two are not causally tied here. Third, the measured confidence  $\kappa$  does carry label-correlated information: on its own it separates the classes with ROC-AUC 0.680 on Devign and 0.874 on Reveal. This is expected and is not supervision leakage, because the generator never sees the ground-truth label;  $\kappa$  is an unsupervised risk judgment, and its predictive value is part of the measured explanation channel. It also prevents attributing the channel’s gain to descriptive explanation content alone: the gain should be read as combining the generator’s risk assessment with its descriptive content, and a field-level ablation isolating the two is the natural next measurement (Section 6).

### 5.3 RQ3: Adaptive modality weighting versus static fusion

Table 7 reports the full ladder under both protocols. Under the base-comparable protocol the adaptive gate delivers the best F1 on both datasets, adding +1.94 over L2 on Devign and +0.75 on Reveal, and it consistently shifts the operating point toward recall (66.62 vs. 61.52 on Devign; 65.51 vs. 63.00 on Reveal). Under the held-out protocol the picture is protocol-sensitive: the gate’s F1 edge reverses on Devign (−2.48) and stays within seed noise on Reveal (+0.44).

Seed-matched paired  $t$ -tests quantify the observed attribution, but the five seeds provide limited statistical power. L3 clears the code-only rung decisively on every metric, dataset, and protocol: F1 gains of +11.50 ( $p = .006$ ) and +8.83 ( $p = .040$ ) on Devign and +17.91 and +17.95 (both  $p < .001$ ) on Reveal, with MCC gains of +15.57 to +20.04 (all  $p < .001$ ). The L3-versus-L2 contrasts, by comparison, do not reach significance with five seeds (smallest  $p = .163$  across all eight metric–protocol–dataset combinations), so the gate’s F1 lead is reported strictly as a base-comparable-protocol result while the explanation channel is the claim that generalizes.

**Quality-feature mechanism study.** A complementary frozen-cache experiment asks whether conditioning the gate on the 22 label-free quality features adds predictive value: it does not reliably beat static concatenation or a no-quality gated twin (ROC-AUC 64.76 vs. 64.58 and 65.71 on Devign; 82.43 vs. 83.17 and 82.66 on Reveal), and per-dimension gate variation is small (0.06–0.12). The features are used for the faithfulness, grounding, and leakage audits in Section 5.2, while the final detector stays

**Table 7** L1–L3 ablation ladder under the base-comparable and held-out protocols; five-seed mean  $\pm$  one standard deviation. L2 (bold) is the primary configuration.

| Stage                      | Protocol          | Acc                              | F1                               | Recall                           | Precision                        |
|----------------------------|-------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|
| <i>Devign</i>              |                   |                                  |                                  |                                  |                                  |
| L1 code-only               | base-comp.        | 61.30 $\pm$ 0.96                 | 53.66 $\pm$ 3.48                 | 52.13 $\pm$ 8.25                 | 56.23 $\pm$ 2.47                 |
| L1 code-only               | held-out          | 62.11 $\pm$ 1.50                 | 53.22 $\pm$ 2.40                 | 49.80 $\pm$ 5.64                 | 57.76 $\pm$ 3.29                 |
| <b>L2 code+explanation</b> | <b>base-comp.</b> | <b>68.99<math>\pm</math>0.24</b> | <b>63.22<math>\pm</math>1.74</b> | <b>61.52<math>\pm</math>4.58</b> | <b>65.28<math>\pm</math>1.50</b> |
| <b>L2 code+explanation</b> | <b>held-out</b>   | <b>70.19<math>\pm</math>0.70</b> | <b>64.53<math>\pm</math>1.67</b> | <b>62.46<math>\pm</math>3.46</b> | <b>66.85<math>\pm</math>0.97</b> |
| L3 adaptive gate           | base-comp.        | 69.10 $\pm$ 0.61                 | 65.16 $\pm$ 1.40                 | 66.62 $\pm$ 4.10                 | 63.92 $\pm$ 1.35                 |
| L3 adaptive gate           | held-out          | 69.44 $\pm$ 1.33                 | 62.05 $\pm$ 5.62                 | 58.62 $\pm$ 11.19                | 67.63 $\pm$ 3.92                 |
| <i>Reveal</i>              |                   |                                  |                                  |                                  |                                  |
| L1 code-only               | base-comp.        | 89.08 $\pm$ 1.51                 | 44.69 $\pm$ 1.42                 | 48.12 $\pm$ 3.86                 | 42.36 $\pm$ 5.17                 |
| L1 code-only               | held-out          | 89.09 $\pm$ 1.50                 | 47.01 $\pm$ 2.79                 | 52.50 $\pm$ 3.51                 | 42.99 $\pm$ 5.17                 |
| <b>L2 code+explanation</b> | <b>base-comp.</b> | <b>92.91<math>\pm</math>0.58</b> | <b>61.85<math>\pm</math>3.15</b> | <b>63.00<math>\pm</math>3.69</b> | <b>60.78<math>\pm</math>3.04</b> |
| <b>L2 code+explanation</b> | <b>held-out</b>   | <b>93.29<math>\pm</math>0.71</b> | <b>64.52<math>\pm</math>3.12</b> | <b>66.54<math>\pm</math>3.07</b> | <b>62.71<math>\pm</math>3.95</b> |
| L3 adaptive gate           | base-comp.        | 92.87 $\pm$ 0.60                 | 62.60 $\pm$ 1.89                 | 65.51 $\pm$ 6.07                 | 60.46 $\pm$ 3.93                 |
| L3 adaptive gate           | held-out          | 93.13 $\pm$ 0.92                 | 64.96 $\pm$ 2.14                 | 69.23 $\pm$ 4.35                 | 61.72 $\pm$ 5.82                 |

simple. The per-sample weight  $w$  is reported as a routing diagnostic, not as a faithful explanation of the final nonlinear prediction.

## 5.4 RQ4: Operating points and ensembling

Focal loss with capped positive weighting is fixed throughout the Reveal ladder and is not treated as a separately measured contribution. This section therefore evaluates operating-point sensitivity rather than the causal effect of imbalance-aware loss. The observed gain occurs at the natural 0.5 threshold, and the remaining optimization levers are modest and dataset-specific.

Table 8 probes sensitivity around the deployed final epoch. On Reveal, a threshold tuned on the train-carved tune split ( $\tau = 0.59$ ) adds +2.53 F1 without touching evaluation data, and the validation-tuned upper bound adds +3.46; on Devign the tuned threshold is flat ( $-0.10$ ), so threshold tuning is a Reveal-specific lever. The Devign upper-bound threshold falls to 0.05, which reflects the final epoch’s poorly calibrated, downward-compressed probabilities on that dataset and is a further reason the headline protocol reports the best-accuracy epoch at 0.5 rather than this one. Multi-seed rank-search ensembling improves threshold-free ranking on both datasets (ROC-AUC 75.22 to 77.27 on Devign, 92.33 to 93.09 on Reveal; PR-AUC +2.45 and +2.50) and, combined with a tuned threshold at the final epoch, lifts F1 by +3.59 (Devign) and +2.48 (Reveal). Both the ensemble weights and this threshold are selected on the split they are scored on, so these rows are optimistic upper bounds on what multi-seed combination can add, reported for completeness rather than as deployment estimates.

## 6 Limitations and future work

SemanticVul’s claims are limited to function-level binary classification of C/C++ under the audited Devign and Reveal splits. Vulnerable-statement localization, cross-function reasoning, and adversarial robustness are outside scope, and all explanations come from a single generator class. Five extensions remain for future work: a

**Table 8** Operating-point diagnostics and multi-seed ensembling. Threshold rows are L3 at the deployed final epoch; ensemble rows are at the headline epoch. Rows marked *upper bound* select on the split they are scored on.

| Dataset | Variant                             | Thr. | F1               | Recall           | $\Delta F1$ |
|---------|-------------------------------------|------|------------------|------------------|-------------|
| Devign  | L3 @0.5 (final epoch)               | 0.50 | 59.08 $\pm$ 0.96 | 59.92 $\pm$ 2.06 | —           |
| Devign  | Tuned on tune split                 | 0.49 | 58.98 $\pm$ 1.61 | 60.02 $\pm$ 3.43 | −0.10       |
| Devign  | Tuned on validation (upper bound)   | 0.05 | 62.36 $\pm$ 0.59 | 72.03 $\pm$ 1.98 | +3.28       |
| Devign  | Ensemble + tuned thr. (upper bound) | —    | 62.67            | 67.82            | +3.59       |
| Reveal  | L3 @0.5 (final epoch)               | 0.50 | 60.57 $\pm$ 3.14 | 73.91 $\pm$ 7.12 | —           |
| Reveal  | Tuned on tune split                 | 0.59 | 63.10 $\pm$ 1.16 | 60.39 $\pm$ 3.11 | +2.53       |
| Reveal  | Tuned on validation (upper bound)   | 0.57 | 64.03 $\pm$ 0.98 | 64.93 $\pm$ 4.93 | +3.46       |
| Reveal  | Ensemble + tuned thr. (upper bound) | —    | 63.05            | 58.94            | +2.48       |

hardware-matched FuSEVul re-run with a fresh project-level holdout, upgrading the reported-reference comparison into a controlled one; a field-level ablation of the serialized explanation text, dropping the measured confidence  $\kappa$  to separate the generator’s risk judgment from its descriptive content (Section 5.2); a paired prompt-and-generator sweep (e.g., DeepSeek-Coder, CodeLlama, Phi-3) to test how much of the grounding advantage is generator-specific; an LLM-as-judge pass to cross-check the automatic faithfulness metrics with independent semantic judgments; and grounded summaries of called functions in the spirit of Wu et al. [25], extending the pipeline from its current function-level scope toward inter-procedural defects.

## 7 Conclusion

SemanticVul combines a schema-constrained, evidence-grounded, verdict-scrubbed explanation pipeline with a controlled L1–L3 fusion ladder. On the audited Devign and Reveal splits, the grounded explanation channel provides the largest gain, up to +17.5 F1 over code-only. SemanticVul is higher than the reported FuSEVul reference on both Accuracy and F1 on both datasets (+7.31 and +15.09 F1); this is a reported-reference comparison rather than a controlled rerun. Under the base-comparable protocol, the exploratory adaptive-gate variant raises F1 further and shifts operating points toward recall, but its effect is not resolved under held-out evaluation. API-mention grounding roughly doubles, hallucinated call mentions halve, and the split audit documents contradictory labels and conditional memorization bounds. The pipeline uses a local open-weight generator, a label-blind prompt, and a grounding validator. The matched ladder compares the explanation channel and alternative fusion mechanisms, while the mechanism study uses the quality features for auditing rather than prediction. The results support auditing explanations before fusing them and measuring the explanation channel separately from the fusion mechanism that carries it.

## Declarations

- **Funding:** The authors received no specific funding for this work.
- **Conflict of interest:** The authors declare no conflict of interest.

- **Ethics approval:** Not applicable. This study involves no human participants or animals; it uses only publicly released source-code datasets.
- **Consent to participate:** Not applicable.
- **Consent for publication:** All authors have read and approved the manuscript for publication.
- **Data availability:** The Devign [3] and Reveal [4] datasets are publicly available; the experiments use the split files distributed with the FuSEVul release [8].
- **Materials availability:** Not applicable.
- **Code availability:** The full implementation, including the explanation pipeline, training scripts, and analysis code, is publicly available at <https://github.com/sendmesymbols/semVul>.
- **Author contribution:** A.R. conceived the study, implemented the explanation pipeline and the fusion ladder, ran the experiments, carried out the analysis, and drafted the manuscript. I.U.I. and R.K. supervised the work, advised on the experimental design and evaluation protocol, and reviewed and edited the manuscript. All authors read and approved the final manuscript.

## References

- [1] MITRE: Common Vulnerabilities and Exposures (CVE) (2024). <https://cve.mitre.org/> Accessed 2026-05-31
- [2] MITRE: Common Weakness Enumeration (CWE) (2024). <https://cwe.mitre.org/> Accessed 2026-05-31
- [3] Zhou, Y., Liu, S., Siow, J., Du, X., Liu, Y.: Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. In: Advances in Neural Information Processing Systems (NeurIPS), vol. 32 (2019)
- [4] Chakraborty, S., Krishna, R., Ding, Y., Ray, B.: Deep learning based vulnerability detection: Are we there yet? IEEE Transactions on Software Engineering **48**(9), 3280–3296 (2022)
- [5] Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., Zhou, M.: CodeBERT: A pre-trained model for programming and natural languages. In: Findings of the Association for Computational Linguistics (EMNLP), pp. 1536–1547 (2020)
- [6] Guo, D., Lu, S., Duan, N., Wang, Y., Zhou, M., Yin, J.: UniXcoder: Unified cross-modal pre-training for code representation. In: Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL), pp. 7212–7225 (2022)
- [7] Zhou, X., Cao, S., Sun, X., Lo, D.: Large language model for vulnerability detection and repair: Literature review and the road ahead. ACM Transactions on Software Engineering and Methodology **34**(5), 145 (2025) <https://doi.org/10.>

- [8] Tian, Z., Li, M., Sun, J., Chen, Y., Chen, L.: Enhancing vulnerability detection by fusing code semantic features with LLM-generated explanations. *Information Fusion* **125**, 103450 (2026) <https://doi.org/10.1016/j.inffus.2025.103450>
- [9] Yu, L., *et al.*: Smart-LLaMA-DPO: Reinforced large language model for explainable smart contract vulnerability detection. *Proceedings of the ACM on Software Engineering* **2**(ISSTA), 009 (2025) <https://doi.org/10.1145/3728878>
- [10] Lu, L., Zhao, Y., Rao, H., Zhang, K., Wang, H.: Evaluating and enhancing the vulnerability reasoning capabilities of large language models (DAGVul). Technical Report 2602.06687v1, arXiv (2026)
- [11] Germano, L.B., Goldschmidt, R.R., Choren Noya, R., Duarte, J.C.: A systematic review on detection, repair, and explanation of vulnerabilities in source code using large language models. *IEEE Access* **13**, 192263–192293 (2025) <https://doi.org/10.1109/ACCESS.2025.3631363>
- [12] Wei, Q., Zhang, Z., Yi, Q., Ji, Z., Liang, H.: CosFormer: A code semantic-aware transformer for vulnerability detection. *IEEE Transactions on Software Engineering*, 1–16 (2026) <https://doi.org/10.1109/TSE.2026.3659630> . early access
- [13] Ding, Y., Fu, Y., Ibrahim, O., Sitawarin, C., Chen, X., Alomair, B., Wagner, D., Ray, B., Chen, Y.: Vulnerability detection with code language models: How far are we? In: *Proc. 47th IEEE/ACM International Conference on Software Engineering (ICSE)* (2025). <https://doi.org/10.1109/ICSE55347.2025.00038>
- [14] Chakraborty, P., Arumugam, K.K., Alfadel, M., Nagappan, M., McIntosh, S.: Revisiting the performance of deep learning-based vulnerability detection on realistic datasets. *IEEE Transactions on Software Engineering* (2024) <https://doi.org/10.1109/TSE.2024.3423712>
- [15] Risse, N., Liu, J., Böhme, M.: Top score on the wrong exam: On benchmarking in machine learning for vulnerability detection. *Proceedings of the ACM on Software Engineering* **2**(ISSTA) (2025) <https://doi.org/10.1145/3728887>
- [16] Guo, D., Ren, S., Lu, S., Feng, Z., Tang, D., Liu, S., Zhou, L., Duan, N., Svyatkovskiy, A., Fu, S., Tufano, M., Deng, S.K., Clement, C., Drain, D., Sundaresan, N., Yin, J., Jiang, D., Zhou, M.: GraphCodeBERT: Pre-training code representations with data flow. In: *International Conference on Learning Representations (ICLR)* (2021)
- [17] Wang, Y., Le, H., Gotmare, A., Bui, N., Li, J., Hoi, S.: CodeT5+: Open code large language models for code understanding and generation. In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1069–1088 (2023)

- [18] Chen, Y., Huang, Y., Chen, X., Shen, P., Yun, L.: GPTVD: Vulnerability detection and analysis method based on LLM’s chain of thoughts. *Automated Software Engineering* **33**(3) (2026)
- [19] Lu, G., Ju, X., Chen, X., Pei, W., Cai, Z.: GRACE: Empowering LLM-based software vulnerability detection with graph structure and in-context learning. *Journal of Systems and Software* **212**, 112031 (2024) <https://doi.org/10.1016/j.jss.2024.112031>
- [20] Lekssays, A., Mouhcine, H., Tran, K., Yu, T., Khalil, I.: LLMxCPG: Context-aware vulnerability detection through code property graph-guided large language models. In: *Proc. 34th USENIX Security Symposium*, pp. 489–507 (2025)
- [21] Yang, Y., Zhou, X., Mao, R., Xu, J., Yang, L., Zhang, Y., Shen, H., Zhang, H.: DLAP: A deep learning augmented large language model prompting framework for software vulnerability detection. *Journal of Systems and Software* **219**, 112234 (2025) <https://doi.org/10.1016/j.jss.2024.112234>
- [22] Mao, Q., Li, Z., Hu, X., Liu, K., Xia, X., Sun, J.: Towards explainable vulnerability detection with large language models. *IEEE Transactions on Software Engineering* **51**(10), 2957–2971 (2025) <https://doi.org/10.1109/TSE.2025.3605442>
- [23] Khare, A., Dutta, S., Li, Z., Solko-Breslin, A., Alur, R., Naik, M.: Understanding the effectiveness of large language models in detecting security vulnerabilities. In: *Proc. 18th IEEE Conference on Software Testing, Verification and Validation (ICST)*, pp. 103–114 (2025). <https://doi.org/10.1109/ICST62969.2025.10988968>
- [24] Zheng, Y., Wang, D., Cao, H., Qian, C., Zhuang, H.: CTVD: Collaborative training of deep learning and large model for C/C++ source code vulnerability detection. In: *Proc. IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 2885–2892 (2025). <https://doi.org/10.1109/SMC58881.2025.11343541>
- [25] Wu, B., Liu, C., Li, Z., Cao, Y., Sun, J., Lin, S.-W.: Enhancing vulnerability detection via inter-procedural semantic completion. *Proceedings of the ACM on Software Engineering* **2**(ISSTA), 825–847 (2025) <https://doi.org/10.1145/3728912>
- [26] Wang, Z., *et al.*: M2CVD: Enhancing vulnerability understanding through multi-model collaboration for code vulnerability detection. *ACM Transactions on Software Engineering and Methodology* (2025) <https://doi.org/10.1145/3771923>. Full author list (arXiv:2406.05940): Z. Wang, G. Li, J. Li, Y. Xiong, M. Yan, Z. Jin.
- [27] Hui, B., et al.: Qwen2.5-Coder technical report. Technical Report 2409.12186, arXiv (2024)

- [28] Ollama: Ollama: Run large language models locally (2024). <https://ollama.com/> Accessed 2026-05-31
- [29] Kadavath, S., Conerly, T., Askell, A., Henighan, T., Drain, D., Perez, E., Schiefer, N., Hatfield-Dodds, Z., DasSarma, N., Tran-Johnson, E., et al.: Language models (mostly) know what they know. arXiv preprint arXiv:2207.05221, Anthropic (2022). <https://arxiv.org/abs/2207.05221>
- [30] Guo, C., Pleiss, G., Sun, Y., Weinberger, K.Q.: On calibration of modern neural networks. In: Proceedings of the 34th International Conference on Machine Learning (ICML), pp. 1321–1330 (2017)
- [31] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V.: RoBERTa: A robustly optimized BERT pretraining approach. Technical Report 1907.11692, arXiv (2019)
- [32] Lin, T.-Y., Goyal, P., Girshick, R., He, K., Dollár, P.: Focal loss for dense object detection. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV), pp. 2980–2988 (2017)
- [33] Jiang, Y., Duan, B., et al.: Syntax-aware hierarchical attention networks for code vulnerability detection. *Computers, Materials and Continua* **86**(1) (2026)