CONTRACTLENS: LATENT SECURITY KNOWLEDGE FOR MALICIOUS SMART CONTRACT DETECTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Detecting malicious smart contracts is essential to safeguarding the Web3.0 ecosystem. However, existing auditing methods based on large language models (LLMs) largely rely on prompt engineering or attack-specific fine-tuning, while the internal representations that support malicious logic detection remain poorly characterized. To address this gap, we propose ContractLens, a framework that localizes and uses latent security knowledge in pretrained LLMs for interpretable malicious smart contract detection. Specifically, we first use layerwise linear probes to identify candidate layers that distinguish malicious from benign contracts, then select critical layers based on their sensitivity to malicious logic removal and stability under behaviour-preserving surface modifications. Next, within these critical layers, we combine activation contrast, malicious-logic specificity, and probe weights into a tri-factor importance score and adaptively select malicious contract-discriminative units (MCDUs) using the knee point of the ranked score curve. Finally, we concatenate the selected units' activations into a compact representation and train a lightweight classifier for detection, keeping the pretrained model frozen throughout. Neuron masking experiments show that masking MCDUs causes larger drops in detection performance than masking randomly selected neurons or neurons matched in activation magnitude. ContractLens achieves higher average detection accuracy than the evaluated methods on all three pretrained models. Further evaluations demonstrate generalization to unseen malicious intents and show that, with the identified MCDUs fixed, a classifier trained on few labelled samples can still achieve strong detection performance.

1 INTRODUCTION

Smart contracts (Zheng et al., 2020) manage digital assets by executing predefined rules and form core infrastructure for decentralized finance (Chen & Bellavitis, 2020), digital asset trading, and on-chain governance. However, financial fraud and other malicious activities continue to threaten this ecosystem (Miao et al., 2025; 2026). Because deployed contract is typically difficult to modify directly, hidden malicious logic may continue to threaten users' assets during subsequent interactions. Accurately detecting such logic is therefore important for safeguarding the Web3.0 ecosystem.

Existing smart contract security analysis follows three main technical routes: program analysis, supervised learning, and auditing with large language models (LLMs). Tools such as Oyente, Securify, and Slither detect violations of specified security properties through symbolic execution, dependency analysis, or static analysis (Badruddoja et al., 2021; Tsankov et al., 2018; Feist et al., 2019), with their coverage depending on the properties and rules encoded in the analyzer. Supervised methods such as SmartIntentNN and its successor (Huang et al., 2025b; 2026) extend detection to malicious intent by learning mappings from contract representations to intent labels. More recently, LLM-based approaches (Wei et al., 2024b; Sun et al., 2024; Ding et al., 2025; Jie et al., 2025) use prompt design, external knowledge, and agent workflows to elicit code understanding and vulnerability reasoning. However, as illustrated in Figure 1, these learned and LLM-assisted approaches emphasize prediction quality and auditing workflows, leaving unclear which layers and neurons in pretrained models encode discriminative information and how they support malicious logic detection.

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.

We draw on experienced developers' ability to flag a purported refund function that redirects funds to the contract controller as suspicious before identifying the attack type. Their judgment hinges on the mismatch between the function's stated purpose and its actual effect. This motivates our hypothesis that LLMs pretrained on smart contract code may encode similar security-relevant distinctions. To test this hypothesis, we must determine whether the relevant internal signals reflect malicious logic rather than superficial code correlations, and whether they contribute to detection. We therefore seek representations that change when malicious logic is removed but remain stable under behaviour-preserving edits, then assess their functional contribution through neuron interventions.

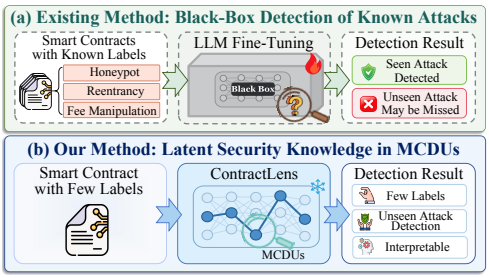


Figure 1: **Latent security knowledge for interpretable smart contract auditing.** ContractLens localizes MCDUs in pretrained LLMs and uses their activations to detect malicious contracts.

Building on this hypothesis, we propose ContractLens, a framework that localizes and uses latent security knowledge in pretrained LLMs for interpretable malicious smart contract detection. We first train layerwise linear probes on the hidden representations of a frozen LLM to identify candidate layers that distinguish malicious from benign contracts. Among these candidates, we select critical layers by testing sensitivity to malicious logic removal and stability under behaviour-preserving surface modifications. Next, within the multilayer perceptron (MLP) modules of these layers, we combine activation contrast, malicious-logic specificity, and probe weights into a tri-factor importance score and adaptively select malicious contract-discriminative units (MCDUs) using the knee point of the ranked score curve. Finally, we concatenate MCDUs activations into a compact representation and train a lightweight classifier for detection, keeping the pretrained model frozen throughout. Neuron masking experiments show that masking MCDUs causes larger drops in detection performance than masking randomly selected neurons or neurons matched in activation magnitude. The main contributions are summarized as follows:

- We propose ContractLens, an interpretable framework that identifies critical layers and neurons in frozen pretrained LLMs and uses their latent security knowledge for malicious smart contract detection.
- We identify MCDUs through tri-factor scoring and adaptive selection, showing that their activations enable effective detection and that masking them causes larger performance drops than matched neuron controls.
- Experiments show higher average detection accuracy than evaluated baselines across three models, generalization to unseen malicious intents, and effective classifier training with few labelled samples using fixed MCDUs.

2 Related Work

LLM-based Smart Contract Security Analysis. Program-analysis tools detect violations of specified security properties in smart contracts (Badruddoja et al., 2021; Tsankov et al., 2018). LLM-based auditing complements these approaches by using code understanding and semantic reasoning to inspect contract behaviour. LLM-SmartAudit employs specialized agents and structured analysis prompts (Wei et al., 2025), while LLM4Vuln systematically evaluates the effects of external vulnerability knowledge, program context, and prompt design (Sun et al., 2024). SmartGuard combines retrieved code examples with chain-of-thought reasoning for in-context learning (Ding et al., 2025), and Agent4Vul integrates LLM-derived semantic information with structural program representations (Jie et al., 2025). Further work explores business-invariant-guided contrastive auditing (Gao et al., 2026), knowledge distillation into lightweight auditing models (Wei et al., 2026), and domain-adapted code representations for unsafe intent detection (Huang et al., 2025c). However, these approaches emphasize generated analyses, auditing workflows, or task-specific representations, leaving LLM components supporting malicious smart contract detection underexplored.

Internal Representations and Neuron-level Interpretability. Mechanistic interpretability investigates how information is represented and used within language models (Bereska & Gavves, 2024).

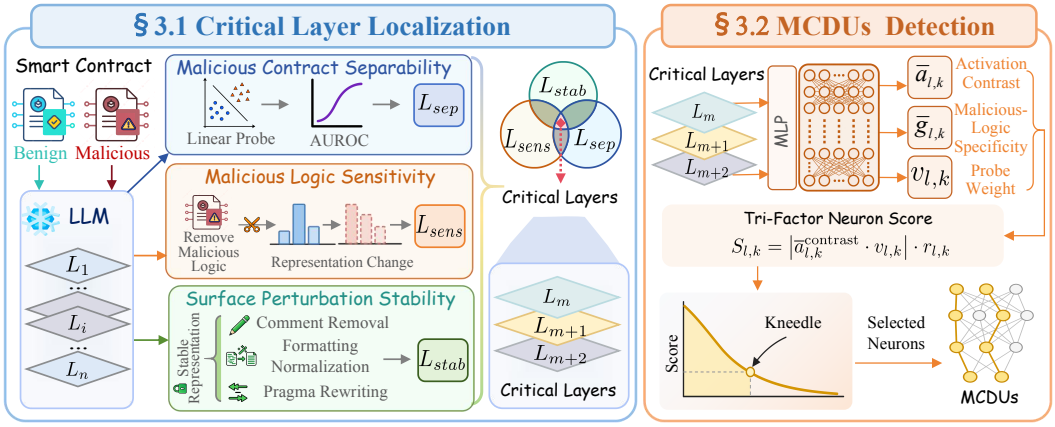


Figure 2: **Overview of the proposed ContractLens.** (1) Critical layer localization: Joint evaluation of malicious contract separability, malicious logic sensitivity, and surface perturbation stability to locate critical layers. (2) MCDUs detection: Tri-factor importance scoring and curvature-guided adaptive truncation within the MLP of critical layers to identify MCDUs for malicious smart contract detection.

Linear probes reveal information decodable from intermediate representations, but predictive accuracy alone does not establish its functional relevance (Somvanshi et al., 2026). Neuron-level attribution localizes units associated with factual knowledge within feed-forward networks (FFNs) (Dai et al., 2022). Skill-neuron and sparse-probing studies (Wang et al., 2022; Gurnee et al., 2023) further identify small sets of task-relevant units and assess their roles through targeted interventions. In code security, recent work (Dao et al., 2026) localizes FFN neurons associated with malicious Python package detection and evaluates their effects through neuron interventions. Complementary work on explainable smart contract vulnerability detection provides evidence through expert-defined patterns (Liu et al., 2021), code regions (Wang et al., 2026), and explanatory graph substructures (Yu et al., 2026). However, identifying pretrained LLM components encoding malicious contract logic, controlling for surface cues, and verifying their contribution to detection remain underexplored.

3 METHOD

As shown in Figure 2, ContractLens localizes and extracts latent security knowledge from pretrained LLMs in two steps. First, it evaluates each layer for malicious contract separability, malicious logic sensitivity, and surface perturbation stability, retaining only layers that satisfy all three criteria as critical layers. Second, within the MLP modules of these critical layers, it identifies key neurons through tri-factor importance scoring and curvature-guided adaptive truncation. We refer to these neurons as MCDUs and use them for malicious smart contract detection.

3.1 CRITICAL LAYER LOCALIZATION

For an input smart contract x, we obtain its layer-level representation $\mathbf{h}_l(x)$ by mean-pooling the hidden states at each layer l. We then localize critical layers using the following three criteria.

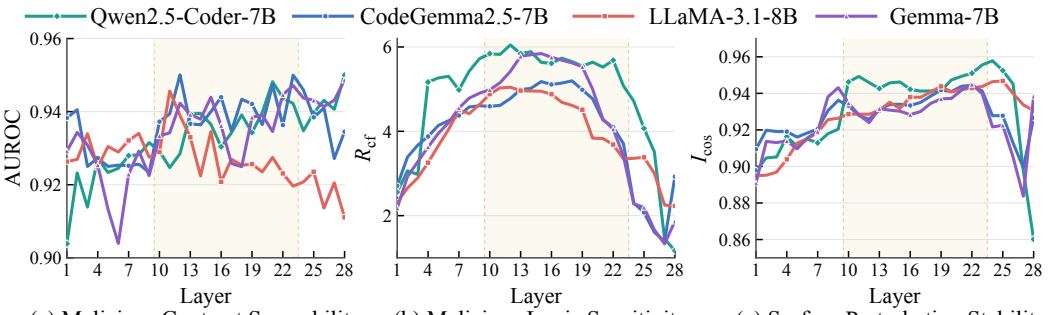
Malicious Contract Separability. We first fit a probe to the representation at each layer:

$$p_l(x) = \sigma\left(\mathbf{w}_l^\top \mathbf{h}_l(x) + b_l\right), \tag{1}$$

where σ is the sigmoid function. The probe parameters $\mathbf{w}_l$ and b_l are optimized while the pretrained model remains frozen. We evaluate each probe on the validation set and denote the AUROC score at layer l by $A^{(l)}$. Given a threshold τ_{sep}, we define the separability candidate set as:

$$\mathcal{L}_{\text{sep}} = \left\{ l \,\middle|\, A^{(l)} \geq \tau_{\text{sep}} \right\}. \tag{2}$$

High probe performance alone does not establish that a layer represents malicious logic. A probe may instead exploit differences in variable names, comments, formatting, contract length, or code templates. We therefore introduce the following two criteria to determine whether the observed separability is specifically associated with malicious behaviour.



(a) Malicious Contract Separability (b) Malicious Logic Sensitivity (c) Surface Perturbation Stability

Figure 3: **Visualization of layer-wise localization criteria across models.** The malicious logic sensitivity $R_{\text{cf}}^{(l)}$ measures the representation change caused by removing malicious logic. The surface perturbation stability $I^{(l)}$ measures representation stability under code edits that preserve contract behaviour. The linear-probe AUROC measures the ability of each layer to distinguish malicious from benign smart contracts.

Malicious Logic Sensitivity. For each malicious contract x, we construct a counterfactual variant x_{cf} by removing the code responsible for the malicious behaviour while preserving the remaining code as closely as possible. We measure the resulting representation change at layer l as:

$$d_{\text{cf}}^{(l)}(x) = \left\| \frac{\mathbf{h}_l(x)}{\|\mathbf{h}_l(x)\|_2} - \frac{\mathbf{h}_l(x_{\text{cf}})}{\|\mathbf{h}_l(x_{\text{cf}})\|_2} \right\|_2 . \tag{3}$$

Because removing code also changes the input structure, $d_{\text{cf}}^{(l)}$ may partly reflect the edit itself. To reduce this effect, we compare it with representation changes caused by random non-semantic code modifications of a similar size. The relative sensitivity to malicious logic removal is defined as:

$$R_{\text{cf}}^{(l)} = \frac{\mathbb{E}_x \left[d_{\text{cf}}^{(l)}(x) \right]}{\mathbb{E}_{x,\widetilde{x}} \left[d_{\text{ctrl}}^{(l)}(x, \widetilde{x}) \right] + \epsilon}, \tag{4}$$

where $\widetilde{x}$ is generated by randomly applying a code modification of similar size, and ϵ prevents division by zero. We additionally measure the decrease in the predicted malicious probability:

$$\Delta p^{(l)} = \mathbb{E}_x \left[p_l(x) - p_l(x_{\text{cf}}) \right].$$

We define the sensitivity candidate set as:

$$\mathcal{L}_{\text{sens}} = \left\{ l \,\middle|\, R_{\text{cf}}^{(l)} \geq \tau_R,\ \Delta p^{(l)} \geq \tau_p \right\}, \tag{5}$$

where τ_R and τ_p denote the selection thresholds for relative representation change and malicious probability reduction, respectively.

Surface Perturbation Stability. We apply a transformation T, such as comment removal, formatting normalization, pragma rewriting, variable renaming, or function reordering, to each smart contract x without changing its program behaviour, obtaining $x_T = T(x)$. Let $\boldsymbol{\mu}_l$ denote the mean representation of all training contracts at layer l. We quantify surface perturbation stability using representation similarity $I_{\text{cos}}^{(l)}$, prediction consistency $C^{(l)}$, and probability shift $S_p^{(l)}$:

$$I_{\text{cos}}^{(l)} = \mathbb{E}\left[\cos\left(\overline{\mathbf{h}}_l(x), \overline{\mathbf{h}}_l(x_T)\right)\right], \quad C^{(l)} = \mathbb{E}[\mathbb{I}[\widehat{y}_l(x) = \widehat{y}_l(x_T)]], \quad S_p^{(l)} = \mathbb{E}[|p_l(x) - p_l(x_T)|]. \tag{6}$$

The stability candidate set is defined as:

$$\mathcal{L}_{\text{stab}} = \left\{ l \,\middle|\, I_{\text{cos}}^{(l)} \geq \tau_I,\ C^{(l)} \geq \tau_C,\ S_p^{(l)} \leq \tau_S \right\}, \tag{7}$$

where τ_I and τ_C are the minimum similarity and consistency, and τ_S is the maximum probability shift. Finally, we define the critical layers as the intersection of the three candidate sets:

$$\mathcal{L}_{\text{critical}} = \mathcal{L}_{\text{sep}} \cap \mathcal{L}_{\text{sens}} \cap \mathcal{L}_{\text{stab}}. \tag{8}$$

As shown in Figure 3, the intermediate layers simultaneously exhibit high malicious contract separability, strong sensitivity to malicious logic removal, and high stability under surface modifications across all four models. Detailed examples of how we construct the malicious logic removal and surface-preserving variants are provided in Appendix D. The theoretical analysis of critical layer localization and MCDUs selection is provided in Appendix A. This consistent pattern indicates that latent security knowledge for malicious smart contract detection is primarily encoded in these layers.

3.2 Malicious Contract-Discriminative Unit Detection

After identifying the critical layers, ContractLens further localizes the MLP units that encode latent security knowledge. For each layer $l \in \mathcal{L}_{\text{critical}}$, we average-pool the non-padding token activations to obtain a contract-level MLP activation vector $\mathbf{a}_l(x) \in \mathbb{R}^{d_{\text{MLP}}}$, where d_{MLP} denotes the MLP width. We then fit a linear probe to the MLP activations:

$$q_l(x) = \sigma\left(\mathbf{v}_l^\top \mathbf{a}_l(x) + c_l\right), \tag{9}$$

where $\mathbf{v}_l \in \mathbb{R}^{d_{\text{MLP}}}$ and $c_l \in \mathbb{R}$ denote the probe weight vector and bias, respectively.

Tri-Factor Neuron Scoring. Probe weights alone may not reliably identify neurons associated with malicious logic. We therefore evaluate each neuron using three complementary signals: activation contrast, probe weight, and malicious-logic specificity. For neuron k in layer l, we define the activation contrast as:

$$\overline{a}_{l,k}^{\text{contrast}} = \mathbb{E}_{x\sim\mathcal{D}_{\text{mal}}}\left[a_{l,k}(x)\right] - \mathbb{E}_{x\sim\mathcal{D}_{\text{ben}}}\left[a_{l,k}(x)\right], \tag{10}$$

where $\mathcal{D}_{\text{mal}}$ and $\mathcal{D}_{\text{ben}}$ denote the malicious and benign training contracts, respectively. The probe weight $v_{l,k}$ measures the contribution of the neuron to the linear decision boundary. To further distinguish neurons associated with malicious semantics from those sensitive to superficial code variations, we define the malicious-logic specificity as:

$$r_{l,k} = \frac{\mathbb{E}_{x\sim\mathcal{D}_{\text{mal}}}\left[\left|a_{l,k}(x) - a_{l,k}(x^{\text{cf}})\right|\right]}{\mathbb{E}_{x,T}\left[\left|a_{l,k}(x) - a_{l,k}(x^{T})\right|\right] + \epsilon}, \tag{11}$$

where $\epsilon > 0$ prevents division by zero. A larger $r_{l,k}$ indicates that neuron k responds more strongly to changes in malicious semantics than to superficial code variations. The three signals are combined into the following importance score:

$$S_{l,k} = \left|\overline{a}_{l,k}^{\text{contrast}} \cdot v_{l,k}\right| \cdot r_{l,k}. \tag{12}$$

This score prioritizes neurons that respond differently to malicious and benign contracts, contribute strongly to the probe decision, and substantially affect the classification loss.

MCDUs Selection. We collect the scores of all candidate neurons from the critical layers and apply global min–max normalization:

$$\widetilde{S}_{l,k} = \frac{S_{l,k} - S_{\min}}{S_{\max} - S_{\min} + \epsilon}, \tag{13}$$

where $S_{\min}$ and $S_{\max}$ are the minimum and maximum scores across all candidate neurons, respectively, and $\epsilon > 0$ prevents division by zero. We sort the normalized scores in descending order and apply Kneedle (Satopaa et al., 2011) to the resulting importance curve. We retain neurons ranked before the detected knee point $K^\star$ as MCDUs. Unlike a fixed retention threshold, this procedure determines the number of MCDUs from the observed score distribution.

Detection with MCDUs. After identifying the MCDUs, we concatenate their activations across the critical layers and use the resulting feature vector to train a logistic regression classifier for malicious smart contract detection.

4 Experiments

4.1 Implementation Details

Dataset. We use the public dataset (Huang et al., 2025b), which contains real-world smart contracts collected from the Binance Smart Chain (BSC) explorer. After source-code parsing and sample filtering, the dataset contains 32,141 contracts covering ten fine-grained intents: fee, disable trading, blacklist, reflect, max transaction, mint, honeypot, reward, rebase, and max sell. For binary detection, contracts containing at least one target intent are treated as malicious, resulting in 21,166 malicious and 10,975 benign contracts. This code-level label indicates the presence of latent malicious capabilities rather than confirmed on-chain exploitation. Accordingly, 21,166 contracts are labelled as malicious, while the remaining 10,975 contracts are labelled as benign.

For critical-layer localization, we construct a balanced probing set by sampling 1,000 contracts annotated with at least one target intent and 1,000 benign contracts. For fine-grained localization,

Table 1: **Detection performance of malicious intent on three backbone models.** We report AUROC (%, ↑) and accuracy (ACC) (%, ↑). Avg. denotes the macro-average performance across ten malicious intents. The Avg. is computed over all ten intents.

Qwen3-8B

Method	Avg.		Fee		DisableTrading		Honeypot		Reflection		Blacklist		MaxTX		Mint	
	AUROC	ACC	AUROC	ACC	AUROC	ACC	AUROC	ACC	AUROC	ACC	AUROC	ACC	AUROC	ACC	AUROC	ACC
Default	35.17	39.60	36.16	38.50	36.80	41.50	37.12	35.71	34.93	40.50	38.62	42.50	36.52	39.50	26.02	39.00
FTSmartAudit	70.45	67.19	72.07	67.00	76.87	66.50	80.84	75.89	69.93	64.00	77.49	67.00	74.30	70.00	80.51	78.00
CodeXplain	89.75	80.90	85.45	75.50	88.87	80.50	91.29	87.50	90.37	83.00	88.23	76.50	86.50	69.00	86.67	71.00
DBESN	88.49	82.42	87.64	78.50	88.44	78.00	80.07	72.32	87.81	83.00	90.96	84.00	90.16	82.00	76.91	70.00
SCMDetector	93.49	87.50	90.99	83.50	92.84	84.00	92.38	90.18	92.93	88.00	93.39	85.00	92.93	83.50	87.56	77.00
SmartIntentNN	94.25	87.82	89.86	83.00	93.89	87.00	94.90	90.18	94.51	87.00	94.42	85.50	**94.10**	84.50	85.90	77.50
SmartIntentV2	93.20	86.27	89.79	83.50	90.24	82.50	94.36	83.04	93.67	86.00	93.35	84.50	93.17	82.50	87.67	76.50
Ours	**95.48**	**91.21**	**92.17**	**85.00**	**95.30**	**91.00**	**98.57**	**91.96**	**95.84**	**91.50**	**94.74**	**87.00**	94.09	**90.50**	**88.32**	**83.00**

Qwen2.5-Coder-7B

Method	Avg.		Fee		DisableTrading		Honeypot		Reflection		Blacklist		MaxTX		Mint	
	AUROC	ACC	AUROC	ACC	AUROC	ACC	AUROC	ACC	AUROC	ACC	AUROC	ACC	AUROC	ACC	AUROC	ACC
Default	43.68	50.76	47.84	57.00	36.64	48.50	42.43	47.32	50.24	56.00	38.96	50.00	42.12	53.50	45.76	48.00
FTSmartAudit	60.09	57.80	63.59	56.50	72.74	64.00	65.21	62.50	66.68	55.50	67.78	62.50	73.92	66.50	56.37	58.00
CodeXplain	89.46	81.82	85.57	80.00	89.02	81.50	81.92	79.46	92.00	84.00	87.29	79.00	89.10	79.50	84.91	74.50
DBESN	95.81	91.90	92.20	81.50	95.42	**92.00**	94.42	89.29	**97.01**	**93.50**	93.99	89.00	**96.31**	**93.50**	**90.71**	**85.50**
SCMDetector	95.67	91.61	91.70	84.50	95.35	91.00	93.43	91.07	96.84	89.00	95.42	89.50	96.07	92.50	90.36	84.50
SmartIntentNN	95.50	91.56	93.40	86.50	95.53	91.50	**94.90**	**91.96**	96.57	93.00	94.12	88.00	93.08	88.50	90.11	85.00
SmartIntentV2	95.50	91.56	93.40	86.50	95.53	91.50	**94.90**	**91.96**	96.57	93.00	94.12	88.00	93.08	88.50	90.11	85.00
Ours	**96.24**	**92.41**	**94.85**	**87.50**	**96.32**	**92.00**	94.77	**91.96**	96.83	92.00	**96.65**	**92.00**	95.88	91.50	88.89	81.50

CodeGemma2.5-7B

Method	Avg.		Fee		DisableTrading		Honeypot		Reflection		Blacklist		MaxTX		Mint	
	AUROC	ACC	AUROC	ACC	AUROC	ACC	AUROC	ACC	AUROC	ACC	AUROC	ACC	AUROC	ACC	AUROC	ACC
Default	39.42	46.47	42.34	46.00	48.54	48.50	31.55	41.96	36.75	54.00	45.59	48.00	39.59	46.50	38.46	44.00
FTSmartAudit	66.82	62.14	56.60	51.50	70.03	66.00	60.62	58.93	59.12	57.00	62.16	60.50	70.67	64.00	64.01	60.50
CodeXplain	85.74	78.68	80.19	71.50	82.86	74.50	93.94	89.29	84.66	79.50	83.39	74.00	85.39	77.50	79.39	71.00
DBESN	**95.33**	83.72	92.10	78.00	**95.38**	89.00	94.34	**95.54**	95.75	90.00	94.72	81.50	93.83	86.50	**91.67**	79.00
SCMDetector	94.42	89.49	89.20	**85.00**	93.53	86.50	95.18	92.86	94.20	87.00	**94.78**	**88.50**	92.63	86.50	90.98	81.50
SmartIntentNN	88.11	81.25	86.82	80.00	86.09	77.00	86.28	78.00	84.55	76.00	88.11	81.25	86.82	80.00	86.09	77.00
SmartIntentV2	86.35	78.01	78.50	71.00	86.37	77.50	87.31	81.25	83.62	75.00	82.87	73.50	84.07	76.50	82.30	76.00
Ours	95.24	**90.94**	**92.33**	84.00	94.68	**89.50**	**95.89**	92.86	**96.62**	**93.00**	93.98	88.00	**95.14**	**91.00**	87.63	**84.50**

we identify MCDUs separately for each target intent by formulating ten independent one-vs-benign binary classification tasks. For each task, contracts annotated with the target intent serve as positive samples, while benign contracts serve as negative samples. To prevent data leakage, we group contracts sharing the same creator or source code and assign each group entirely to one dataset partition. For each intent, we split the resulting groups into training, validation, and test sets using a ratio of $0.64 : 0.16 : 0.20$.

Metrics. Following Zheng et al. (2020), we evaluate classification performance using the area under the receiver operating characteristic curve (AUROC) and accuracy (ACC), and the F_1-score.

Baselines. We first assess the ability of different LLMs to detect malicious smart contracts by providing contract code directly in the prompt, and use these results as the default baseline. These include one general-purpose LLM, Qwen3-8B (Yang et al., 2025), and two code-oriented LLMs, Qwen2.5-Coder-7B (Hui et al., 2024) and CodeGemma-2.5-7B (Team et al., 2024). Then, we also compare our method with seven representative smart contract detectors. EDSCVD (Wu et al., 2024) integrates adversarial sample augmentation with dual-channel feature extraction. FTSmartAudit (Wei et al., 2024a) fine-tunes compact LLMs using knowledge distilled from stronger models for smart contract auditing. DBESN (Bedi et al., 2025) uses a dual-branch Siamese network to capture discriminative contract features. SCMDetector (Huang et al., 2025a) combines language-model-based representations with an attention-enhanced BiLSTM (Siami-Namini et al., 2019). SmartIntentNN (Huang et al., 2025b) combines contextual sentence encoding, intent-feature highlighting, and a BiLSTM classifier. SmartIntentV2 (Huang et al., 2026) employs a domain-adapted pretrained model to learn smart contract representations. CodeXplain (Chen et al., 2026) combines LLM-generated code explanations with contract code to train a supervised vulnerability detector.

Hyperparameters. We set the maximum input length to 2,048 tokens and process longer contracts with sliding windows overlapping by 512 tokens. We obtain contract-level representations by mean-pooling the non-padding token embeddings. For the logistic-regression probes, we standardize the

Table 2: **Detection performance on unseen malicious intents using CodeGemma2.5-7B.** Under the leave-one-intent-out setting, we report ACC (%, ↑) and F_1-score(%, ↑) across five held-out malicious intents, where each intent is excluded from training and used only for testing in its corresponding experiment.

Method	Fee		MaxSell		Reflection		Reward		Rebase		Avg.	
	ACC	F1	ACC	F1	ACC	F1	ACC	F1	ACC	F1	ACC	F1
DBESN	58.33	37.50	53.85	33.33	60.53	55.09	67.42	60.81	52.33	22.64	58.49	41.88
EDSCVD	59.38	36.07	46.15	36.36	63.68	51.75	61.80	53.42	67.44	65.85	59.69	48.69
SCMDetector	60.94	43.61	73.08	74.07	51.05	7.92	67.42	63.29	77.91	80.00	66.08	53.78
SmartIntentV2	61.98	66.36	57.69	62.07	55.26	50.87	61.24	63.87	61.63	70.27	59.56	62.69
Ours	**67.19**	**70.97**	**84.62**	**84.62**	**65.26**	**59.76**	**70.22**	**69.71**	**80.23**	**82.47**	**73.50**	**73.51**

input features and search C over $\{0.001, 0.01, 0.1, 1, 10\}$. The main comparison experiments use $C = 1.0$. Critical layer selection uses $\delta_{\text{AUROC}} = 0.008$, $T_R = 1.50$, and $T_I = 0.925$. The counterfactual probability-drop threshold is 0.02. The prediction-consistency threshold and maximum probability shift under surface perturbations are 0.95 and 0.04, respectively. All confidence estimates use $1{,}500$ bootstrap resamples. The detailed search ranges and selection experiments for the key hyperparameters are provided in Appendix B. All probing experiments and lightweight classifier training tasks are conducted on a single NVIDIA Tesla V100 GPU with 32 GB of memory.

4.2 Main Results

Quantitative Results. We evaluate ContractLens using one general-purpose LLM, Qwen3-8B (Yang et al., 2025), and two code-oriented LLMs, Qwen2.5-Coder-7B (Hui et al., 2024) and CodeGemma-2.5-7B (Team et al., 2024). Table 1 reports the detection performance across malicious intent categories, with the Avg. representing the mean over all ten categories. Our method achieves the highest average accuracy on all three backbone models and the highest average AUROC on Qwen3-8B and Qwen2.5-Coder-7B. On Qwen3-8B, our method achieves an average AUROC of 95.48% and an average accuracy of 91.21%. On Qwen2.5-Coder-7B, it achieves an average AUROC of 96.24% and an average accuracy of 92.41%. On CodeGemma-2.5-7B, our method obtains an average AUROC of 95.24% and an average accuracy of 90.94%. Together, these results show that MCDUs encode security knowledge that supports detecting diverse malicious intent in smart contracts. A detailed analysis of efficiency and computational cost is provided in Appendix C. A real-world malicious smart contract detection case is presented in Appendix E.

Evaluation on Unseen Malicious Intents. To evaluate generalization to unseen malicious behaviours, we conduct a leave-one-intent-out experiment. For each held-out intent, all localization and training stages use only the remaining malicious intents and benign contracts, while the held-out intent is used exclusively for testing. We hold out one malicious intent for testing and train the detector on the remaining intents, without fine-tuning on the held-out samples. Multi-label malicious smart contracts may serve as positive test samples, but each smart contract is assigned to only one held-out intent to prevent reuse across test sets. Each test set contains equal numbers of benign and malicious contracts. Table 2 reports the results on CodeGemma2.5-7B across five held-out intents. Our method achieves the highest accuracy and F_1-score for every intent, with average scores of 73.50% and 73.51%, respectively. Compared with the strongest baselines, our method improves the average accuracy over SCMDetector by 7.42% and the average F_1-score over SmartIntentV2 by 10.82%. These results show that the neuron-level discriminative signals extracted support transferable representations and robust generalization to unseen malicious intents.

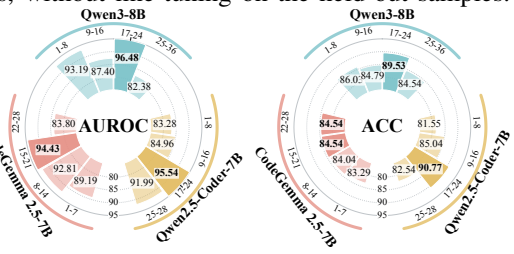


Figure 4: **Ablation study on layer depth intervals.** We partition each model into four contiguous layer intervals. The bar charts compare accuracy and AUROC across three LLMs.

4.3 Ablation Study

Layers Validation. To test whether intent-discriminative is concentrated within specific depth ranges, we conduct layer-interval ablations across three backbones. We partition the Transformer layers of each model into four contiguous depth intervals: shallow, transition, critical, and deep.

Then, we construct the detection features exclusively from one layer interval, while keeping the classifier architecture, training data, and hyperparameters unchanged. As shown in Figure 4, the critical intervals produce the strongest overall performance across the three backbones. On CodeGemma-2.5-7B, AUROC increases from 89.19% in the shallow interval to 94.43% in the critical interval, before decreasing to 83.80% in the deep interval. The metric of accuracy increases from 83.29% to 84.54%, although the deep interval achieves the same accuracy. On Qwen3-8B, the critical interval reaches 96.48% AUROC and 89.53% accuracy, compared with 93.19% and 86.02% in the shallow interval. On Qwen2.5-Coder-7B, the critical interval similarly achieves the highest AUROC and accuracy, reaching 95.54% and 90.77%, respectively. These consistent intermediate-layer peaks support our viewpoint that latent security knowledge associated with malicious smart contract intent is concentrated within specific depth intervals rather than uniformly distributed across the network.

Neurons Validation. To determine whether detection performance depends on the identified neurons rather than arbitrary neurons within the critical layers, we compare MCDUs with two matched random baselines. Each random baseline is evaluated over 30 independent repetitions. As shown in Figure 5, MCDUs consistently achieve the highest test AUROC across all three models. Specifically, MCDUs obtain 94.4%, 96.4%, and 94.9% on CodeGemma-2.5-7B, Qwen3-8B, and Qwen2.5-Coder-7B, respectively. Random-In achieves 88.0%, 93.1%, and 92.8%, whereas Random-Ex achieves 88.6%, 93.3%, and 92.2%. These consistent performance gaps show that randomly selecting neurons from the critical layers cannot reproduce the effectiveness of MCDUs. Instead, the results indicate that malicious intent-related latent security knowledge is concentrated in the sparse neurons identified by our method.

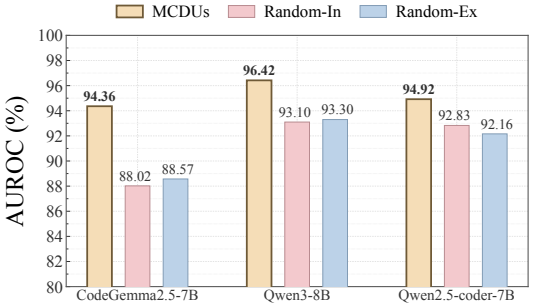


Figure 5: **Validation of neuron specificity.** We compare the AUROC of the identified MCDUs with two random selection baselines: Random-In, which randomly selects neurons from the same critical layers, and Random-Ex, which randomly selects neurons while excluding the identified MCDUs.

Functional Specificity & Robustness. Table 3 examines whether the identified MCDUs make a specific functional contribution to malicious smart contract detection. The default setting trains a linear probe on the complete MLP activations from the critical layers while keeping the backbone frozen. "Masked U0" masks the same number of randomly selected neurons from the critical layers. "Masked U1" masks non-MCDUs with similar activation magnitudes to control for activation strength. The MCDUs directly evaluate the contribution of the neurons identified by our method. The experiment results show that "Masked MCDUs" reduce accuracy and AUROC by 4.49% and 2.55% on CodeGemma-2.5-7B, and by 8.48% and 0.85% on Qwen3-8B. In contrast, "Masked U0" causes no performance degradation on either model. Under "Masked U1", accuracy is unchanged on CodeGemma-2.5-7B and decreases by only 0.25% on Qwen3-8B, while AUROC increases slightly on both models. These controlled comparisons show that neither the number and layer locations of the masked neurons nor their activation magnitudes explain the observed degradation. The larger performance drops caused by masking MCDUs support their functional specificity across both models.

Table 3: **Functional ablation of MCDUs.** U0 denotes randomly selected neurons, and U1 denotes non-MCDUs with similar activation magnitudes. Lower performance after masking indicates a greater contribution to our smart contract detection task.

Method	ACC (↑)	AUROC (↑)
CodeGemma-2.5-7B	85.54	89.26
+ Maskede U0	85.58 (+0.04)	89.34 (+0.08)
+ Maskede U1	85.54 (0.00)	89.64 (+0.38)
+ Maskede MCDUs	**81.05** (-4.49)	**86.71** (-2.55)
Qwen3-8B	91.52	95.02
+ Maskede U0	91.52 (0.00)	95.05 (+0.03)
+ Maskede U1	91.27 (-0.25)	95.05 (+0.03)
+ Maskede MCDUs	**83.04** (-8.48)	**94.17** (-0.85)

4.4 More Results

Robustness to Training-Data Size. Figure 6 evaluates the data efficiency of our method under limited supervision. We freeze the CodeGemma-2.5-7B backbone, retain the same identified MCDUs, and train only a logistic-regression probe. Each training set contains equal numbers of benign and malicious contracts. With only 32 labelled contracts, our method achieves an average AUROC

Table 4: **Detection performance on newly deployed malicious smart contracts.** We compare ContractLens with baseline methods on malicious smart contracts deployed after the release of Gemma-7B. We report accuracy and AUROC (%, ↑) for three malicious intents and their averages.

Method	Honeypot		Black/Whitelist		Transfer Restriction		Avg.	
	ACC	AUROC	ACC	AUROC	ACC	AUROC	ACC	AUROC
SmartIntentNN	43.75	56.25	40.00	32.00	62.50	76.56	48.75	54.94
EDSCVD	50.00	34.38	**80.00**	72.00	50.00	82.81	60.00	63.06
DBESN	50.00	51.56	50.00	**100.00**	50.00	**90.63**	50.00	80.73
Ours	**81.25**	**95.31**	70.00	68.00	**81.25**	89.06	**77.50**	**84.13**

of 85.03% across the seven malicious intents. The average AUROC increases to 88.80%, 91.89%, 94.09%, and 94.97% as the training-set size increases, yielding an overall gain of 9.94%. The improvement decreases beyond 256 labelled smart contracts, indicating that performance gradually approaches saturation. The effect of additional supervision also varies across malicious intents. The AUROC increases from 78.70% to 98.35% for Reward and from 85.51% to 96.04% for Reflection. In contrast, Blacklist and MaxTX already achieve approximately 95% AUROC with limited supervision and remain stable as more labelled data become available. These results show that the discriminative signals extracted from MCDUs can be effectively decoded using limited labelled data, supporting the few-shot data efficiency of ContractLens without updating the backbone.

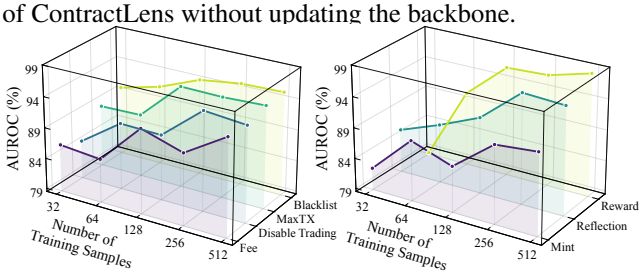


Figure 6: **Evaluation under different volumes of training data.** We report the AUROC of our method across seven malicious intents under few-shot settings.

Detection of Newly Deployed Malicious Smart Contracts. Malicious smart contracts continually evolve, requiring detectors to recognize newly deployed smart contracts and emerging attack patterns. To evaluate this capability, we conduct a temporal evaluation using a dataset collected from Dexscreener (Sun et al., 2026; Dexscreener, 2025). The dataset covers 14,671 BSC token addresses collected between March 18, 2023 and June 15, 2025, together with smart contract and transaction records retrieved through BscScan. After deduplication and activity filtering, it contains 1,495 representative tokens, from which the original study selects 1,201 labelled tokens, including 898 scam and 303 benign tokens. Because the collection period spans both before and after the release of Gemma-7B (Gemma Team et al., 2024), we filter the dataset by smart contract deployment time. Specifically, all smart contracts used in this evaluation are deployed after the model's public release. We focus on three malicious mechanisms: honeypots, blacklist/whitelist controls, and transfer restrictions, which restrict token selling, address eligibility, and token transfers, respectively. As shown in Table 4, our method consistently outperforms the comparison methods overall, achieving the highest average accuracy of 77.50% and AUROC of 84.13%. These results exceed the strongest competing results by 17.50% in accuracy and 3.40% in AUROC. Our method performs particularly well on Honeypot, reaching an accuracy of 81.25% and an AUROC of 95.31%. It also achieves the highest accuracy of 81.25% on Transfer Restriction. Overall, the results support that our method can detect newly deployed malicious smart contracts and adapt to evolving malicious behaviours without updating the backbone model.

5 Conclusion

This work investigates whether pretrained LLMs contain localized latent security knowledge that supports malicious smart contract detection without adapting the backbone. Our method localizes this knowledge at both the layer and neuron levels and uses the activations of the identified MCDUs as compact features for a lightweight classifier. Across three pretrained models, our method achieves higher average detection accuracy than all evaluated methods while keeping the backbone frozen. Masking MCDUs causes larger performance drops than masking random or activation-matched neurons, supporting their functional specificity. Our method also generalizes to unseen malicious intents and maintains strong detection performance with limited labelled data. Together, these results demonstrate that latent security knowledge in pretrained LLMs can be systematically localized and utilized for interpretable and data-efficient malicious smart contract detection.

AI DISCLOSURE

In this work, we used generative AI tools to edit and polish the manuscript, including improving its grammar, clarity, and readability. We also used generative AI tools to assist with code development. All AI-assisted textual revisions were reviewed by the authors to ensure that they preserved the intended meaning and accurately reflected the research methods and results. AI-assisted code was reviewed and verified for correctness by multiple authors. We take full responsibility for the final content of this work.

ETHICS STATEMENT

This work uses publicly available datasets and open-weight models and does not collect private or sensitive personal information. We acknowledge potential biases in the source datasets, evaluated models, and automated evaluators.

REPRODUCIBILITY STATEMENT

The main text specifies the layer and MCDUs selection procedures, data split, and detection evaluations. Appendix A provides the assumptions and proofs for the theoretical analysis; Appendices B and C report hyperparameter sensitivity and computational costs; and Appendices E and F document code transformations and case studies.

REFERENCES

Syed Badruddoja, Ram Dantu, Yanyan He, Kritagya Upadhayay, and Mark Thompson. Making smart contracts smarter. In *2021 IEEE international conference on blockchain and cryptocurrency (ICBC)*, pp. 1–3. IEEE, 2021.

Punam Bedi, Vinita Jindal, Ningyao Ningshen, and Pushkar Gole. Dbesn: A novel model for detecting and identifying malicious code in a smart contract. *Blockchain: Research and Applications*, pp. 100304, 2025.

Leonard Bereska and Efstratios Gavves. Mechanistic interpretability for ai safety–a review. *arXiv preprint arXiv:2404.14082*, 2024.

Yan Chen and Cristiano Bellavitis. Blockchain disruption and decentralized finance: The rise of decentralized business models. *Journal of business venturing insights*, 13:e00151, 2020.

Yizhou Chen, Zeyu Sun, Guoqing Wang, Qingyuan Liang, Xiao Yu, and Dan Hao. From cryptic to clear—training on llm explanations to detect smart contract vulnerabilities. *ACM Transactions on Software Engineering and Methodology*, 35(6):1–24, 2026.

Damai Dai, Li Dong, Yaru Hao, Zhifang Sui, Baobao Chang, and Furu Wei. Knowledge neurons in pretrained transformers. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8493–8502, 2022.

Lam D Dao, Vang T Nguyen, Anh MT Bui, and Phuong T Nguyen. Which neurons detect malicious code? a probing study of llm security knowledge. *arXiv preprint arXiv:2607.10221*, 2026.

Dexscreener. Dexscreener: Real-time dex market data. `https://dexscreener.com`, 2025. Accessed: 2025-06-30.

Hao Ding, Yizhou Liu, Xuefeng Piao, Huihui Song, and Zhenzhou Ji. Smartguard: An llm-enhanced framework for smart contract vulnerability detection. *Expert Systems with Applications*, 269: 126479, 2025.

Josselin Feist, Gustavo Grieco, and Alex Groce. Slither: a static analysis framework for smart contracts. In *2019 IEEE/ACM 2nd international workshop on emerging trends in software engineering for blockchain (WETSEB)*, pp. 8–15. IEEE, 2019.

Jiaqi Gao, Zijian Zhang, Yuqiang Sun, Ye Liu, Chengwei Liu, Han Liu, Yi Li, and Yang Liu. LogicScan: An LLM-driven framework for detecting business logic vulnerabilities in smart contracts, 2026. URL https://arxiv.org/abs/2602.03271.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology, 2024. URL https://arxiv.org/abs/2403.08295.

Wes Gurnee, Neel Nanda, Matthew Pauly, Katherine Harvey, Dmitrii Troitskii, and Dimitris Bertsimas. Finding neurons in a haystack: Case studies with sparse probing. *arXiv preprint arXiv:2305.01610*, 2023.

Jingyu Huang, Xiaorui Gong, and Xiu Zhang. Scmdetector: Smart contract malicious detection technique based on glm and ablstm-a. In *2025 28th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pp. 490–495. IEEE, 2025a.

Youwei Huang, Sen Fang, Jianwen Li, Bin Hu, Jiachun Tao, and Tao Zhang. Deep smart contract intent detection. In *2025 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 124–135. IEEE, 2025b.

Youwei Huang, Jianwen Li, Sen Fang, Yao Li, Peng Yang, Bin Hu, and Tao Zhang. Smart contract intent detection with pre-trained programming language model. *arXiv e-prints*, pp. arXiv–2508, 2025c.

Youwei Huang, Jianwen Li, Bin Hu, Sen Fang, Yao Li, and Peng Yang. Detecting malicious intents in smart contracts with pre-trained programming language models. In *22nd International Conference on Predictive Models and Data Analytics in Software Engineering*, pp. 41–50, 2026.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

Wanqing Jie, Wangjie Qiu, Haofu Yang, Muyuan Guo, Xinpeng Huang, Tianyu Lei, Qinnan Zhang, Hongwei Zheng, and Zhiming Zheng. Agent4vul: multimodal llm agents for smart contract vulnerability detection. *Science China Information Sciences*, 68(6):160101, 2025.

Zhenguang Liu, Peng Qian, Xiang Wang, Lei Zhu, Qinming He, and Shouling Ji. Smart contract vulnerability detection: from pure neural network to interpretable graph feature and expert pattern fusion. *arXiv preprint arXiv:2106.09282*, 2021.

Shuyi Miao, Wangjie Qiu, Hongwei Zheng, Qinnan Zhang, Xiaofan Tu, Xunan Liu, Yang Liu, Jin Dong, and Zhiming Zheng. Know your account: Double graph inference-based account deanonymization on ethereum. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*, pp. 1305–1318. IEEE, 2025.

Shuyi Miao, Wangjie Qiu, Xiaofan Tu, Yunze Li, Yongxin Wen, and Zhiming Zheng. Tracing your account: A gradient-aware dynamic window graph framework for ethereum under privacy-preserving services. *IEEE Transactions on Information Forensics and Security*, 2026.

Ville Satopaa, Jeannie Albrecht, David Irwin, and Barath Raghavan. Finding a" kneedle" in a haystack: Detecting knee points in system behavior. In *2011 31st international conference on distributed computing systems workshops*, pp. 166–171. IEEE, 2011.

Sima Siami-Namini, Neda Tavakoli, and Akbar Siami Namin. The performance of lstm and bilstm in forecasting time series. In *2019 IEEE International conference on big data (Big Data)*, pp. 3285–3292. IEEE, 2019.

Shriyank Somvanshi, Md Monzurul Islam, Amir Rafe, Anannya Ghosh Tusti, Arka Chakraborty, Anika Baitullah, Tausif Islam Chowdhury, Nawaf Alnawmasi, Anandi Dutta, and Subasish Das. Bridging the black box: a survey on mechanistic interpretability in ai. *ACM Computing Surveys*, 58(8):1–35, 2026.

Litong Sun, YangTian Mi, Xiapu Luo, and Weigang Wu. Octopusguard: K-line enhanced token scam detector powered by multimodal llms. In *Proceedings of the 2026 IEEE/ACM 48th International Conference on Software Engineering*, pp. 2071–2083, 2026.

Yuqiang Sun, Daoyuan Wu, Yue Xue, Han Liu, Wei Ma, Lyuye Zhang, Yang Liu, and Yingjiu Li. Llm4vuln: A unified evaluation framework for decoupling and enhancing llms' vulnerability reasoning. *arXiv preprint arXiv:2401.16185*, 2024.

CodeGemma Team, Heri Zhao, Jeffrey Hui, Joshua Howland, Nam Nguyen, Siqi Zuo, Andrea Hu, Christopher A Choquette-Choo, Jingyue Shen, Joe Kelley, et al. Codegemma: Open code models based on gemma. *arXiv preprint arXiv:2406.11409*, 2024.

Petar Tsankov, Andrei Dan, Dana Drachsler-Cohen, Arthur Gervais, Florian Buenzli, and Martin Vechev. Securify: Practical security analysis of smart contracts. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pp. 67–82, 2018.

Bin Wang, Shan Li, Xiaohan Yuan, Xueshuo Xie, Junyong Wang, Tao Li, Bin Wang, and Wei Wang. Contractscanner: Detecting and localizing vulnerabilities of smart contracts via graph-based semantic modeling of source code. *IEEE Transactions on Network Science and Engineering*, 13: 4093–4106, 2026. doi: 10.1109/TNSE.2025.3583926.

Xiaozhi Wang, Kaiyue Wen, Zhengyan Zhang, Lei Hou, Zhiyuan Liu, and Juanzi Li. Finding skill neurons in pre-trained transformer-based language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11132–11152, 2022.

Zhiyuan Wei, Jing Sun, Zijian Zhang, Xianhao Zhang, and Zhe Hou. Ftsmartaudit: A knowledge distillation-enhanced framework for automated smart contract auditing using fine-tuned llms. *arXiv preprint arXiv:2410.13918*, 2024a.

Zhiyuan Wei, Jing Sun, Zijiang Zhang, Xianhao Zhang, Meng Li, and Zhe Hou. Llm-smartaudit: Advanced smart contract vulnerability detection. *arXiv preprint arXiv:2410.09381*, 2024b.

Zhiyuan Wei, Jing Sun, Yuqiang Sun, Ye Liu, Daoyuan Wu, Zijian Zhang, Xianhao Zhang, Meng Li, Yang Liu, Chunmiao Li, Mingchao Wan, Jin Dong, and Liehuang Zhu. Advanced smart contract vulnerability detection via llm-powered multi-agent systems. *IEEE Transactions on Software Engineering*, 51(10):2830–2846, 2025. doi: 10.1109/TSE.2025.3597319.

Zhiyuan Wei, Jing Sun, Zijian Zhang, Zhe Hou, Xianhao Zhang, Meng Li, Yuqiang Sun, Daoyuan Wu, Yang Liu, Chunmiao Li, et al. Hkt-smartaudit: Distilling lightweight models for smart contract auditing. *IEEE Transactions on Information Forensics and Security*, 2026.

Huaiguang Wu, Yibo Peng, Yaqiong He, and Siqi Lu. Edscvd: Enhanced dual-channel smart contract vulnerability detection method. *Symmetry*, 16(10):1381, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Yiming Yu, Yunfei Guo, Junchen Liu, Yiping Sun, and Junliang Du. Interpretable smart contract vulnerability detection with llm-augmented hilbert–schmidt information bottleneck. *Computers, Materials & Continua*, 87(2):28, 2026. doi: 10.32604/cmc.2025.074959.

Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Weili Chen, Xiangping Chen, Jian Weng, and Muhammad Imran. An overview on smart contracts: Challenges, advances and platforms. *Future Generation Computer Systems*, 105:475–491, 2020.

APPENDIX

The appendices provide additional theoretical and empirical details that support and extend the main paper. Appendix A presents a theoretical analysis of critical-layer localization and MCDUs selection, including the semantic selectivity of the identified layers and the discriminative contribution of MCDUs. Appendix B reports the implementation hyperparameters and evaluates their sensitivity. Appendix C analyzes detection performance and per smart contract processing costs across different backbone models. Appendix D provides a case study illustrating the differences among the original smart contract, malicious logic removal, and surface only rewriting. Appendix E presents a detailed analysis of a real smart contract exhibiting honeypot and mint behaviours. Finally, Appendix F provides further discussion of the method and its scope.

A THEORETICAL ANALYSIS

This section provides a theoretical analysis of critical layer localization and MCDUs selection. We first analyze how the criteria for malicious logic sensitivity and surface perturbation stability help identify layers that respond to malicious logic rather than superficial code changes. We then decompose the contribution of individual MLP units to the probe's class separation and derive an upper bound on the loss change caused by masking selected units. The assumptions and proofs for these results are provided below.

A.1 SEMANTIC SELECTIVITY OF CRITICAL-LAYER LOCALIZATION

Consider a smart contract x as being determined by its behavioural semantics m, including whether malicious logic is present, and surface factors u, including identifier names, comments, formatting, and code organization. Let the normalized representation at layer l be

$$\boldsymbol{\phi}_l(x) = \frac{\mathbf{h}_l(x)}{\|\mathbf{h}_l(x)\|_2} = \mathbf{s}_l(m) + \mathbf{n}_l(u) + \boldsymbol{\varepsilon}_l, \tag{14}$$

where $\mathbf{s}_l$ and $\mathbf{n}_l$ denote the semantic and surface components, respectively, and $\boldsymbol{\varepsilon}_l$ collects residual interactions that are not captured by this additive approximation.

The counterfactual x_{cf} replaces the malicious semantics m with a benign counterpart m_0 while preserving u as closely as possible. The matched control $\widetilde{x}$ preserves m and changes u to $\widetilde{u}$. Define the latent semantic and surface responses as

$$D^{(l)}_{\mathrm{sem}} = \mathbb{E}\|\mathbf{s}_l(m) - \mathbf{s}_l(m_0)\|_2, \qquad D^{(l)}_{\mathrm{surf}} = \mathbb{E}\|\mathbf{n}_l(u) - \mathbf{n}_l(\widetilde{u})\|_2. \tag{15}$$

Let $N_l = \mathbb{E}[d^{(l)}_{\mathrm{cf}}(x)]$ and $B_l = \mathbb{E}[d^{(l)}_{\mathrm{ctrl}}(x, \widetilde{x})]$ denote the observed counterfactual and matched-control changes used in Equation 4.

Theorem 1 (lower bound on semantic selectivity). Assume that the additive decomposition in Equation 14 holds locally and that the expected residual changes are bounded by $\eta^{(l)}_{\mathrm{cf}}$ and $\eta^{(l)}_{\mathrm{ctrl}}$ for the counterfactual and control transformations, respectively. If layer l satisfies $R^{(l)}_{\mathrm{cf}} = N_l/(B_l + \epsilon) \geq \tau_R$, then

$$D^{(l)}_{\mathrm{sem}} - \tau_R D^{(l)}_{\mathrm{surf}} \geq \tau_R \epsilon - \eta^{(l)}_{\mathrm{cf}} - \tau_R \eta^{(l)}_{\mathrm{ctrl}}. \tag{16}$$

In particular, the semantic response is no smaller than the surface response, $D^{(l)}_{\mathrm{sem}} \geq D^{(l)}_{\mathrm{surf}}$, whenever

$$(\tau_R - 1) D^{(l)}_{\mathrm{surf}} + \tau_R \epsilon \geq \eta^{(l)}_{\mathrm{cf}} + \tau_R \eta^{(l)}_{\mathrm{ctrl}}. \tag{17}$$

Proof. By the triangle and reverse-triangle inequalities applied to Equation 14,

$$N_l \leq D^{(l)}_{\mathrm{sem}} + \eta^{(l)}_{\mathrm{cf}}, \qquad B_l \geq D^{(l)}_{\mathrm{surf}} - \eta^{(l)}_{\mathrm{ctrl}}. \tag{18}$$

The selection condition $R^{(l)}_{\mathrm{cf}} \geq \tau_R$ gives

$$\begin{aligned} D^{(l)}_{\mathrm{sem}} + \eta^{(l)}_{\mathrm{cf}} &\geq N_l && (19)\\ &\geq \tau_R(B_l + \epsilon) && (20)\\ &\geq \tau_R\Big(D^{(l)}_{\mathrm{surf}} - \eta^{(l)}_{\mathrm{ctrl}} + \epsilon\Big). && (21) \end{aligned}$$

Rearranging proves Equation 16. □

Theorem 1 gives a quantitative meaning to malicious logic sensitivity: after accounting for residual coupling, a selected layer has a provably larger response to malicious-semantic removal than to a matched surface edit. The remaining two criteria complete the layer-level argument. Membership in $\mathcal{L}_{\text{sep}}$ ensures that the signal is linearly readable, whereas membership in $\mathcal{L}_{\text{stab}}$ requires prediction consistency and bounded probability shift under behaviour-preserving transformations. Thus, the intersection in Equation 8 combines decodability, the semantic-response bound in Theorem 1, and empirical surface invariance.

A.2 Discriminative Contribution and Masking Effect of MCDUs

We next analyze the unit-level probe in Equation 9. Let

$$z_l(x) = \mathbf{v}_l^\top \mathbf{a}_l(x) + c_l \quad \text{and} \quad q_l(x) = \sigma(z_l(x)) \tag{22}$$

denote its logit and predicted malicious probability. Define the class-mean logit gap at layer l as

$$\Gamma_l = \mathbb{E}_{x\sim\mathcal{D}_{\text{mal}}}[z_l(x)] - \mathbb{E}_{x\sim\mathcal{D}_{\text{ben}}}[z_l(x)]. \tag{23}$$

Theorem 2 (unit contribution and masking bound). The class-mean logit gap decomposes exactly over the MLP units:

$$\Gamma_l = \sum_{k=1}^{d_{\text{MLP}}} v_{l,k}\, \overline{a}_{l,k}^{\text{contrast}}. \tag{24}$$

Hence, $|v_{l,k}\overline{a}_{l,k}^{\text{contrast}}|$ is the magnitude of unit k's contribution to the class-mean separation induced by the linear probe. Furthermore, let $\mathcal{F}$ be a set of units that are masked by setting their activations to zero while keeping the probe fixed. The resulting logit change is

$$\Delta z_{l,\mathcal{F}}(x) = z_l(x) - z_l^{\setminus\mathcal{F}}(x) = \sum_{k\in\mathcal{F}} v_{l,k} a_{l,k}(x). \tag{25}$$

Let $\ell_{\text{BCE}}(z, y) = \log(1 + \exp(z)) - yz$ be binary cross-entropy written as a function of the logit. The masking-induced loss change satisfies

$$\left|\ell_{\text{BCE}}(z_l(x), y) - \ell_{\text{BCE}}\left(z_l^{\setminus\mathcal{F}}(x), y\right)\right| \le |\Delta z_{l,\mathcal{F}}(x)| \le \sum_{k\in\mathcal{F}} |v_{l,k} a_{l,k}(x)|. \tag{26}$$

Proof. Substituting Equation 22 into Equation 23 gives

$$\Gamma_l = \sum_k v_{l,k} \left(\mathbb{E}_{\mathcal{D}_{\text{mal}}}[a_{l,k}(x)] - \mathbb{E}_{\mathcal{D}_{\text{ben}}}[a_{l,k}(x)]\right), \tag{27}$$

because the bias c_l cancels between the two class expectations. This is Equation 24. Zeroing the coordinates in $\mathcal{F}$ removes exactly their terms from the logit, which proves Equation 25. Finally,

$$\frac{\partial \ell_{\text{BCE}}(z, y)}{\partial z} = \sigma(z) - y, \qquad |\sigma(z) - y| \le 1. \tag{28}$$

The mean-value theorem therefore gives the first inequality in Equation 26; the triangle inequality gives the second. □

Interpretation of the MCDUs score. Let

$$\Gamma_{l,k} = v_{l,k}\overline{a}_{l,k}^{\text{contrast}} \tag{29}$$

denotes the signed contribution of unit k to the class-mean logit gap. Equation 24 shows that $|\Gamma_{l,k}|$ directly measures the discriminative contribution of the unit under the linear probe.

However, a large discriminative contribution alone does not establish that the unit responds specifically to malicious semantics, since it may also be sensitive to superficial code variations. We therefore further weight this contribution by the malicious-logic specificity $r_{l,k}$ defined in Equation 11. The resulting MCDUs score can be written as

$$S_{l,k} = |\Gamma_{l,k}|\, r_{l,k} = \left|v_{l,k}\overline{a}_{l,k}^{\text{contrast}}\right| r_{l,k}. \tag{30}$$

Table 5: **Hyperparameter sensitivity analysis on CodeGemma-2.5-7B.** Unit Jaccard measures the similarity between the selected neuron set and MCDUs identified by our method under the same setting. Blue cells indicate the final hyperparameter values selected for our model.

Parameter	Value	$\mathcal{L}_{\text{critical}}$	Unit Jaccard	AUROC	Accuracy
δ_{AUROC}	0.005	$\{21\}$	0.25	0.95	**0.90**
	0.008	$\{\mathbf{16}, \mathbf{18}, \mathbf{19}, \mathbf{21}\}$	**1.00**	**0.95**	0.84
	0.010	$\{16, 18, 19, 21\}$	1.00	0.95	0.84
	0.015	$\{10, 14, 15, 16, 18, 19, 20, 21, 22\}$	0.33	0.93	0.84
τ_R	1.25	$\{16, 18, 19, 21\}$	1.00	0.95	0.84
	1.50	$\{\mathbf{16}, \mathbf{18}, \mathbf{19}, \mathbf{21}\}$	**1.00**	**0.95**	**0.84**
	1.75	$\{16, 18, 19, 21\}$	1.00	0.95	0.84
	2.00	$\{16, 18, 19, 21\}$	1.00	0.95	0.84
τ_I	0.920	$\{16, 18, 19, 21\}$	1.00	0.95	0.84
	0.925	$\{\mathbf{16}, \mathbf{18}, \mathbf{19}, \mathbf{21}\}$	**1.00**	**0.95**	**0.84**
	0.930	$\{18, 19, 21\}$	0.74	0.95	0.84
$N_{\text{bootstrap}}$	500	$\{16, 18, 19, 21\}$	1.00	0.95	0.84
	1,500	$\{\mathbf{16}, \mathbf{18}, \mathbf{19}, \mathbf{21}\}$	**1.00**	**0.95**	**0.84**
	3,000	$\{16, 18, 19, 21\}$	1.00	0.95	0.84

Thus, the score favours units that both contribute strongly to class separation and respond preferentially to malicious-logic changes rather than behaviour-preserving surface variations.

The decomposition and masking bound justify the discriminative and functional relevance of the selected units, while the malicious-logic specificity term further constrains their semantic selectivity. Together, these properties motivate the MCDUs scoring mechanism and the masking controls in Table 3. The masking bound does not imply that masking more units must monotonically reduce Accuracy or AUROC, because correlated units may compensate for or cancel one another. The Kneedle cutoff is therefore used only to identify a data-adaptive boundary between the high-score head and the long tail, rather than being treated as a globally optimal subset size.

B Implementation Hyperparameters

Table 5 reports a sensitivity analysis of four key hyperparameters: δ_{AUROC}, τ_R, τ_I, and $N_{\text{bootstrap}}$. We use the Jaccard index to measure the similarity between the neuron set U_θ selected under a given configuration and the set U_{default} selected under the default configuration:

$$J(U_\theta, U_{\text{default}}) = \frac{|U_\theta \cap U_{\text{default}}|}{|U_\theta \cup U_{\text{default}}|}. \tag{31}$$

The value of $J(U_\theta, U_{\text{default}}) = 1$ indicates that the selected neuron set is identical to the MCDUs. Across most evaluated settings, AUROC and accuracy vary only modestly, while the selected neurons remain unchanged for different values of τ_R and $N_{\text{bootstrap}}$. Only larger changes in δ_{AUROC} noticeably affect critical-layer and neuron selection. These results indicate that the localization and detection performance is generally robust to moderate hyperparameter variations.

Based on the validation results, we select $\delta_{\text{AUROC}} = 0.008$, $\tau_R = 1.50$, $\tau_I = 0.925$, and $N_{\text{bootstrap}} = 1{,}500$ as the optimal parameter configuration, which provides the best overall balance between detection performance and localization stability. We set the thresholds for probability drop, prediction consistency, and maximum probability shift to 0.02, 0.95, and 0.04, respectively. We then tune the regularization parameter C of the logistic regression probe on the validation set, selecting $C = 1.0$ from $\{0.001, 0.01, 0.1, 1, 10\}$.

C Efficiency and Computational Cost Analysis

To evaluate practical efficiency, we compare our method with the conventional last-layer probe across four backbone models under the same hardware and evaluation data. We report AUROC,

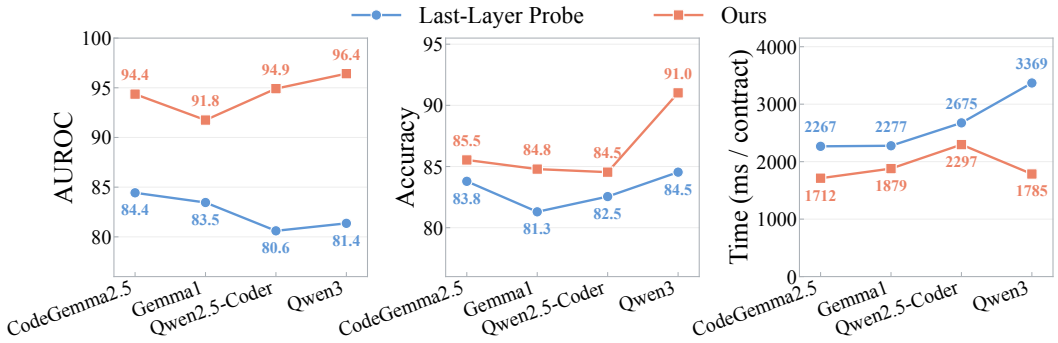


Figure 7: **Detection performance and processing efficiency across backbone models.** We compare ContractLens with the conventional last-layer probe in terms of AUROC, accuracy, and per-contract processing time. Higher AUROC and accuracy indicate better detection performance, while lower processing time indicates greater efficiency.

accuracy, and per-smart contract processing time. As shown in Figure 7, our method consistently improves detection performance across all four backbones. Its AUROC ranges from approximately 91% to 96%, compared with 80% to 85% for the last-layer probe. Our method also achieves higher accuracy on every backbone, with the largest improvement observed on Qwen3-8B. These results indicate that MCDUs provide more discriminative representations than conventional last-layer features. Our method also requires less processing time across all evaluated backbones, with the largest reduction observed on Qwen3-8B. Thus, the performance gains do not introduce additional inference overhead. Instead, the compact representation formed by the selected MCDUs improves both detection effectiveness and processing efficiency. These results support the practical use of our method for large-scale malicious smart contract screening.

D Malicious Logic Removal and Surface Perturbation

Table 6 illustrates the two controlled code transformations used for critical layer localization. The original input x contains an owner-controlled transfer restriction associated with the honeypot intent. Malicious logic removal produces the counterfactual input x_{cf} by removing this restriction, changing the smart contract from malicious to benign. In contrast, a behaviour-preserving surface perturbation produces $x_T = T(x)$ by renaming identifiers without changing the underlying unsafe logic or its label. These transformations separate malicious semantics from superficial code form. A representation associated with malicious logic should change substantially between x and x_{cf}, while remaining stable between x and x_T. This example explains why critical layer localization jointly evaluates malicious logic sensitivity and surface perturbation stability.

E Case Study: A Real Honeypot–Mint Token

Table 7 examines the Lucky Shiba contract, which the BSC dataset labels malicious. Our method correctly identifies the contract as malicious, while several baseline methods miss it. The highlighted code shows why this sample requires more than recognizing individual function names.

In `transferFrom`, an address with `allow[_from]` set to `false` can make a delegated transfer only when `sellOutNum[_from] == 0`. A successful transfer increments this counter, and the contract provides no corresponding reset. Later delegated transfers from the same address may therefore fail while the flag remains false. This can restrict token sales through exchanges, which commonly rely on delegated transfers. The restriction is controlled through `addAllow`: its `onlyOwner` modifier allows the owner to decide which addresses are exempt. The `mint` function also allows the owner to overwrite `balances[miner]` without updating `totalSupply` or emitting a `Transfer` event.

These behaviours become apparent when the transfer condition, persistent counter, owner control, and balance update are considered together. Our method uses activations from selected MLP units in critical layers to form its detection representation, which may help capture this combination of signals. By contrast, examining the functions or keywords in isolation could miss the relationship

Table 6: **Illustration of malicious logic removal and surface perturbation.** The original input contains an unsafe transfer restriction. Malicious logic removal eliminates this restriction and changes the target label from malicious to benign. The behaviour-preserving surface perturbation renames identifiers while retaining the unsafe logic and malicious label. The excerpts are abridged from variants that pass Solidity 0.5.17 compilation.

Original malicious smart contract	

Original input: x	`address private blockedTarget;`
	`function setBlockedTarget(address target) external`
	`onlyOwner {`
	`  blockedTarget = target;`
	`}`
	`// unsafe:  owner selects the blocked target`
	`require(to != blockedTarget, "please wait");`
	Unsafe behaviour: The owner selects an address that is prevented from receiving token transfers.
	(unsafe logic present → malicious/honeypot, label = 1)

Malicious logic removal	

Counterfactual input: x_{cf}	`function setBlockedTarget(address target) external`
	`onlyOwner {`
	`  target = target;`
	`}`
	`// owner-controlled state update removed`
	`require(true, "please wait");`
	`// transfer restriction removed`
	Semantic change: The owner-controlled state update and the blocked-address restriction are removed.
	(unsafe logic removed → benign, label = 0)

Behaviour-preserving surface perturbation	

Perturbed input: $x_T = T(x)$	`address private restrictedAddress;`
	`function setBlockedTarget(address candidate) external`
	`onlyOwner {`
	`  restrictedAddress = candidate;`
	`}`
	`// identifiers renamed only`
	`require(to != restrictedAddress, "please wait");`
	`// unsafe:  transfer restriction unchanged`
	Surface change: The identifiers are renamed while the owner-controlled transfer restriction remains active.
	(unsafe logic retained → malicious/honeypot, label = 1)

between them. This case illustrates how our method detects malicious behaviour distributed across the contract when several baseline methods fail to do so.

F More Discussions

▷ ***Do MCDUs capture latent security knowledge or merely classification correlations?***
High classification performance alone does not establish that the identified units encode malicious logic, since a detector may rely on superficial cues such as identifiers, comments, or code templates. ContractLens therefore introduces semantic controls at both the layer and neuron levels. We require the selected representations to respond to malicious-logic removal while remaining stable under behaviour-preserving surface perturbations. In addition, masking the identified MCDUs causes substantially larger performance degradation than masking randomly selected or activation-matched neurons. Together, these results suggest that MCDUs capture functionally relevant signals associated with malicious contract semantics rather than arbitrary classification correlations.

▷ ***Why are multiple signals required for MCDUs identification?***
Each signal captures a different property of a candidate neuron. Activation contrast measures

Table 7: **Case study of the Lucky Shiba contract.** The excerpt shows the transfer restriction and privileged balance update in a malicious BSC token. Our method detects this sample, while the four baseline methods listed below miss it. Scores represent each method's predicted probability of the malicious MINT label.

Relevant contract logic

```
mapping(address => bool) public allow;
mapping(address => uint256) sellOutNum;
modifier onlyOwner() { require(msg.sender == address(1530756684...602));
_; }
// Privileged access is restricted to a fixed address.
function transferFrom(address _from, address _to, uint256 _value) public {
  ...
  if(!allow[_from]) { require(sellOutNum[_from] == 0); }
// Non exempt addresses must have no prior delegated transfer.
  ...
  sellOutNum[_from]++;
}
// No reset appears in the source, so later transfers may fail.
function addAllow(address holder, bool allowApprove) external onlyOwner {
allow[holder] = allowApprove; }
// The privileged caller can change which holders are exempt.
function mint(address miner, uint256 _value) external onlyOwner {
balances[miner] = _value; }
// Balance overwrite without a supply update or Transfer event.
```

Why it is malicious. When `allow[_from]` is false, `transferFrom` requires `sellOutNum[_from] == 0` and increments the counter after a successful transfer. As the counter is not reset, later delegated transfers may fail and restrict token sales. The owner can change the restriction through `addAllow` and overwrite an account balance through `mint` without updating `totalSupply`.

Detection results. The scores report each method's predicted probability for the malicious MINT label
Predicted malicious: Our method (95.68%).
Predicted benign: SmartIntentNN (33.70%), SmartIntentV2 (35.87%), FTSmartAudit (56.22%), and CodeXplain (4.26%).

whether its response differs between malicious and benign contracts, while the probe weight measures its contribution to the learned decision boundary. Malicious-logic specificity further distinguishes neurons that respond to changes in malicious semantics from those primarily sensitive to superficial code variations. Combining these complementary signals therefore favours neurons that are simultaneously discriminative, decision-relevant, and semantically specific, rather than neurons that score highly under only one criterion.

▷ ***Can the identified security knowledge generalize beyond malicious behaviours observed during training?***
Our experiments indicate that the identified representations exhibit meaningful, although not universal, cross-behaviour generalization. In the leave-one-intent-out evaluation, all localization and training stages exclude the held-out malicious intent, which is used only for testing. ContractLens still achieves higher average performance than the evaluated baselines. We further evaluate contracts deployed after the release of the backbone model and observe strong detection performance on several malicious mechanisms. These results suggest that the localized representations capture reusable characteristics of malicious contract logic rather than only memorizing individual attack categories. Nevertheless, ContractLens should not be interpreted as a universal detector for arbitrary future attacks, and its coverage remains dependent on the security knowledge represented by the underlying pretrained model.