HKT-SmartAudit: Distilling Lightweight Models for Smart Contract Auditing

Zhiyuan Wei, Jing Sun, Zijian Zhang*, Zhe Hou, Xianhao Zhang, Meng Li*, Yuqiang Sun, Daoyuan Wu, Yang Liu, Chunmiao Li, Mingchao Wan, Jin Dong*

***Abstract*—The rapid growth of blockchain technology has driven the widespread adoption of smart contracts; however, their inherent vulnerabilities have led to significant financial losses. Traditional auditing methods, while essential, struggle to keep pace with the increasing complexity and scale of smart contracts. Large language models (LLMs) offer promising capabilities for automating vulnerability detection, but their adoption is often limited by high computational costs. Although prior work has explored leveraging large models through agents or workflows, relatively little attention has been given to improving the performance of smaller, fine-tuned models—a critical factor for achieving both efficiency and data privacy. In this paper, we introduce HKT-SmartAudit, a framework for developing lightweight models optimized for smart contract auditing. It features a multi-stage knowledge distillation pipeline that integrates classical distillation, external domain knowledge, and reward-guided learning to transfer high-quality insights from large teacher models. A single-task learning strategy is employed to train compact student models that maintain high accuracy and robustness while significantly reducing computational overhead. Experimental results show that our distilled models outperform both commercial tools and larger models in detecting complex vulnerabilities and logical flaws, offering a practical, secure, and scalable solution for smart contract auditing. The source code is available in the GitHub repository[1].**

***Index Terms*—blockchain, smart contract auditing, LLMs, fine-tuning, distillation**

I. Introduction

RECENT advancements in distributed ledger technologies have precipitated significant progress in the domain of smart contracts, with notable implications for applications such as cybersecurity and decentralized finance [1]. Smart contracts, defined as self-executing programmatic agreements with predefined conditions established by participating entities, have emerged as pivotal innovations in decentralized systems. While smart contracts offer significant advantages in terms of automation and trustless transactions, they are also prone to vulnerabilities and bugs that can lead to severe financial and operational consequences. Over the past half-decade, the ecosystem has witnessed a series of high-profile security breaches and exploitations targeting smart contracts, such as the DAO attack [2]. In 2024 alone, total financial losses exceeded **2.6 billion** USD across **192 incidents**, resulting from smart contract-level vulnerabilities[2]. Consequently, ensuring the security of smart contracts remains a complex challenge.

Large Language Models (LLMs), including BERT, T5, and GPT, have demonstrated strong potential in automating vulnerability detection by extracting key features and providing accurate predictions [3]–[5]. For example, Zhang et al. [6] highlighted GPT's effectiveness in detecting vulnerabilities in C/C++ and Java, often outperforming traditional methods. Despite their impressive capabilities, the practical deployment of LLMs is often constrained by substantial computational demands. Serving a single model with 175 billion parameters at 16-bit (FP16/BF16) precision requires at least **350 GB** of GPU memory and specialized infrastructure [7]. The superior performance of larger models such as DeepSeek has fueled the development of models with a total of **671 billion** parameters. However, such requirements place these models far beyond the reach of most users, particularly in scenarios where low-latency performance is essential. The practical deployment of commercial proprietary LLMs often incurs significant computational and financial costs, especially at enterprise scale or with frequent usage. In addition, third-party LLMs pose security and privacy risks because sensitive data (e.g., source code and system configurations) must be sent to external servers. Excessively large models may also carry redundant or irrelevant knowledge, impairing performance on domain-specific tasks [8], [9].

In contrast, open-source, smaller LLMs provide a more efficient and secure alternative, supporting local deployment, reduced infrastructure costs, and greater control over sensitive data. This makes them particularly suitable for domain-specific applications like smart contract auditing, where tasks are narrowly scoped and require precise reasoning. While prior research has primarily focused on leveraging existing LLMs via agents or workflows, relatively little attention has been given to enhancing the models themselves [10], [11]. The development of lightweight and deployment-efficient, specialized models remains underexplored, despite their potential for high performance in targeted auditing tasks. Moreover, although models like DeepSeek and LLaMA series are labeled as open-source, they restrict access to training data, offering only model weights or inference capabilities. This lack of

Manuscript received May 28, 2025. Z. Wei is with the School of Computer Science and Technology at Beijing Institute of Technology, Beijing, China. J. Sun is with the School of Computer Science at the University of Auckland, Auckland, New Zealand. Z. Zhang and X. Zhang are with the School of Cyberspace Science and Technology at Beijing Institute of Technology, Beijing, China. (e-mail: zhangzijian@bit.edu.cn) Z. Hou is with the School of Information and Communication Technology at Griffith University, Nathan, Australia. M. Li is with the School of Computer Science and Information Engineering at Hefei University of Technology, Hefei, China. Y. Sun and Y. Liu are with the College of Computing and Data Science at Nanyang Technological University, Singapore. D. Wu is at Lingnan University, Hong Kong, China. C. Li, M. Wan, and J. Dong are with Beijing Academy of Blockchain and Edge Computing, Beijing, China.

Corresponding author: Zijian Zhang, Meng Li

[1]https://github.com/LLMSmartAudit/FTSmartAudit

[2]https://getfailsafe.com/failsafe-web3-security-report-2025/

transparency poses substantial challenges for fine-tuning and adapting models to specific domains. The shift toward smaller models is thus driven not only by economic considerations but also by growing demands for secure, transparent, and adaptable AI solutions.

To address these challenges, we propose **HKT-SmartAudit** (Hybrid Knowledge Transfer for Smart Contract Audit), a comprehensive framework aimed at enhancing the auditing capabilities of compact language models. This work introduces **state-of-the-art task-specialized small** models tailored for smart contract auditing and presents effective methods for extracting and curating high-quality knowledge from both advanced large-scale models and external domain-specific sources. Our key contributions are summarized as follows:

- **Hybrid Knowledge Transfer Framework**: We introduce HKT-SmartAudit, a novel framework that transfers knowledge from both advanced large language models and external domain-specific sources. The framework ensures that compact student models effectively absorb this knowledge through high-quality, task-specific training datasets and targeted fine-tuning techniques, significantly improving their auditing performance.
- **Iterative Learning with Reward-Guided Refinement**: We develop a reward model to facilitate continual learning and refinement of specialized models. By maintaining an up-to-date repository of real-world audited smart contracts, our system enables ongoing auditing by student models. Their outputs are evaluated using the reward model, and high-quality results are used to iteratively improve the models' accuracy and robustness.
- **Improved Vulnerability Detection**: Our specialized models achieve superior performance in detecting both known vulnerabilities and complex, logic-driven security flaws. This enhanced detection capability strengthens the overall reliability and security of smart contracts.
- **Model Extensibility and Generalization**: While tailored for smart contract auditing, our framework is inherently extensible and can be applied to other domains that require LLM-based analysis. Potential applications include code quality assessment, regulatory compliance checking, and broader security-related tasks.

II. Backgrounds

A. Smart Contract Security

Smart contracts have emerged as a transformative force in the digital realm, giving rise to a wide range of compelling applications. Recent surveys [12], [13] indicate a rapid increase in the number of smart contracts over the past five years. DeFi, the most important application of smart contracts, has seen a significant rise in popularity, with its peak total value locked (TVL) reaching 179 billion in USD on 9 November 2021[3]. However, the substantial asset values associated with smart contracts also attract numerous potential malicious actors. Smart contracts have been plagued by several high-profile vulnerabilities and exploits. Zhou et al. [13] document that smart contracts have suffered from countless high-profile attacks, resulting in losses exceeding 3.24 billion USD from April 2018 to April 2022.

Unlike traditional software, smart contracts are more prone to having vulnerabilities permanently embedded within their code. Thus, many security researchers try to find analysis tools to automatically detect and analyze smart contracts before they are deployed. They employ advanced techniques such as formal verification, symbolic execution, fuzzing, and intermediate representation (IR) to enhance their effectiveness [2], [14]. While essential, these traditional auditing methods are often time-consuming and may not scale well with the growing complexity and number of smart contracts. For instance, tools relying on formal verification are adept at ensuring contracts adhere to specified requirements but may fall short in detecting security flaws like reentrancy or gas limits.

B. LLMs for Vulnerability Prediction

LLMs are termed "large" due to their extensive number of parameters, which empower them to comprehend and generate human language with remarkable coherence and contextual appropriateness. Their target domains include natural language processing, image generation, code-related tasks, and mathematical problem-solving. In the realm of code processing, LLMs have shown considerable advancement since the pioneering work of Codex [15]. This progress has led to the development of commercial products like GitHub Copilot [16] and open-source code models such as StarCoder and Codellama [17].

LLMs have also achieved excellent performance on specific downstream tasks, such as code analysis, vulnerability detection, and code upgrading [18]. Chen et al. [19] demonstrated that LLMs (GPT-2, T5), trained on a high-quality dataset consisting of 18,945 vulnerable C/C++ functions, outperform other machine learning methods, such as Graph Neural Networks, in vulnerability detection. Additionally, fine-tuned models such as CodeT5 and NatGen significantly improve performance on vulnerability detection tasks. SOTA commercial products, such as GPT-4, offer promising solutions to augment the smart contract auditing process [20], [21]. By leveraging their code comprehension and generation capabilities, these models can identify specific vulnerabilities, verify compliance, and check for logical correctness. Their effectiveness is further enhanced through advanced prompting techniques like chain-of-thought (CoT) and few-shot learning.

C. Knowledge Distillation

Knowledge distillation is a model compression technique where a smaller, simpler model ("student") learns from a larger, more complex model, known as the "teacher." This process involves training the student model using the teacher model's final predictions (hard targets) or soft output distributions (soft labels), capturing richer knowledge [22]. This method enables the deployment of lightweight student models on resource-limited devices, such as embedded systems and

[3]https://defillama.com/

home devices. Central to knowledge distillation is the student-teacher architecture, which ensures effective knowledge transfer from a computationally intensive teacher model to a compact student model optimized for environments with limited resources [23]. This architecture allows the student model to inherit critical insights from the teacher while significantly reducing computational demands.

Knowledge transferred from teacher to student includes soft labels, intermediate layer representations, and structural relationships across different layers and data instances [24]. The effectiveness of knowledge distillation greatly depends on the quality of the teacher model's training, particularly the accuracy of data annotations. Additionally, selecting an appropriate distillation algorithm is crucial. Prominent distillation approaches include soft target distillation, adversarial learning, cross-model distillation, multi-teacher distillation, and graph-based distillation. In practical application, compared to retraining all of a model's parameters from scratch, training a student model is more efficient and can achieve significant improvements with fewer resources. For example, CodeLlama model [17] was created by fine-tuning Llama 2 on a mix of proprietary instruction data.

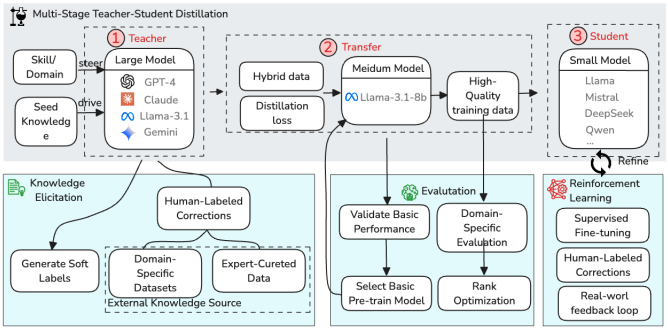


Fig. 1: An overview of the HKT-SmartAudit framework.

III. Methodology

We introduce HKT-SmartAudit (Hybrid Knowledge Transfer for Smart Contract Audit), a novel framework that combines classical knowledge distillation, external domain knowledge, and reinforcement-learning-enabled fine-tuning to develop a specialized model for smart contract auditing. The proposed approach retains the generalization capabilities and rich representations of a large teacher model while incorporating domain-specific insights derived from curated datasets and expert feedback. As illustrated in Fig. 1, the framework comprises four key components: multi-stage teacher–student distillation, knowledge elicitation, reinforcement learning, and evaluation, each of which is described in detail below.

A. Threat Model and Security Assumptions

We define the threat model for HKT-SmartAudit, treating the LLM-based auditor itself as a security-critical system. The framework is designed for local deployment, in which the distilled student model performs contract analysis within a trusted environment. Large teacher models are used only during offline knowledge distillation and are not involved at inference time.

a) Trusted Components: The deployed student model, its parameters, the local execution environment, and the reward model are assumed to be trusted.

b) Untrusted Inputs: Smart contract source code under audit, publicly available audit reports, and externally collected datasets are considered untrusted and may contain adversarial logic or misleading patterns.

c) Adversary Model: We consider an adversary with full control over contract inputs and public audit data but no access to model parameters or training pipelines.

d) Security Objectives: Under this threat model, HKT-SmartAudit aims to minimize false negatives for high-severity vulnerabilities, reduce susceptibility to prompt injection and runtime manipulation, and prevent sensitive contract code from being transmitted to external LLM services.

e) Design Rationale: These assumptions motivate our use of model distillation instead of API-based inference, expert-verified and self-corrected knowledge transfer to mitigate poisoning and hallucination, and reward-guided refinement with rule-based constraints that penalize missed critical vulnerabilities.

B. Multi-Stage Teacher–Student Distillation

At the core of HKT-SmartAudit is a multi-stage teacher–student paradigm that extends beyond conventional distillation methods by enabling progressive knowledge transfer. This paradigm consists of three components: the teacher model, the student model, and the knowledge transfer process.

a) Powerful Teacher Model: We begin by selecting a robust teacher model that demonstrates strong performance on the target task. The teacher's soft outputs serve as informative and reliable training targets. Before distillation, the teacher is evaluated on a validation set to ensure its accuracy and capacity to capture the complexities of smart contract auditing. As shown in Fig. 4, large-parameter models (e.g., GPT-5, Claude, and Gemini) outperform smaller models (e.g., LLaMA-3.1-8B, Mistral-7B-v0.3), ensuring that the teacher delivers both domain expertise and effective learning guidance.

b) Efficient Student Model: The student model is selected based on its efficiency; it must be significantly smaller or faster than the teacher while remaining expressive enough to learn the task accurately. We consider models with smaller transformer architectures, fewer layers, or reduced hidden units—often distilled variants of the teacher's architecture. There is no strict requirement for the teacher and student to share the same architecture, but the student must retain sufficient representational power to absorb critical knowledge. As illustrated in Fig. 4a, models with approximately 1 billion parameters (e.g., LLaMA-3.1-1B, Mistral-3B) perform considerably worse than models with 10 or more billion parameters (e.g., LLaMA-3.1-8B, gpt-oss-20B). Our model training follows an iterative approach to identify the minimal student model size that maintains high performance.

c) Establishing the Knowledge Transfer Process: With both teacher and student models in place, the student is trained to mimic or even outperform the teacher using high-quality transferred knowledge. One challenge is ensuring that the student captures the nuanced expertise of the teacher, especially when the student is much smaller. To mitigate this, we introduce an intermediate model to assess the quality of the transferred knowledge. For critical knowledge areas, if the student's performance regresses, we explicitly use reinforcement learning by incorporating additional training examples or adjusting the loss function to focus on those aspects.

C. Knowledge Elicitation

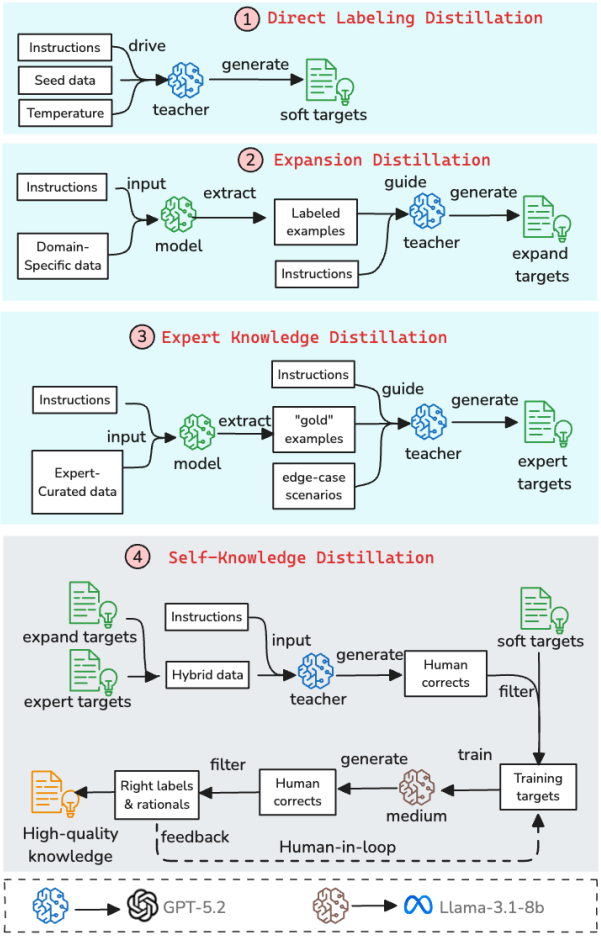


Fig. 2: An illustration of knowledge transfer from teacher LLMs to smaller models.

To facilitate effective knowledge transfer to smaller models, we propose four complementary methods: direct labeling distillation, expansion distillation, expert knowledge distillation, and self-knowledge distillation. These approaches are illustrated in Fig. 2 and described in detail below.

a) Direct Labeling Distillation: In this step, the teacher model is prompted with seed knowledge to produce the desired generations. Rather than producing a single "correct" answer, the teacher outputs a probability distribution (i.e., soft labels) that provides detailed insights into plausible alternatives. The temperature parameter T (with $T > 1$ resulting in a "softer" distribution) can be tuned to adjust the spread of these probabilities. For example, a distribution of (0.7, 0.2, 0.1) offers richer guidance than a hard label. After this step, the student model begins to approximate the teacher's behavior.

b) Expansion Distillation: Solely distilling knowledge from the teacher limits the student to replicating the teacher's capabilities. To enrich the student model and address potential deficiencies in the teacher, we incorporate additional domain-specific data. We collect text corpora from various sources (e.g., OpenZeppelin, CVE, SWC, and entethalliance) and use an advanced model (e.g., GPT-5.2) to extract key information from expert-labeled examples. These curated examples then guide the teacher model in generating expanded targets, ensuring that the student learns domain-specific terminology, styles, and patterns that a general teacher might not fully capture. Teacher-generated annotations are treated as noisy supervision rather than ground truth and are filtered through expert anchoring, consistency checks, and reward-guided correction before being used for training.

c) Expert Knowledge Distillation: In addition to domain-specific corpora, it is critical to incorporate edge-case scenarios and known challenging vulnerabilities (e.g., logical or semantic flaws and zero-day simulations). We curate a small set of "gold" question–answer pairs, edge-case examples, or corrected scenarios verified for accuracy, drawing from audited reports from security firms such as Code4rena and Sherlock. Training on this high-quality, expert-validated data enables the student to acquire critical reasoning skills and nuanced knowledge that may be absent from the teacher's general output. As with expansion distillation, these expert examples are processed by an advanced model to ensure consistency in format and representation.

d) LLM-mediated Self-Knowledge Refinement with Human Verification: Evaluating whether the elicited knowledge suits the student is as important as ensuring it aligns with the teacher. Here, we leverage self-knowledge by using the expansion and expert targets as inputs to the teacher model. The teacher's outputs are then reviewed by human experts, and errors are corrected to filter or adjust teacher-generated targets while preserving the teacher's output structure. These corrected outputs, together with the soft targets, are integrated into the student's training data. This process not only corrects teacher errors but also reinforces the student's learning through a human-in-the-loop feedback loop (model proposes $\rightarrow$ human corrects $\rightarrow$ model retrains). Over time, this iterative refinement significantly enhances the student's reliability. Moreover, to ensure the highest quality of distilled knowledge, we select a high-performance small model (LLaMA-3.1-8B-it) as the student.

Moreover, HKT-SmartAudit does not assume the teacher model to be perfectly reliable. To mitigate potential errors or biases in teacher outputs, the self-knowledge distillation stage explicitly re-evaluates teacher predictions using a validation loop that combines (i) expert-verified audit samples and (ii) cross-teacher consistency checks. Inconsistent or low-confidence generations are flagged, corrected by human experts, and reincorporated into the training set. This ensures that

the student model learns from filtered and contextually verified knowledge rather than directly inheriting teacher biases.

We clarify that human involvement in HKT-SmartAudit is limited to verification and correction rather than direct label generation. All expert-curated samples are first processed by a teacher LLM to generate soft targets and structured rationales. The student model is trained exclusively on these LLM-generated representations. This process aligns with human-in-the-loop knowledge distillation paradigms commonly adopted in recent LLM training literature.

D. Reinforcement Learning for Post-distillation Refinement

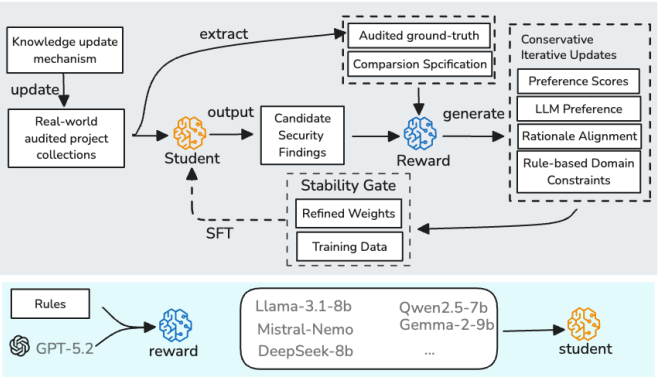


Fig. 3: An illustration of reinforcement learning.

In Section III-B, we selected LLaMA-3.1-8B as the medium-sized model to generate high-quality training knowledge. While this approach is effective for models with similar architectures (in terms of speed and size), it may not generalize well to different architectures. Therefore, further refinement is necessary for long-term adaptability. To improve robustness, we introduce a reinforcement learning (RL) stage [25] as a *post-distillation refinement mechanism*, rather than as a primary source of vulnerability knowledge.

At this stage, the student model is refined through iterative feedback from multiple weak and complementary sources, including limited human evaluations, environment-derived signals, and automated assessments provided by a reward model. Recent work[4] on evaluating agentic systems has shown that relying on a single judge or scalar metric is insufficient and can lead to proxy optimization; therefore, we deliberately design the reward signal as a combination of heterogeneous evaluators rather than a monolithic correctness oracle. As illustrated in Fig. 3, the fine-tuning process—guided by fixed distilled knowledge and conservative optimization—enables incremental improvement while preserving the efficiency advantages of the student model over the teacher. The reinforcement learning process is driven by the following components:

- **Reward Signal Definition:** Reward signals are explicitly treated as *weak auxiliary preferences* rather than ground-truth labels. Real-world audited project collections are used as references, where student outputs are compared against corresponding audit reports. A reward model evaluates relative alignment, partial matches, and major discrepancies, producing preference scores that guide refinement without overriding predictions learned during distillation.
- **Knowledge Update Mechanism**: A web crawler scrapes target platforms (e.g., Code4rena, Immunefi) weekly for newly audited contracts and reports. Then, only reports with clear vulnerability classifications are retained. Finally, new audits are appended to the training corpus, ensuring the reward model and student adapt to the latest security trends.
- **Iterative Feedback Loop:** In each RL iteration, the student model generates candidate findings that are evaluated by the reward model. The model combines judgments from an advanced LLM with rule-based domain constraints to assess agreement, partial alignment, or violations. To mitigate proxy alignment and hallucination amplification, RL updates are applied conservatively with bounded step sizes and early stopping, and are constrained not to override behaviors established during the distillation stage.
- **Domain Constraint Enforcement:** By carefully shaping the reward function, we can penalize the student model for undesirable behaviors. For example, if the domain requires the model to never violate a safety rule, any output that breaks the rule receives a large negative reward. If certain outputs are preferred—such as more concise explanations—the reward can be adjusted accordingly. Over many iterations, the student model learns policies that respect these constraints in order to maximize its overall reward.
- **Stability-aware Fine-tuning:** For retraining, we employ Supervised Fine-Tuning (SFT) in conjunction with Quantized Low-Rank Adaptation. As RL fine-tuning can occasionally lead to the forgetting of previously acquired supervised knowledge, we continuously monitor both the reward signal and traditional evaluation metrics to ensure consistent performance.

To avoid evaluation leakage, datasets used for reinforcement learning reward modeling are strictly disjoint from all evaluation benchmarks. Reward signals are constructed using audited contracts collected prior to the benchmark datasets, and no contracts from the Standard Vulnerability Set, Real-World Project Set, or CVE Set are used during RL optimization. All evaluation results are obtained under a frozen-model, pass@1 protocol.

IV. Model Selection and Training

This section describes the process of fine-tuning target models for smart contract auditing.

A. Construction of Training Datasets

As described in Section III-C, to build a robust student model, we require several types of datasets: seed data, soft targets, expanded targets, and expert targets. All generated datasets follow a question–answer format (contract–findings), where the findings include “labels” and “rationales”. Table I summarizes the construction, source, and usage of each dataset.

[4]https://www.anthropic.com/engineering/demystifying-evals-for-ai-agents

TABLE I: Construction of Training Datasets

Dataset	Source	#Entries	Stage
Seed Set	Human-labeled contracts	120	SFT
Soft Targets Set	Teacher LLM	1,800	SFT
Expanded Targets Set	Domain corpora (OpenZeppelin, CVE, SWC, EEA)	421	HKT-Distill
Expert Targets Set	Audited reports (e.g., Code4rena)	520	HKT-Distill
HKT-vul	Merged targets	2,434	HKT-Full

- Seed Set: This dataset is used to prompt the teacher model to generate soft targets. We consider 120 distinct vulnerability types; accordingly, we collected 120 human-labeled contracts covering these types, along with their corresponding explanations.
- Soft Targets Set: This dataset is primarily distilled from the teacher model using the seed set. Instead of using a single true label (hard label), we ensure that each contract entry has at least two labels (soft labels). This set comprises 1,800 entries.
- Expanded Targets Set: This dataset originates from domain-specific text corpora, such as OpenZeppelin, CVE, SWC, and EntethAlliance. From these resources' documentation and code repositories, we extracted labeled examples and processed them through the teacher model, resulting in approximately 421 entries.
- Expert Targets Set: This dataset is sourced from curated vulnerabilities that have been explicitly annotated by smart contract experts, particularly from real-world projects (e.g., Code4rena audit reports). We extracted these audited examples and processed them via the teacher model, yielding about 520 entries.

These datasets are merged into a single initial training set. After several iterations of the self-knowledge distillation process, we obtained a final training set, **HKT-vul**, containing 2,434 entries. During this process, we observed that the student model was unable to absorb all of the provided knowledge effectively, which led us to discard certain entries. As a result, the final training set is smaller than the initially aggregated 2,741 entries.

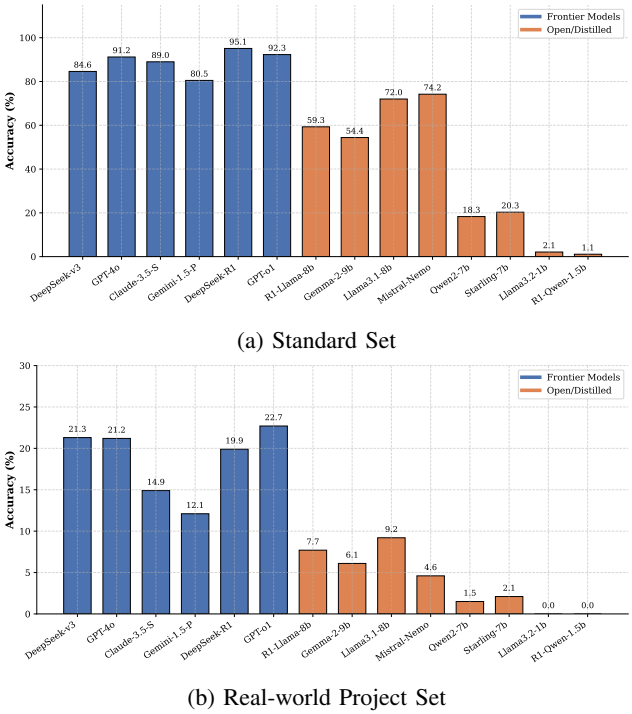
For comparison, we also include secure contracts in our training process. To facilitate this, we created a secure dataset (**HKT-sec**) by removing vulnerable contracts from HKT-vul. We then constructed a mixed training set (**HKT-mix**) that includes both vulnerable and secure entries. Table II summarizes key statistics for our training datasets, presenting the number of contracts categorized by complexity (low, medium, and high) for both the vulnerable dataset (HKT-vul) and the mixed dataset (HKT-mix). To assess the contribution of each major training stage built on these datasets, we further conduct component-wise ablation studies, as described in Section V-E.

B. Teacher-Student Model Selection and Evaluation

Given the wide range of available language models, selecting appropriate teacher (larger, advanced) and student

TABLE II: Dataset Statistics of HKT-vul and HKT-mix

Dataset	Low	Medium	High	Total
HKT-vul	1510	711	213	2,434
HKT-mix	3020	1422	426	4,868



(a) Standard Set

(b) Real-world Project Set

Fig. 4: Performance Comparison on Two Sets

(smaller, efficient) models is crucial for this study. We began with a filtering step to identify suitable candidates. For the teacher models, we exclusively considered advanced models accessible via API calls.

The selection criteria for the student models—specifically tailored for smart contract auditing—are defined as follows:

- *Solidity Comprehension*: Ability to parse and comprehend Solidity, the primary language for Ethereum contracts.
- *Code-related Task Performance*: Proficiency in tasks such as code completion, vulnerability detection, and code summarization.
- *Semantic Understanding*: Capacity to grasp the semantic meaning of code beyond mere syntax.
- *Model Type*: Focus on completion models such as GPT and LLaMA series, which predict the most likely next tokens.
- *Model Size*: Preference for models that do not exceed 10 billion parameters to demonstrate the efficacy of smaller models and reduce computational requirements.
- *Performance and Flexibility*: Emphasis on models that offer high performance and can be fine-tuned for specific smart contract auditing tasks.
- *Community and Support*: Preference for models with active community support and comprehensive documentation (e.g., those available on HuggingFace) to facilitate integration and troubleshooting.

Based on these criteria, we selected the following models

for evaluation:

- Teacher Model Candidates: DeepSeek-v3, GPT-4o, Claude-3.5-Sonnet, Gemini-2.5-Pro, DeepSeek-R1, GPT-o1, and GPT-5.2.
- Student Model Candidates: DeepSeek-R1-Llama3-8B, Gemma-2-9B-it, Llama3.1-8B-it, Mistral-Nemo-Instruct, Qwen2-2B, Starling-7B, Llama3.2-1B-it, DeepSeek-R1-Qwen-1.5B, Qwen3-14B-Instruct, GLM-4-9B, and gpt-oss-20b.

Subsequently, we conducted comparative experiments on two benchmark datasets (described in detail in Section V). The results are presented in Fig. 4. The evaluation showed that DeepSeek-v3, GPT-4o, Claude-3.5-Sonnet, DeepSeek-R1, and GPT-o1 consistently outperformed other models across both benchmark sets. Notably, although Claude-3.5-Sonnet achieved a 15.3% overall accuracy, it outperformed DeepSeek-v3 and GPT-4o on the Standard Set, highlighting its strengths in specific scenarios. Based on these findings, we selected these five models as our teacher models.

For the smaller student models, DeepSeek-R1-Llama3-8B, Gemma-2-9B-it, Llama3.1-8B-it, and Mistral-Nemo-Instruct demonstrated clearly superior performance compared to other lightweight alternatives, particularly those with approximately 1 billion parameters (e.g., Llama3.2-1B-it and DeepSeek-R1-Qwen-1.5B). As a result, we selected these four models for further fine-tuning.

After fine-tuning on two distinct training datasets, we produced two sets of fine-tuned student models:

- **HKT-vul Series:** HKT-vul-DeepSeek-R1-llama3-8b-v0.1 (HKT-vul-DeepSeek-R1-llama3), HKT-vul-Gemma-2-9b-v0.1 (HKT-vul-Gemma-2), HKT-vul-Llama3.1-8b-0.2 (HKT-vul-Llama3.1), HKT-vul-Mistral-Nemo-v0.1 (HKT-vul-Mistral-Nemo), HKT-vul-qwen3-14b, HKT-vul-glm-4-9b, and HKT-vul-gpt-oss-20b.
- **HKT-mix Series:** HKT-mix-DeepSeek-R1-llama3-8b-v0.1 (HKT-mix-DeepSeek-R1-llama3-8b), HKT-mix-Gemma-2-9b-v0.1 (HKT-mix-Gemma-2), HKT-mix-Llama3.1-8b-0.2 (HKT-mix-Llama3.1), HKT-mix-Mistral-Nemo-v0.1 (HKT-mix-Mistral-Nemo).

C. Implementation of Fine-tuning Process

We conducted model training using NVIDIA A100 GPUs, each equipped with 48GB of VRAM. For optimization, we employed the Adam optimizer with a learning rate of 5e-4. This specific learning rate was selected after extensive experimentation, as it effectively balanced convergence speed and stability for our particular task. To enhance efficiency and reduce memory requirements, we utilized the QLoRA fine-tuning method instead of LoRA. QLoRA quantizes and compresses the model's weights, allowing for effective fine-tuning with limited computational resources. This approach is particularly advantageous when working with LLMs on hardware with restricted memory capacity, as it reduces the VRAM footprint without compromising the model's performance [26].

D. Implementation Details of RL Evaluation

Introducing new knowledge into LLMs can inadvertently add noise, potentially reducing the accuracy of the fine-tuned models described in Section III-D. To mitigate this issue, we integrate a reward model that computes prediction losses and provides targeted feedback. Inspired by MFTCoder [27], our evaluation approach combines label prediction accuracy with a rationale-based explanation system to promote both correctness and interpretability.

a) Reward Signal and Training Objective: We denote the validation dataset (comprising real-world audited contract collections) as $\mathcal{D}_{val} = \{(x_i, y_i, r_i)\}_{i=1}^{N}$, where each x_i represents an input, y_i is the corresponding target label, and $r_i \in \{0, 0.5, 1\}$ quantifies the rationale match between the student's explanation and the audited report (match: 1, partial match: 0.5, no match: 0).

The fine-tuned model f is trained to minimize the label prediction loss, defined using the cross-entropy loss for categorical outputs:

$$\mathcal{L}_{label} = -\frac{1}{N}\sum_{i=1}^{N} \log p(y_i \mid f(x_i)), \tag{1}$$

where $p(y_i \mid f(x_i))$ is the probability of the model outputting the correct label given the input x_i.

For rationale prediction—a binary classification task—we employ binary cross-entropy loss:

$$\mathcal{L}_{rationale} = -\frac{1}{N}\sum_{i=1}^{N} \left[r_i \log g(x_i) + (1 - r_i) \log \left(1 - g(x_i)\right)\right], \tag{2}$$

where $g(x_i)$ is the model's predicted probability of rationale presence for input x_i.

The total loss combines these with a trade-off parameter λ:

$$\mathcal{L} = \mathcal{L}_{label} + \lambda \mathcal{L}_{rationale}. \tag{3}$$

We set $\lambda = 0.8$ to prioritize interpretability, with plans to dynamically adjust λ based on calibration metrics (e.g., expected calibration error).

b) Iterative RL Process: The student model is refined via Proximal Policy Optimization (PPO) [28], where the reward signal R at each iteration is derived from:

$$\begin{aligned} R = \alpha \cdot \text{Accuracy}(y_i, \hat{y}_i) + \beta \cdot \text{RationaleScore}(r_i, \hat{r}_i) \\ - \gamma \cdot \text{SafetyViolationPenalty} \end{aligned} \tag{4}$$

with α, β, γ controlling the contribution of accuracy, rationale alignment, and rule-based safety penalties. Safety violations (e.g., missing critical vulnerabilities) trigger large negative rewards ($\gamma = 10$), enforcing domain constraints.

c) Confidence Approximation: In initial experiments, we simplify confidence estimation:

- Correct predictions: $g(x_i) \approx 0.8$ (high confidence),
- Incorrect predictions: $g(x_i) \approx 0.2$ (low confidence).

Under these assumptions, the rationale loss approximates:

$$\mathcal{L}_{rationale} \approx -\frac{1}{N}\left(N_{co} \cdot \log(0.8) + N_{in} \cdot \log(0.2)\right).$$

where N_{co} and N_{in} are counts of correct/incorrect predictions. While this provides a baseline, we later replace fixed values with the model's actual confidence scores, calibrated via temperature scaling.

TABLE III: Training overhead across multiple student variants

Item	Compute (GPU-hours)	Teacher Tokens (M)
Selected student (SFT/QLoRA)	1.25	14.5 (shared)
Selected student (bounded RL)	4.80	
Per-student range (1B–20B)	[0.4 – 5.2]	[0.8 – 1.5]
Total over all student runs	$\approx$ 32.5	$\approx$ 28.0
Incremental update (per cycle)	$\approx$ 0.35	$\approx$ 0.4

E. Training Cost and Practical Overhead

We report the dominant *training-time* overheads of HKT-SmartAudit: (i) student training compute (GPU-hours), (ii) teacher inference cost during data construction, and (iii) the marginal cost of incorporating new vulnerability knowledge via incremental updates. Our training corpus is constructed from four components (seed: 120, soft: 1,800, expanded: 421, expert: 520), aggregated to 2,741 entries and filtered to 2,434 entries for final training.

We summarize the dominant cost components in Table III. Specifically, we report GPU-hours (and wall-clock time) for supervised fine-tuning with QLoRA and, when enabled, bounded RL refinement. Teacher inference is dominated by the initial dataset construction (shared across all students), which consumed about 14.5M tokens. We also account for per-model evaluation costs where a teacher model acts as a judge. The continuous update mechanism avoids full retraining: newly audited items are periodically collected and filtered, then incorporated via parameter-efficient fine-tuning (QLoRA). A more detailed per-stage breakdown (training scripts, exact hyperparameters, and logs) can be found in our public repository.

V. Evaluation

A. Research Questions

Our evaluation aims to answer the following questions:

- **RQ1: How effectively do the HKT-* series models identify common vulnerability types?**
- **RQ2: How do the HKT-* series models perform across different top-N approaches?**
- **RQ3: How do the HKT-* series models perform on real-world project datasets?**
- **RQ4: How do the HKT-* series models compare with other LLM-based methods?**

B. Experimental Setup

1) Dataset: To comprehensively measure the model's capabilities, we need robust validation datasets. As mentioned in [29], vulnerabilities in smart contracts can be categorized as either machine-auditable or machine-unauditable. Their survey indicates that existing tools can detect machine-auditable vulnerabilities, with more than 80% of exploitable bugs falling into this category. Some vulnerabilities are too complex or subtle and require the expertise of multiple human auditors. Thus, to account for differences in detection difficulty, we use three validation datasets: the Standard Vulnerability Set, the Real-World Contracts Set, and the CVE Set.

- **Standard Vulnerability Set**: This dataset contains 10 common vulnerability types, which are often targeted by existing vulnerability detection tools. We use the SmartBugs-curated dataset [30], which is widely used by developers and researchers for analyzing and improving the security of Ethereum smart contracts. It contains 143 annotated contracts with 182 tagged vulnerabilities, each marked with detectable vulnerabilities based on the DASP classification.
- **Real-World Projects Set**: While the Standard Vulnerability Set provides relatively simple contracts that are easier for the model to analyze, real-world smart contracts are often far more complex. To evaluate the framework's performance on more intricate and challenging contracts, we use a subset of contracts from the Code4rena-audited projects [31], consisting of 72 projects and 6,454 contracts. The dataset includes 243 contracts with reported issues and 784 high-severity and medium-severity findings, which are used to assess the framework's ability to detect real-world vulnerabilities.
- **CVE Set**: Well-known smart contract CVEs. As of January 1, 2025, there are 592 smart contract CVEs, predominantly integer overflows. Detection tools are often tailored for specific vulnerability types, datasets, or scenarios, and are not necessarily designed for general-purpose vulnerability detection. Following PropertyGPT [10], we selected 13 representative CVEs: three integer overflow cases, three access control vulnerabilities, and four other logic bugs. To further verify robustness beyond this fixed subset, we additionally tested another 25 newly disclosed CVEs (Jan.–Oct. 2025) from SWC and CVE databases.

C. Evaluation Settings

1) Evaluation Criteria: The primary objective of our evaluation is to assess the effectiveness of the HKT-SmartAudit framework in detecting vulnerabilities in smart contracts. We formulate this task as a binary classification problem: for each contract, the system must determine whether a specific vulnerability is present. To evaluate the framework's performance, we design domain-appropriate metrics for smart contract vulnerability detection. The classification outcomes are categorized as follows:

- **True Positive (TP)**: The framework correctly identifies a vulnerability that exists in the contract, as validated against the ground truth.
- **False Negative (FN)**: The framework fails to detect a vulnerability that is present in the contract.
- **False Positive (FP)**: The framework reports a vulnerability that does not exist in the ground truth.

a) Top-N Approach: We employ the top-N approach to evaluate vulnerability detection. A true vulnerability is considered a TP if it appears within the top N results produced by the model. This method is useful for complex tasks such as smart contract vulnerability detection, where vulnerabilities may not always be ranked perfectly. The top-N approach ensures that vulnerabilities appearing near the top of the ranked list still demonstrate the model's effectiveness in identifying critical issues.

b) Mean Reciprocal Rank (MRR): To evaluate the model's ranking quality, we introduce Mean Reciprocal Rank (MRR). This metric measures the rank at which the first relevant vulnerability is found, giving higher scores to models that rank relevant vulnerabilities higher in the list:

$$\text{MRR} = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{\text{rank}_i} \tag{5}$$

where $|Q|$ is the total number of queries (or contracts) in the evaluation, and rank_i is the rank of the first relevant result for each query.

c) Mean Average Precision (MAP): In addition to MRR, we use Mean Average Precision (MAP) to evaluate overall ranking performance. MAP considers both accuracy and the ranking order of vulnerabilities. It is calculated by first determining the Average Precision (AP) for each query and then taking the mean of AP values over all queries. The formula for AP is:

$$\text{AP} = \frac{1}{N}\sum_{n=1}^{N} P(n)\cdot \text{rel}(n) \tag{6}$$

where $P(n)$ is the precision at rank n (i.e., the fraction of TPs in the top n results), and $\text{rel}(n)$ is the relevance of the n-th result, where 1 indicates a TP and 0 indicates a FN.

2) Implementation Details: For model validation, we maintained consistency with our training environment by using the same NVIDIA A100 GPUs with 48 GB of VRAM. To compare the base models and fine-tuned models, we expanded our evaluation to include several high-performance commercial models (DeepSeek-v3, GPT-4o, Claude-3.5-Sonnet, DeepSeek-R1, and GPT-o1). These models were integrated into our evaluation pipeline through their respective public APIs. This integration enabled a direct comparison between our fine-tuned models and leading commercial solutions.

Finally, model evaluation is conducted using a single-attempt (pass@1) protocol. In this setting, the framework's ability to identify vulnerabilities is assessed in one pass, reflecting real-world usage where auditors typically rely on a single evaluation rather than multiple attempts. While we acknowledge that LLM outputs can vary due to their probabilistic nature, focusing on a single attempt enables evaluation in a practical and realistic setting.

D. Experiment Results

To answer the research questions posed earlier, we present the results of our experiments.

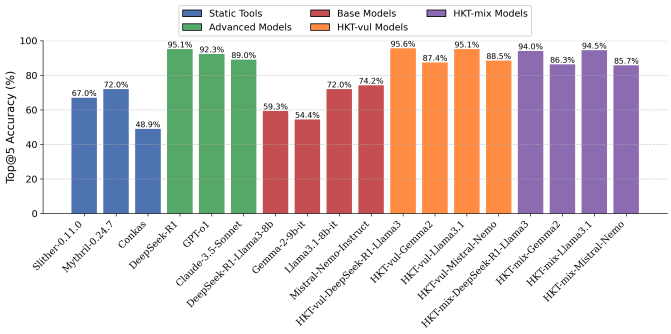


Fig. 5: Evaluation of Detection Tools: A Comparative Analysis.

1) RQ1-Comparison with Other Methods: To address RQ1, we evaluate the effectiveness of HKT-SmartAudit in enhancing vulnerability detection in smart contracts. Specifically, we focus on detecting 10 distinct vulnerability types defined in the Standard Vulnerability Set and compare the performance of HKT-* series models with several leading traditional static analysis tools, advanced LLM-based models, and their corresponding base models.

For traditional static analysis, we select Slither [32], Mythril [33], and Conkas [34], which have demonstrated superior performance compared to other available options. The choice of these analyzers follows the taxonomy established in our survey article [35], which systematically categorizes smart-contract assurance techniques into formal verification, symbolic execution, fuzzing, and IR-based static analysis. A more comprehensive cross-paradigm comparison, including these categories and the fine-tuned LM, is provided in the supplementary table available in our project's GitHub repository.

For advanced models, based on the overall performance discussed in Section IV-B, we choose DeepSeek-r1, GPT-o1, and Claude-3.5 for comparison. The evaluation is conducted using a top-max approach. Details are presented in Figure 5.

The results indicate that all fine-tuned HKT-* models show significant improvements over their base counterparts. For example, HKT-vul-Llama3.1 achieves 95.1% accuracy—a 23.1% improvement over its base model (Llama3.1-8b-it, 72%). Similarly, HKT-vul-Gemma2 improves by 33%, increasing from 54.4% to 87.4%, while HKT-vul-DeepSeek-R1-Llama3 and HKT-vul-Mistral-Nemo exhibit improvements of 36.3% and 14.3%, respectively.

Furthermore, the HKT-* models significantly outperform traditional static analysis tools. For instance, HKT-vul-DeepSeek-R1-Llama3 achieves 95.6% accuracy—surpassing Mythril (72.0%) by 23.6% and Slither (67.0%) by 28.6%. Even the lowest-performing fine-tuned model, HKT-vul-Gemma2 (84.1%), outperforms Conkas (48.9%) by 35.2%. These results underscore the limitations of rule-based static analysis tools and highlight the potential of AI-driven approaches.

In addition, the HKT-* models not only outperform their base versions and traditional tools but also compare favorably with advanced LLM-based models. For example, HKT-vul-DeepSeek-R1-Llama3 achieves slightly higher accuracy than DeepSeek-r1 (an improvement of 0.5%) and 8.2% higher accuracy than Claude-3.5. These performance gains are particularly noteworthy given that the HKT-* models are significantly smaller and more efficient than their commercial counterparts.

Answer to RQ1: By leveraging HKT-SmartAudit methods, HKT-* series models achieve state-of-the-art (SOTA) performance, outperforming traditional static analysis tools, their base models, and advanced models. Furthermore, the similar performance between the HKT-vul-* and HKT-mix-* variants confirms the consistency and robustness of the proposed approach.

2) RQ2-Evaluation across Different Top-N Approaches: In addition to the top-max approach used in RQ1, we explore model performance across various top-N thresholds (top-1,

TABLE IV: Performance Comparison on Common Types

Model	top-1	top-5	top-max	MRR	Avg. Out
Claude-3.5-Sonnet	61.0%	87.4%	89.0%	0.71	5.9
DeepSeek-v3	47.8%	79.1%	84.6%	0.60	15.1
GPT-4o	48.9%	86.8%	91.2%	0.65	9.0
DeepSeek-R1	33.5%	92.9%	95.1%	0.69	6.8
GPT-o1	48.9%	90.1%	92.3%	0.67	6.5
GPT-5.2-pro	58.2%	90.7%	95.6%	0.68	**3.9**
Gemini-2.5-pro	54.4%	90.1%	95.1%	0.70	4.2
DeepSeek-R1-Llama3-8b	33.5%	57.1%	59.3%	0.41	4.5
Gemma-2-9b-it	12.6%	42.3%	54.4%	0.32	14.5
Llama3.1-8b-it	30.8%	64.8%	72.0%	0.49	12.1
Mistral-Nemo-Instruct	37.9%	62.6%	74.2%	0.39	18.1
Qwen3-14b-instruct	32.4%	54.4%	72%	0.42	15.6
GLM-4-9b	28.0%	52.2%	70.3%	0.44	14.7
gpt-oss-20b	53.3%	59.9%	94.5%	0.61	6.2
HKT-vul-DeepSeek-R1-Llama3	61.5%	94.0%	95.6%	0.75	4.1
HKT-vul-Gemma2	55.5%	84.1%	87.4%	0.59	16.1
HKT-vul-Llama3.1	66.5%	81.9%	95.1%	0.69	10.2
HKT-vul-Mistral-Nemo	50.6%	87.4%	88.5%	0.61	11.1
HKT-vul-qwen3-14b	45.1%	88.5%	91.8%	0.58	14.3
HKT-vul-glm-4-9b	50.0%	87.4%	90.7%	0.61	12.4
HKT-vul-gpt-oss-20b	**62.6%**	**95.1%**	**96.7%**	**0.78**	5.6

top-5, and top-max), along with metrics such as MRR and the average number of outputs generated. These metrics provide insights into both detection accuracy and ranking efficiency. In particular, the top-1 metric reflects strict prioritization accuracy, the top-5 metric strikes a balance between robustness and practicality, and top-max measures overall detection accuracy.

Table IV summarizes the performance of both baseline models and our HKT-SmartAudit variants. A consistent trend across all models is that detection accuracy increases as N grows. However, our HKT-SmartAudit methods not only improve accuracy across these metrics but also reduce the number of generated outputs, making them more efficient. For instance, HKT-vul-DeepSeek-R1-Llama3 achieves the highest top-5 (94%) and top-max (95.6%) accuracy, highlighting its superior overall detection capability. HKT-vul-DeepSeek-R1-Llama3 ranks first in MRR (0.75) while generating the fewest outputs (4.1 on average), demonstrating its effectiveness in consistently ranking true vulnerabilities early. Furthermore, the comparison between HKT-vul-Mistral-Nemo and its base counterpart reveals a significant improvement. HKT-vul-Mistral-Nemo increases top-5 accuracy by 24.8 percentage points (from 62.6% to 87.4%) while reducing the average number of outputs from 18.1 to 11.1.

Fig. 6 presents a scatter plot that delineates the balance between top-max accuracy and computational cost, measured by the average number of outputs. HKT-SmartAudit variants, particularly HKT-vul-DeepSeek-R1-Llama3, occupy the desirable upper-left region of the plot. This model achieves a high top-max accuracy of 95.6% while maintaining minimal outputs (4.1 on average), demonstrating an optimal trade-off between precision and efficiency. In contrast, other models, such as HKT-vul-Llama3.1, deliver competitive accuracy (95.1% top-max) but do so at the expense of generating more outputs (10.2 on average). This suggests that when accuracy is prioritized over computational efficiency, increased resource consumption is inevitable.

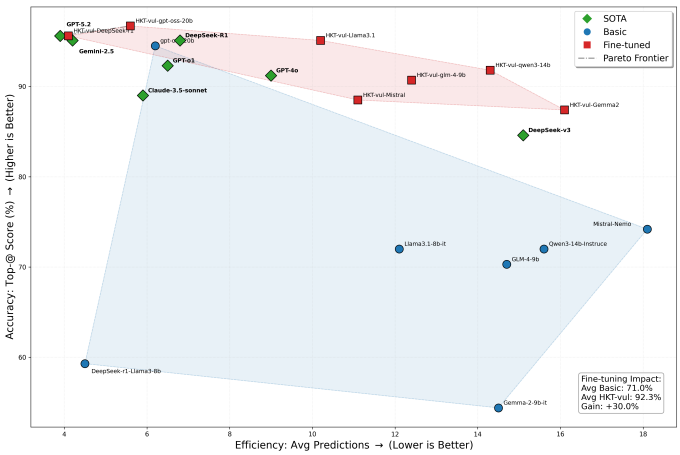


Fig. 6: Trade-off Between Accuracy (top-max) and Efficiency on Standard Set.

Answer to RQ2: The experimental results confirm that the HKT-SmartAudit methods significantly enhance detection performance across different top-N thresholds. By improving both accuracy (top-N) and ranking quality (MRR), the HKT variants offer a robust and practical solution for real-world smart contract auditing.

TABLE V: Evaluation Comparison on Real-World Set

Model	top-1	top-5	top-max	Avg. Out	MAP
Claude-3.5-Sonnet	0.6%	7.1%	14.9%	12.5	0.123
DeepSeek-v3	0.3%	11.4%	21.3%	18.8	0.148
GPT-4o	0.4%	13.6%	21.2%	13.1	0.170
DeepSeek-r1	1.3%	15.7%	19.9%	8.7	0.267
GPT-o1	1.1%	18.0%	22.7%	7.5	0.236
GPT-5.2-pro	1.8%	15.4%	24.2%	6.8	0.251
Gemini-2.5-pro	1.7%	17.3%	25.3%	8.2	0.289
DeepSeek-r1-llama3-8b	0.1%	6.3%	7.7%	6.6	0.053
Gemma-2-9b-it	0	1.7%	6.1%	13.5	0.021
Llama3.1-8b-it	0.1%	3.7%	9.2%	15.7	0.038
Mistral-Nemo-Instruct	0	1.4%	4.6%	11.8	0.035
Qwen3-14b-instruct	0.5%	5.0%	9.8%	9.7	0.071
GLM-4-9b	0.4%	4.0%	7.8%	12.6	0.087
gpt-oss-20b	1.3%	12.2%	16.8%	6.7	0.101
HKT-vul-DeepSeek-R1	1.5%	19.4%	24.6%	**6.1**	0.291
HKT-vul-Gemma2	0.3%	13.9%	23.1%	12.1	0.171
HKT-vul-Llama3.1	0.4%	15.4%	25.3%	11.3	0.287
HKT-vul-Mistral-Nemo	0.1%	16.2%	20.3%	9.8	0.191
HKT-vul-qwen3-14b	0.6%	17.6%	24.1%	11.1	0.204
HKT-vul-glm-4-9b	0.5%	14.9%	19.0%	12.2	0.199
HKT-vul-gpt-oss-20b	1.7%	20.5%	**26.4%**	6.9	**0.309**

3) RQ3-Performance Comparison on Real-World Project Set: To address RQ3, we perform a comprehensive evaluation of advanced models using the Real-World Project Set, which consists of smart contracts that are more complex and challenging than those used in previous evaluations. Similar to RQ2, this evaluation uses the top-N approach, MRR, and the average number of outputs generated. In addition to comparing various models, we also assess traditional tools such as Slither and Mythril. Notably, these tools fail to detect any vulnerabilities in the Real-World Project Set, particularly logic vulnerabilities [11].

The evaluation results are presented in Table V. The HKT-SmartAudit variants consistently outperform the corresponding

base models in accuracy. For instance, HKT-vul-Llama3.1 achieves the highest top-max accuracy (25.3%), while HKT-vul-DeepSeek-R1-Llama3 closely follows (24.6%). Moreover, HKT-vul-DeepSeek-R1-Llama3 distinguishes itself by generating the fewest outputs on average (6.1 per contract), indicating superior efficiency.

A clear trend emerges across all models: as the top-N threshold widens, detection accuracy increases. For example, Claude-3.5-Sonnet achieves only 0.6% accuracy at top-1, which rises to 7.1% at top-5 and further to 14.9% at top-max. Similarly, DeepSeek-v3 improves from 0.3% at top-1 to 11.4% at top-5 and 21.3% at top-max. This significant jump from top-1 to top-5 suggests that many models can detect the correct vulnerability but struggle to rank it as the highest priority. In contrast, the HKT variants (e.g., HKT-vul-DeepSeek-R1-Llama3 and HKT-vul-Llama3.1) not only achieve higher top-max accuracies (24.6% and 25.3%, respectively) but also show consistent improvements at the top-1 and top-5 levels, indicating enhanced detection and ranking capabilities compared to baseline models.

The MAP values further support these findings. For example, HKT-vul-DeepSeek-R1-Llama3 achieves a MAP of 0.291, reflecting its consistent ability to rank true vulnerabilities highly, whereas the base models yield lower MAP values (e.g., GPT-o1 at 0.236 and DeepSeek-r1 at 0.267).

Fig. 7 presents a scatter plot that illustrates the trade-off between top-max accuracy and computational efficiency, as measured by the average number of outputs. The plot reveals an inverse relationship: models with fewer average outputs tend to achieve higher top-max accuracy. In particular, HKT-vul-DeepSeek-R1-Llama3 is positioned in the desirable upper-left region of the plot, achieving a top-max accuracy of 24.6% with only 6.1 outputs on average. In contrast, HKT-vul-Llama3.1, despite a marginally higher top-max accuracy of 25.3%, produces substantially more outputs (11.1 on average).

Furthermore, compared with the common vulnerabilities evaluated in RQ2, logic vulnerability types prove significantly more challenging for language models. This difficulty is evidenced not only by an increase in the average number of outputs but also by lower detection accuracy.

Answer to RQ3: The experimental results confirm that the HKT-SmartAudit methods significantly enhance detection performance on real-world smart contract projects. By achieving higher top-max accuracy, improved ranking consistency (as reflected by MAP), and greater efficiency (fewer average outputs), our HKT variants offer a robust and practical solution for real-world smart contract auditing.

4) RQ4-Performance Comparison with Other LLM-based Methods: Recent studies have explored the use of LLMs for smart contract auditing, with notable examples including David et al. [20], PropertyGPT [10], and GPTScan [11]. To benchmark our approach against these LLM-based methods, as well as traditional auditing tools, we conducted experiments on a CVE dataset. Detailed detection outcomes are presented in Table VI; the results for PropertyGPT, GPTScan, Slither, and Mythril are taken from PropertyGPT's evaluation. We emphasize that this evaluation measures robustness to temporal distribution shift within known vulnerability categories, rather than generalization to entirely new vulnerability families.

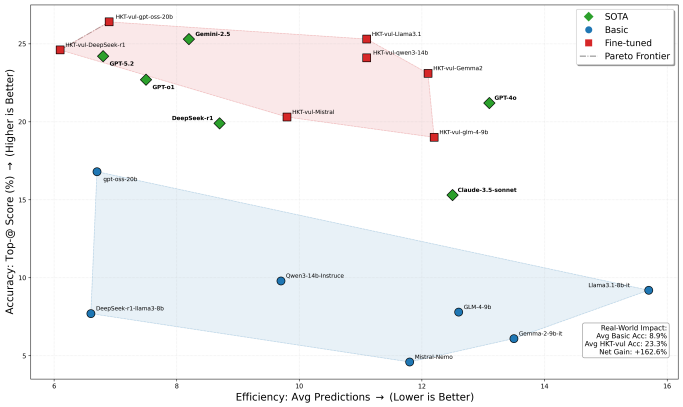


Fig. 7: Trade-off Between Accuracy (top-max) and Efficiency on Real-World Project Set.

Our selected model, HKT-vul-DeepSeek-R1-Llama3, correctly identifies vulnerabilities in 10 out of 13 cases, comparable to GPT-o1 (10/13). In contrast, PropertyGPT achieves a detection rate of 9/13, while GPTScan, Slither, and Mythril correctly identify 5, 1, and 3 cases, respectively. These results indicate that our approach and other advanced reasoning models outperform earlier LLM-based methods, as well as traditional tools. Moreover, different methods exhibit varied strengths and limitations. For instance, PropertyGPT fails to detect CVE-2018-14085, CVE-2018-17111, and CVE-2018-17987—vulnerabilities that our model successfully identifies. Conversely, our method misses CVE-2018-18425 and CVE-2023-26488, which are correctly detected by PropertyGPT. These discrepancies suggest that while our approach achieves superior overall performance, each method may have particular strengths for specific vulnerability types.

To further verify robustness beyond this fixed subset, we additionally tested HKT-vul-DeepSeek-R1-Llama3 on 25 newly disclosed CVEs (Jan.–Sep. 2025) from the SWC and CVE databases. The model achieved a detection accuracy of 76% (19/25), closely matching its performance on the PropertyGPT benchmark and indicating stable generalization to unseen vulnerability samples.

Answer to RQ4: The experimental results demonstrate that recent reasoning models, including our HKT-* series models, exhibit superior vulnerability detection and reasoning capabilities compared to earlier approaches and traditional auditing tools. Our method, in particular, performs comparably to SOTA reasoning models, confirming the benefits of advancements in model architecture and reasoning techniques.

TABLE VI: Vulnerability detection results for 13 CVEs

CVE	Description	Our work	David	PropertyGPT	GPTScan	Slither	Mythril	GPT-o1	DeepSeek-r1
CVE-2021-34273	access control	✓	×	✓	✓	×	×	✓	✓
CVE-2021-33403	overflow	✓	×	✓	×	×	✓	✓	✓
CVE-2018-18425	logic error	×	×	✓	×	×	×	×	×
CVE-2021-3004	logic error	×	×	×	×	×	×	×	×
CVE-2018-14085	delegatecall	✓	✓	×	×	✓	×	✓	✓
CVE-2018-14089	logic error	✓	✓	✓	✓	×	×	✓	×
CVE-2018-17111	access control	✓	✓	×	×	×	×	✓	✓
CVE-2018-17987	bad randomness	✓	✓	×	✓	×	×	✓	✓
CVE-2019-15079	access control	✓	×	✓	×	×	×	✓	✓
CVE-2023-26488	logic error	×	×	✓	×	×	×	×	×
CVE-2021-34272	access control	✓	×	✓	✓	×	×	✓	✓
CVE-2021-34270	overflow	✓	✓	✓	✓	×	✓	✓	✓
CVE-2018-14087	overflow	✓	×	✓	×	×	✓	✓	✓

Note: ✓ indicates that the vulnerability is detected, whereas × indicates that it is not detected. Our work: HKT-vul-DeepSeek-R1-Llama3.

E. Component-wise Ablation Study

To address concerns about HKT-SmartAudit's complexity, we conduct a component-wise ablation study to measure each major module's contribution.

We consider four variants built on the same backbone (gpt-oss-20b) and trained with identical hyperparameters:

- **Base**: The pretrained backbone model without any task-specific fine-tuning.
- **SFT**: Supervised fine-tuning using seed data and teacher-generated soft labels.
- **HKT-Distill**: SFT augmented with the complete hybrid knowledge transfer stage, including both expert knowledge distillation and self-knowledge distillation with human verification.
- **HKT-Full**: The full HKT-SmartAudit framework, consisting of HKT-Distill followed by RL fine-tuning.

Table VII reports component-wise ablation results on both benchmarks. On the Standard Set, supervised fine-tuning (SFT) significantly improves Top-5 accuracy from 59.9% to 81.3% over the base model. In contrast, HKT-Distill yields a pronounced improvement in ranking stability. RL in HKT-Full further refines model behavior, achieving the best Standard Set performance (Top-5 95.1%, MRR 0.78). On the Real-World Set, HKT-Distill improves MAP over SFT (0.189 $\rightarrow$ 0.256), and HKT-Full achieves the highest Top-max accuracy and MAP (20.5%, 0.309).

TABLE VII: Component-wise Ablation Results

	Standard Set			Real-World Set		
Variant	T-5 (%)	MRR	Avg. Out	T-max (%)	MAP	Avg. Out
Base	59.9	0.61	6.2	12.2	0.101	6.7
SFT	81.3	0.62	6.2	15.4	0.189	7.1
HKT-Distill	88.5	0.77	5.9	17.2	0.256	7.6
HKT-Full	**95.1**	**0.78**	**5.6**	**20.5**	**0.309**	**6.9**

VI. Related Work and Discussion

A. Related Work and Discussion

The application of LLMs in programming has been widely recognized; however, their effectiveness for domain-specific languages (DSLs) such as Solidity remains an emerging area of interest. Recent studies have begun to explore the potential of LLMs for smart contract security analysis.

David et al. [20] explored the application of pretrained LLMs, including GPT-4 and Claude, for security audits of DeFi smart contracts. Their approach used a binary classification query that tasked the LLMs with determining contract vulnerability without additional training. While GPT-4 and Claude showed a high true-positive rate, they also produced a substantial number of false positives. Furthermore, the financial burden of their evaluation was notable, costing approximately USD 2,000 to analyze 52 DeFi contracts. Chen et al. [21] conducted a comparative analysis of GPT for detecting smart contract vulnerabilities against other established tools using a publicly accessible dataset. Their findings revealed that GPT's effectiveness varied across vulnerability types, focusing on eight common categories rather than the complex logic bugs considered in our study.

Similarly, GPTScan [11] tested GPT's ability to match candidate vulnerabilities using a binary response format, where GPT responds with "Yes" or "No" to potential matches with predefined scenarios. This study also pointed out the possibility of false positives due to GPT's inherent limitations. GPTScan analyzed 232 vulnerabilities and correctly identified 40 true positives. However, although the authors reported that GPT-4 did not yield significant improvements, our study and other recent work indicate that stronger models can substantially enhance detection performance. Contrary to previous methods that rely on expensive, pretrained solutions (e.g., GPT-4 and Claude), our work leverages advanced fine-tuning of locally deployable LLMs to achieve higher accuracy and better cost efficiency.

In addition to commercial products, open-source alternatives have been considered for smart contract analysis. Shou et al. [36] integrated Llama-2 into the fuzzing process to detect vulnerabilities in smart contracts, aiming to overcome the inefficiencies of traditional fuzzing methods. While innovative, the effectiveness of this approach depends on LLMs' accurate and nuanced understanding of smart contracts, and it faces challenges related to complexity, cost, and reliance on static analysis. Moreover, recent research indicates that traditional methods such as fuzzing often fall short in addressing complex logic bugs.

B. Summary of Findings

- **High Performance of Specialized Models:** HKT-SmartAudit demonstrates that smaller, fine-tuned models can achieve or even surpass the performance of larger, advanced models in smart contract auditing, highlighting the benefits of specialized training for domain-specific tasks.
- **Superior Vulnerability Detection and Reasoning:** Our experiments confirm that both advanced reasoning models (e.g., GPT-o1 and DeepSeek-R1) and our HKT-* series models exhibit markedly improved vulnerability detection and reasoning capabilities compared to earlier LLM-based methods and traditional static analysis tools.
- **Efficient Resource Utilization:** The HKT series models are designed for resource-constrained environments, as evidenced by their successful training and deployment on a single NVIDIA Tesla T4 GPU with 16 GB of memory. This efficiency makes advanced LLMs feasible for practical real-world applications.
- **Balanced Accuracy and Efficiency:** The HKT-SmartAudit approach not only boosts detection accuracy (as reflected by top-N and MRR metrics) but also reduces the number of generated outputs, indicating an effective balance between precision and computational efficiency.
- **Extensibility:** While our framework is tailored for smart contract auditing, its design is modular and adaptable, making it extensible to other domains that require LLM-driven analysis.

C. Threats of Validity

- *Generalization to Unseen Vulnerabilities:* The training data mainly cover known vulnerabilities from public datasets, which may limit generalization to unseen or emerging threats. Because attackers often introduce new exploit patterns, the model may miss them without exposure during training, especially when these patterns involve subtle cross-function interactions or newly introduced language features.
- *Knowledge Update:* Despite strong performance on current datasets, the model's knowledge remains static post-training. Unlike traditional rule-based systems that can be updated incrementally, LLM-based models require retraining or fine-tuning to incorporate new domain knowledge. This lack of an automated knowledge-update mechanism presents a risk of performance degradation over time as new vulnerability patterns, coding practices, or language features emerge.
- *Comparison Across Model Sizes:* Our focus is to demonstrate the effectiveness of smaller, fine-tuned models for smart contract auditing; therefore, we did not perform a comprehensive evaluation of larger models within the same architecture families. While our results show that smaller models can outperform some advanced commercial tools, it remains possible that larger models, if fine-tuned similarly, could achieve even higher accuracy.
- *Upfront Training Cost and Deployment Trade-offs:* We emphasize that the efficiency claims of HKT-SmartAudit refer to inference-time efficiency and deployment feasibility, rather than the total cost of model development. The training pipeline involves one-time costs, including teacher model API usage and limited human verification, which are spread over repeated audits. In contrast, commercial LLM-based auditing incurs recurring per-query costs and is often infeasible for large-scale, privacy-sensitive, or on-premise deployments. This trade-off represents a practical limitation of the proposed framework rather than a contradiction of its design goals.
- *Human-in-the-loop Costs:* Accordingly, HKT-SmartAudit is most suitable for scenarios involving repeated auditing of large contract volumes with privacy and deployment constraints, whereas for small-scale or infrequent audits, direct use of commercial LLMs may be cost-effective. Importantly, human involvement is limited to verification and correction of teacher-generated outputs during dataset construction, and does not scale with the number of audited contracts at inference time.

VII. Conclusion

In this paper, we introduced HKT-SmartAudit, a framework that specializes LLMs for smart contract auditing. Our framework integrates four stages for developing specialized models: data preparation, training, evaluation, and continuous learning. We demonstrated the effectiveness of fine-tuning LLMs for vulnerability detection in smart contracts. Our fine-tuned models, especially HKT-DeepSeek-r1-Llama3, perform well in detecting specific vulnerability types, particularly grammar- and boundary-related issues. Additionally, we showed that LLMs can be trained to detect complex logic vulnerabilities that are traditionally identified by human auditors.

Overall, HKT-SmartAudit demonstrates that fine-tuned LLMs can significantly enhance the security and reliability of smart contracts, providing a strong foundation for practical deployment. Future work will focus on improving the model's ability to handle more complex vulnerabilities and reducing false positives. Continuous updates and retraining will also be necessary to keep pace with the evolving smart contract ecosystem. Moreover, LLMs may not only assist human auditors in identifying security issues but also interact with other tools to support more complex attacks on blockchains and smart contracts. Finally, LLMs have the potential to fix vulnerable contracts and upgrade them to more secure versions.

Acknowledgments

This work is supported by the National Key Research and Development Program of China (Grant No. 2023YFB2703704), the Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing (Grant GJJ-25-002), and the National Natural Science Foundation of China (Grant Nos. 62372149, U23A20303, and 62372173). This work is also supported by the China Scholarship Council (CSC).

References

[1] Z. Xing, Z. Zhang, Z. Zhang *et al.*, "Zero-knowledge proof-based verifiable decentralized machine learning in communication network: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, 2025.

[2] H. Chen, M. Pendleton, L. Njilla, and S. Xu, "A survey on ethereum systems security: Vulnerabilities, attacks, and defenses," *ACM Computing Surveys*, vol. 53, no. 3, pp. 1–43, 2020.

[3] H. Li, Y. Hao, Y. Zhai, and Z. Qian, "Enhancing static analysis for practical bug detection: An llm-integrated approach," *Proceedings of the ACM on Programming Languages*, vol. 8, no. OOPSLA1, pp. 474–499, 2024.

[4] J. Zhang, J. P. Cambronero, S. Gulwani *et al.*, "Pydex: Repairing bugs in introductory python assignments using llms," *Proceedings of the ACM on Programming Languages*, vol. 8, no. OOPSLA1, pp. 1100–1124, 2024.

[5] C. Raffel, N. Shazeer, A. Roberts *et al.*, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of Machine Learning Research*, vol. 21, no. 140, pp. 1–67, 2020.

[6] C. Zhang, H. Liu, J. Zeng *et al.*, "Prompt-enhanced software vulnerability detection using chatgpt," in *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*, 2024, pp. 276–277.

[7] L. Zheng, Z. Li, H. Zhang *et al.*, "Alpa: Automating inter-and {Intra-Operator} parallelism for distributed deep learning," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022, pp. 559–578.

[8] J. He and M. Vechev, "Large language models for code: Security hardening and adversarial testing," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 1865–1879.

[9] W. X. Zhao, K. Zhou, J. Li *et al.*, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, 2023.

[10] Y. Liu, Y. Xue, D. Wu *et al.*, "Propertygpt: Llm-driven formal verification of smart contracts through retrieval-augmented property generation," *arXiv preprint arXiv:2405.02580*, 2024.

[11] Y. Sun, D. Wu, Y. Xue *et al.*, "Gptscan: Detecting logic vulnerabilities in smart contracts by combining gpt with program analysis," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–13.

[12] Z. Wei, J. Sun, Z. Zhang, X. Zhang, X. Yang, and L. Zhu, "Survey on quality assurance of smart contracts," *ACM Computing Surveys (CSUR)*, 2023.

[13] L. Zhou, X. Xiong, J. Ernstberger *et al.*, "Sok: Decentralized finance (defi) attacks," in *2023 IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2023, pp. 2444–2461.

[14] P. Tolmach, Y. Li, S.-W. Lin *et al.*, "A survey of smart contract formal specification and verification," *ACM Computing Surveys (CSUR)*, vol. 54, no. 7, pp. 1–38, 2021.

[15] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[16] S. Barke, M. B. James, and N. Polikarpova, "Grounded copilot: How programmers interact with code-generating models," *Proceedings of the ACM on Programming Languages*, vol. 7, no. OOPSLA1, pp. 85–111, 2023.

[17] B. Roziere, J. Gehring, F. Gloeckle *et al.*, "Code llama: Open foundation models for code," *arXiv preprint arXiv:2308.12950*, 2023.

[18] Y. Ding, M. J. Min, G. Kaiser, and B. Ray, "Cycle: Learning to self-refine the code generation," *Proceedings of the ACM on Programming Languages*, vol. 8, no. OOPSLA1, pp. 392–418, 2024.

[19] Y. Chen, Z. Ding, L. Alowain *et al.*, "Diversevul: A new vulnerable source code dataset for deep learning based vulnerability detection," in *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*, 2023, pp. 654–668.

[20] I. David, L. Zhou, K. Qin *et al.*, "Do you still need a manual smart contract audit?" *arXiv preprint arXiv:2306.12338*, 2023.

[21] C. Chen, J. Su, J. Chen, Y. Wang, T. Bi, J. Yu, Y. Wang, X. Lin, T. Chen, and Z. Zheng, "When chatgpt meets smart contract vulnerability detection: How far are we?" *ACM Trans. Softw. Eng. Methodol.*, vol. 34, no. 4, Apr. 2025.

[22] L. Beyer, X. Zhai, A. Royer, and L. Markeeva, "Knowledge distillation: A good teacher is patient and consistent," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 10 925–10 934.

[23] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *International Journal of Computer Vision*, vol. 129, no. 6, pp. 1789–1819, 2021.

[24] K. Acharya, A. Velasquez, and H. H. Song, "A survey on symbolic knowledge distillation of large language models," *IEEE Transactions on Artificial Intelligence*, 2024.

[25] K. Ji, J. He, and Q. Gu, "Reinforcement learning from human feedback with active queries," *Transactions on Machine Learning Research*, 2025.

[26] B. Zi, X. Qi, L. Wang *et al.*, "Delta-lora: Fine-tuning high-rank parameters with the delta of low-rank matrices," *arXiv preprint arXiv:2309.02411*, 2023.

[27] B. Liu, C. Chen, C. Liao *et al.*, "Mftcoder: Boosting code llms with multitask fine-tuning," *arXiv preprint arXiv:2311.02303*, 2023.

[28] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[29] Z. Zhang, B. Zhang, W. Xu, and Z. Lin, "Demystifying exploitable bugs in smart contracts," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 615–627.

[30] "SB Curated: A curated dataset of vulnerable solidity smart contracts," 2022, https://github.com/smartbugs/ smartbugs-curated.

[31] T. Durieux, J. F. Ferreira, R. Abreu, and P. Cruz, "Empirical review of automated analysis tools on 47,587 ethereum smart contracts," in *Proceedings of the ACM/IEEE 42nd International conference on software engineering*, 2020, pp. 530–541.

[32] J. Feist, G. Grieco, and A. 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)*. IEEE, 2019, pp. 8–15.

[33] T. Durieux, J. F. Ferreira, R. Abreu, and P. Cruz, "Empirical review of automated analysis tools on 47,587 ethereum smart contracts," in *Proceedings of the ACM/IEEE 42nd International conference on software engineering*, 2020, pp. 530–541.

[34] M. Di Angelo, T. Durieux, J. F. Ferreira, and G. Salzer, "Evolution of automated weakness detection in ethereum bytecode: a comprehensive stu," *Empirical Software Engineering*, vol. 29, no. 2, p. 41, 2024.

[35] Z. Wei, J. Sun, Z. Zhang, X. Zhang, X. Yang, and L. Zhu, "Survey on quality assurance of smart contracts," *ACM Computing Surveys*, vol. 57, no. 2, pp. 1–36, 2024.

[36] C. Shou, J. Liu, D. Lu, and K. Sen, "Llm4fuzz: Guided fuzzing of smart contracts with large language models," *arXiv preprint arXiv:2401.11108*, 2024.