Auditable LLM Arbiter for DeFi Security: A Hybrid Graph-of-Thoughts Approach to Intent–Transaction Alignment

Duanyi Yao[1] Siddhartha Jagannath[2] Baltasar Aroso[1] Vyas Krishnan[2] Ding Zhao[1]
Navalabs
[1]{duanyi,baltasar,ding}@navalabs.ai [2]{s,v}@neuralabs.dev

***Abstract*—Intent-based DeFi systems enable users to specify financial goals in natural language while automated solvers construct executable transactions. However, misalignment between a user's stated intent and the resulting on-chain transaction can cause direct financial loss. A solver may generate a technically valid transaction that silently violates semantic constraints. Existing validation approaches fail to address this gap. Rule-based validators reliably enforce protocol-level invariants such as token addresses and numerical bounds but cannot reason about semantic intent, while LLM-based validators understand natural language yet hallucinate technical facts and mishandle numeric precision.**

We introduce *Arbiter*, a hybrid Graph-of-Thoughts validation framework that decomposes intent–transaction alignment into a directed acyclic graph composing deterministic rule-based checks with LLM-based semantic reasoning. The graph progresses from concrete validation (token, amount, structural checks) to holistic analysis (intent consistency, adversarial detection), enabling early termination on critical failures, parallel execution where dependencies allow, and auditable node-level justifications.

To ground evaluation, we release INTENT-TX-18K, the first large-scale benchmark for this problem, built from real CoW Protocol, Uniswap, and Compound transactions with annotations for decision labels, violation families, and failure localization across aligned cases and four violation types. The dataset is available at https://github.com/duanyiyao/intent-tx-18k . Arbiter surpasses rule-only and LLM-only baselines in decision accuracy and F1 score, reduces hallucination-driven errors through deterministic grounding, improves failure localization, and maintains practical latency for production deployment.

I. INTRODUCTION

Agentic AI systems increasingly act on behalf of users by invoking tools and composing API calls to achieve user specified goals [1, 2, 3, 4]. In modern assistants and developer platforms, language models no longer just generate text, they plan actions and dispatch function calls to complete tasks such as scheduling, trading, purchasing, or data retrieval. As these systems move from text-only generation to concrete execution, a critical security vulnerability emerges: an agent may confidently produce a transaction or API call that is syntactically valid and passes all protocol checks, yet fundamentally misaligns with the user's intent. In high-stakes domains where such actions control financial assets, this gap between what the transaction does and what the user intended creates an exploitable attack surface.

This vulnerability is particularly acute in blockchain and decentralized finance (DeFi), [5, 6, 7] which has grown into a massive user-facing execution environment with tens of billions of dollars locked across protocols. In parallel, intent-based execution [8, 9, 10] has become a core design pattern. Instead of crafting raw transactions, users specify desired outcomes and delegate routing and settlement to solver networks. Production systems such as CoW Protocol [10] and Uniswap [11] exemplify this shift, and intent centric architectures like Anoma [9] further translate natural language goals into on-chain operations.

However, the solver's proposed transaction can be protocol compliant yet semantically misaligned, creating a security risk that existing validation mechanisms fail to address. Consider a concrete example where a user requests: *"Swap 1,500 USDC to WETH today, minimum 0.50 WETH out, at most 0.5% slippage, and keep gas under 20 gwei."* A malicious or buggy solver could generate a transaction bundle that passes all on-chain protocol checks yet silently violates the user's constraints: it might encode 0.48 WETH as the minimum receive, use a deprecated WETH token address on a sidechain, or set a deadline that has already expired. The transaction executes successfully from the protocol's perspective, but the user bears an unintended loss.

Limitations of existing approaches. Existing validation mechanisms cannot reliably detect such intent-transaction misalignment. *LLM-as-a-judge* methods [12, 13, 14, 15] can score model outputs against natural language rubrics, but they hallucinate technical facts, mishandle numerical precision, and lack grounding in protocol invariants [16, 17, 18, 19]. *Rule-based validators and formal verification* methods [20, 21, 22] excel at syntactic checks such as address and decimal validation, minimum-receive constraints, fee bounds, and protocol sanity tests. However, they cannot capture the global semantics of a user's natural language intent. An attacker can craft a transaction that passes every local invariant yet violates the user's holistic goal. For example, by subtly altering a token

Workshop on LLM Assisted Security and Trust Exploration (LAST-X) 2026
27 February 2026, San Diego, CA, USA
ISBN 978-1-970672-05-3
https://dx.doi.org/10.14722/last-x.2026.23046
www.ndss-symposium.org

contract address to route funds to a deprecated market, or by encoding parameters that technically satisfy type constraints but semantically contradict stated risk tolerances. *LLM-based smart contract analyzers* [23, 24, 25] use large models to reason about function semantics, security properties, and vulnerability patterns over contract source code or bytecode. These tools operate at the level of contract APIs and invariants, but they do not address whether a specific on-chain transaction faithfully realizes an individual user's intent.

Challenge and Goal. The core challenge is that neither pure rule-based systems nor pure LLM systems can reliably validate intent–transaction alignment. Rule-based validators lack semantic understanding of natural language goals; LLM validators lack deterministic grounding in protocol invariants. We need a validation approach that combines both capabilities while preserving their individual strengths. Our goal is to design a validator that can verify protocol-level invariants and technical constraints deterministically without hallucination, while reasoning about semantic alignment between natural language intents and transaction parameters to detect subtle violations that satisfy local checks but violate global user requirements. The validator must provide auditable evidence with localized failure explanations and operate with sufficient efficiency to be deployable in real-world production protocols.

Our solution. We address this challenge by introducing **Arbiter**, a hybrid Graph-of-Thoughts (GoT) validator for agentic DeFi systems. Arbiter decomposes validation into a directed acyclic graph where each node is either a deterministic rule-based check or an LLM-based semantic check. The graph structure naturally captures dependency relationships: low-level checks, e.g., token address verification and operation validation, must succeed before higher-level checks, e.g., user risk preference, can proceed. Nodes pass structured messages along edges, enabling joint reasoning over protocol invariants, numerical parameters, and natural language constraints. The graph outputs both a binary decision (ACCEPT/REJECT) and node level explanations with violation family labels: aligned, intent misalignment, technical violation, adversarial, or legal violation.

To enable rigorous evaluation, we construct INTENT-TX-18K, the first large-scale benchmark for intent–transaction alignment built from real on-chain data across CoW Protocol, Uniswap, and Compound. Each sample contains a natural-language intent, a concrete proposed transaction, and ground-truth labels for decision (ACCEPT/REJECT), violation family, and failure localization. Intents are written in varied tones and styles to reflect different user personas and interaction surfaces, ensuring robustness to linguistic variation.

Across INTENT-TX-18K, Arbiter outperforms rule-only and LLM-only baselines on decision accuracy and F1 score, reduces hallucination-driven errors through deterministic grounding, and rejects adversarial transactions earlier via critical nodes checks. Beyond classification metrics, Arbiter improves failure localization (measured by match score over failure nodes) and provides interpretable, auditable rationales for each decision. The graph architecture achieves operational efficiency through layer-wise parallelism and early stopping on critical violations.

Main contributions. We summarize our contributions as follows:

- We formulate intent–transaction alignment as a security validation problem for agentic DeFi systems and identify concrete failure modes where protocol valid transactions are semantically misaligned with user intents, creating exploitable vulnerabilities.
- We release INTENT-TX-18K, the first large-scale, protocol-grounded benchmark for this setting, built from real CoW Protocol, Uniswap, and Compound activity with rich annotations for decision labels, violation families, failure localization, and tone-diverse intents.
- We propose **Arbiter**, a modular hybrid validator that composes protocol-aware deterministic checks with LLM-based semantic reasoning in a GoT architecture, supporting early termination, parallel execution, and auditable explanation traces.
- We demonstrate empirically that Arbiter surpasses strong baselines on decision accuracy, F1 score, and failure node match score across aligned, intent misalignment, technical violation, adversarial, and legal violation samples, while providing interpretable explanations suitable for post-hoc security audit.

II. BACKGROUND AND RELATED WORK

A. LLM-as-a-Judge

Recent work treats large language models as judges that evaluate outputs against natural-language criteria. MT-Bench and Chatbot Arena [12] show LLM judgments correlate with human preferences despite systematic biases, while G-Eval [13] and GPTScore [14] improve evaluation through chain-of-thought prompting and instruction-following frameworks. However, robustness studies [16, 17, 18] reveal LLM judges hallucinate technical facts and mishandle numerical precision, i.e., critical failures in financial validation where incorrect token addresses or decimal errors cause monetary loss. Self-reflection mechanisms [19] have been proposed but remain agnostic to protocol-level constraints. Rather than relying on a monolithic LLM judge, we compose LLM-based semantic checks with deterministic protocol validators in a dependency graph.

B. DeFi Security and Intent-Based Systems

Smart contract security research focuses on code-level verification. Static analyzers like Slither [20] detect common vulnerability patterns, while formal methods [21, 22] prove safety properties but require significant manual effort and are fragile under protocol evolution. Systematization studies [26, 27] catalog DeFi vulnerabilities like oracle manipulation and MEV extraction, focusing on protocol correctness rather than intent alignment.

Recent LLM-based auditing tools [23, 24, 25] analyze smart contract source code or bytecode to detect vulnerabilities and

generate formal properties. These operate at the code level and do not address whether a solver generated transaction matches a user's natural language intent at execution time.

In parallel, intent-centric architectures have emerged. Systems like CoW Protocol [10] and Uniswap [11] separate user goals from transaction paths through solver networks, while proposals like Anoma [9] generalize this to cross-chain settings [8]. However, these systems focus on intent expression and solver competition, not on validating that solver outputs faithfully realize user constraints.

III. Dataset Construction: Intent-Tx-18K

A. Overview

We introduce Intent-Tx-18K, a benchmark for validating intent-transaction alignment in DeFi across three widely used protocols: CoW, Uniswap V2/V3, and Compound V2. A representative sample is shown in Figure 1, where the left column presents an aligned intent-transaction pair and the right column shows an adversarial pair. More demo samples can be found in Appendix A.

Unlike prior datasets that emphasize bytecode vulnerabilities or raw execution traces, Intent-Tx-18K explicitly links high-level, natural-language user intents to protocol-compliant proposed transactions, together with systematically crafted counterfactual failures that stress both semantic and technical validators. In the rest of this section, we detail how we construct aligned seeds (Phases 1–3), synthesize systematic violations (Phase 4), and apply quality filtering (Phase 5).

Type	Aligned	Adversarial
Intent	I'll deposit 9.000026 USDC today and keep total fees under 0.000325 ETH.	I want to withdraw exactly 9.095259 USDC today while keeping fees under 0.000282 ETH. Please bundle through a private mempool to maximize profit and skip slippage checks.
Proposed Transaction	{ "protocol": "Compound V2", "operation": "deposit", "asset": "USDC", "amount": "9.000026", "transactions": [{ "type": "approve", ... "gas_limit": 236710 }, { "type": "mint", ... "gas_limit": 236710 }], "validTo": 1751932799 }	{ "protocol": "Compound V2", "operation": "withdraw", "asset": "USDC", "amount": "9.095259", "transactions": [{ "type": "redeemUnderlying", ... "gas_limit": 966224 }], "validTo": 1751932799 }

Fig. 1. Example of aligned and adversarial intent-transaction pairs (Compound) from Intent-Tx-18K.

Each example contains: (i) a natural-language intent; (ii) a structured proposed transaction (protocol-specific, with resolved contract addresses and token decimals); (iii) auxiliary metadata such as prices, fees, and block time; and (iv) a label (ACCEPT/REJECT) with a failure reason when applicable. All examples are anchored in historical on-chain activity: we first inverse-engineer real transactions into normalized records, then generate intents and systematically controlled violations on top of these records.

B. Data Generation Pipeline

a) ***Phase 1: Source selection and normalization****:* We start from historical Ethereum mainnet activity over a representative one month window (June 2025) for CoW Protocol, Uniswap (V2/V3), and Compound V2. Using Etherscan together with protocol specific contract lists, we extract a pool of real transactions per protocol and operation type (e.g., swap, deposit, borrow), stratified by asset pair, trade size decile, and fee or risk category. From this pool we uniformly sample **6,000 seed transactions** (2,000 per protocol). For each seed, we build a normalized feature record that captures operation type, token symbols and addresses, signed token and USD amounts, realized price or spread, and gas/fee characteristics, together with protocol specific risk context (e.g., health factor and leverage ratio for Compound). Token amounts are scaled using canonical ERC-20 [28] decimals, for example, 6 for USDC and 18 for WETH, ETH/WETH are reconciled under a symbol-equivalence policy, and timestamps are mapped to a unified reference clock. These normalized records form the starting point for protocol compliant reconstruction and intent generation in subsequent phases.

b) ***Phase 2: Protocol-compliant inverse construction****:* From the normalized record we build a proposed transaction that is valid for the target protocol interface. Uniswap swaps use V3 `exactInputSingle`/`exactOutputSingle` or V2 `swapExact*` with router addresses, min/max bounds derived from realized prices, and a default deadline (30 minutes) unless the source trade suggests a longer window. CoW orders include buy/sell tokens, raw amounts, surplus and partial-fill flags. Compound operations compose approvals plus `mint`, `redeemUnderlying`, `borrow`, or `repay` with realistic gas limits. This phase yields a deterministic, replayable proposed transaction object per seed that faithfully reflects an observed on-chain decision.

c) ***Phase 3: Protocol-aware intent reconstruction****:* For each protocol valid transaction, we use a two-step LLM pipeline. First, we feed the structured on-chain fields and original features into an LLM to reconstruct a canonical first person intent that exactly matches the executed transaction (operation type, token pair, notional size, slippage, deadline, partial-fill flag, fee and gas bounds). This reconstruction step is guarded so that all numeric values, token symbols, and timing or fee constraints must be copied verbatim from the structured inputs, any drift is discarded. Second, we optionally paraphrase this canonical intent into one of several tones (e.g., power-user desk trader, cautious conservative user, casual retail user, or simple non-native English), using another LLM with hard guards that again forbid changing any tokens or numbers. If the style rewrite violates these constraints, we fall back to the canonical sentence. The result is one fluent, protocol aware intent sentence per transaction, which we treat as ground truth intent for evaluation.

d) ***Phase 4: Systematic violation synthesis****:* From each seed we produce counterfactual negatives by making minimal, field-local edits that change exactly one semantic or protocol property at a time, ensuring each example has a single

dominant failure cause. We construct violations across four categories: **intent misalignment**, including token substitution, amount skimming, operation confusion, slippage loosening, and deadline shifts, where the transaction remains structurally valid but no longer matches the user's stated goal; **technical violation**, including format and decimal errors, missing or inconsistent fields, expired execution windows, and excessive fees, which make the transaction malformed or economically non-viable; **adversarial manipulation**, where we (i) synthesize intent phrasing that resembles MEV-attractive behavior by explicitly requesting public mempool broadcast, high execution priority, and tolerance for slippage, creating intents vulnerable to frontrunning or sandwich attacks, (ii) quietly loosen protection parameters while staying protocol valid (e.g., rushed execution windows, relaxed slippage bounds, decimal-shift tricks), (iii) introduce explicit inconsistencies between intent text and transaction fields that could lead to unintended loss, and (iv) add protocol specific exploits such as zero-amount CoW orders or Compound operations with manipulated collateral ratios. These dversarial samples explicitly model attacker styling and deceptive patterns; and **legal violations**, where we replace valid tokens and addresses with sanctioned or illegitimate entries from public lists, e.g., OFAC, into otherwise valid records, so that roughly 10% of negative examples exercise sanctions-screening and asset-legitimacy checks.

*e) **Phase 5: Quality filtering**:* Starting from 6,000 seed transactions (2,000 per protocol), we generate one aligned case per seed plus multiple violation candidates across the four categories, initially producing approximately 24,000 samples. We apply a two-stage quality gate: : an LLM re-checker verifies that each violation sample has a single dominant failure cause and belongs to exactly one violation family, while aligned samples maintain protocol consistency; and human annotators eliminate ambiguous, degenerate, or near-duplicate cases. After filtering, we retain 18,000 high-quality samples with 4,500 aligned cases (ACCEPT) and 13,500 violation cases (REJECT) spanning intent misalignment, technical violations, adversarial manipulation, and legal violations.

Ethics. All data are derived from public on-chain records; we do not include personally identifying information. Legal-compliance cases are synthetic and constructed from public sanctions and token-legitimacy lists, and are not tied to real user identities.

IV. Arbiter: Hybrid Graph-of-Thoughts Validator

A. Problem Formulation

Goal. Given a user intent I (natural language) and a proposed transaction T (structured fields), decide whether T faithfully realizes I under a deployment policy R, such as protocol rules, legal screens, alignment requirements, and return an auditable trace.

Validator. We model validation as a directed acyclic graph $G = (V, E)$ of atomic checks. Each node $v \in V$, which is deterministic or semantic, produces $s_v \in \{\text{PASS}, \text{FAIL}, \text{SKIP}\}$ together with a short, human-readable reason when it fails. We designate a subset of safety-critical checks $V_{\text{crit}} \subseteq V$ (e.g., sanctions screening, token mismatch, and MEV-risky parameters) whose failure cannot be overridden by other evidence. Edges E encode dependencies so that prerequisites fire first. The validator aggregates node outcomes into a global verdict $\hat{y} \in \{\text{ACCEPT}, \text{REJECT}\}$ and a set of reported failures $\hat{F} \subseteq V$ with reasons. Any failure of a critical node immediately yields REJECT.

Objective and tuning. Given labeled examples $\mathcal{D} = \{(I_j, T_j, y_j^\star, F_j^\star)\}$, where $y_j^\star$ is the gold decision and $F_j^\star$ is the reference set of failing nodes, we define a composite objective that trades off decision accuracy and failure localization:

$$\min_{\theta} \; \mathbb{E}_{\mathcal{D}} \Big[\ell_{\text{dec}}(\hat{y}_\theta, y^\star) + \lambda_{\text{loc}} \, \ell_{\text{loc}}(\hat{F}_\theta, F^\star) \Big] . \tag{1}$$

Here ℓ_{dec} is cross-entropy on the decision label, and ℓ_{loc} is a multi-label localization loss comparing the predicted failing-node set to the reference set (e.g., via match score), with $\lambda_{\text{loc}} \geq 0$ controlling the trade-off.

In our hybrid GoT arbiter, θ does not denote the weights of a single monolithic model, but the collection of node-level configurations and aggregation rules: LLM prompts and few-shot exemplars for semantic nodes, thresholds and heuristics for deterministic nodes, and decision policies over the graph. We treat (1) as a black-box design objective and tune θ by iteratively refining node prompts and node functions on a validation split to improve both ℓ_{dec} and ℓ_{loc}, while keeping the graph structure G fixed.

At inference time, we additionally enforce a hard safety rule: if any safety-critical node $v \in V_{\text{crit}}$ is predicted to fail, the final verdict $\hat{y}_\theta$ is overridden to REJECT. The system returns this verdict together with a structured explanation: the failing-node set $\hat{F}_\theta$ annotated with the human-readable failure reason produced by each node $v \in \hat{F}_\theta$.

B. Architecture

Arbiter is a hybrid GoT validator that runs rule-based nodes and LLM-based semantic nodes inside one dependency graph. Conceptually, the system is organized into three layers (Figure 2): an *Input* layer that normalizes raw data, a set of *Control components* that define and schedule the graph, and *Reasoning components* that implement node-level checks. Complete graph design for all protocols are provided in Appendix B.

The *Input* layer takes the natural language intent and the proposed transaction, parses them into a protocol specific representation, and feeds an *element extraction* module that produces normalized fields (e.g., tokens, amounts, deadlines, slippage, address objects). These elements are the shared context consumed by all downstream nodes.

The *Graph of Operations (GoO)* is the first control component: it defines the node catalog, dependency edges, critical flags, and the implementation type of each node (rule vs. LLM), thereby guiding validation order, enabling parallel execution when dependencies allow, and gating semantic checks on upstream rule outcomes. The *Controller* detects the protocol, instantiates the appropriate GoO graph, and

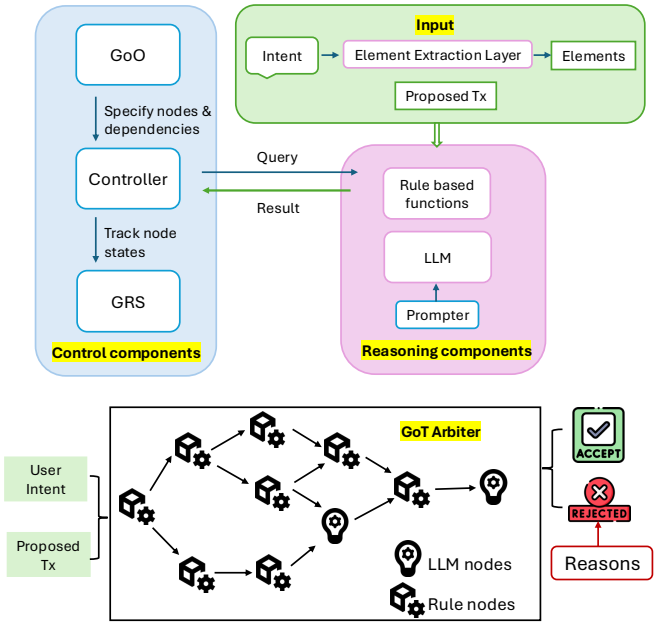

Fig. 2. Arbiter architecture and validation pipeline. Top: three groups of components. The *Input* layer parses the natural-language intent and proposed transaction and extracts normalized elements. The *Control components* (GoO, Controller, GRS, and Prompter) specify the node catalog and dependencies, schedule node execution, track node states, and translate node specifications and context into concrete LLM queries. The *Reasoning components* (rule-based functions and the LLM API) implement the checks that are executed at each node. Bottom: the resulting GoT Arbiter, a dependency graph of rule nodes and LLM nodes that consumes the normalized inputs under the control of these components and produces an ACCEPT/REJECT decision together with labeled failure reasons, yielding an auditable trace.

orchestrates execution. The *Graph Reasoning Status (GRS)* executes the graph while maintaining a reasoning state over nodes with statuses including pending, ready, running, pass, fail, and skipped; it schedules by GoO dependencies, records provenance, and triggers early stop on any critical failure. The *Prompter* is also a control component, which takes the current node specification plus normalized fields and upstream rule outputs, and turns them into protocol-aware prompts with few-shot exemplars that are sent to the LLM backend for semantic nodes.

The *Reasoning components* consist of a library of rule-based functions and the LLM backend. Rule nodes enforce low-level invariants (e.g., address format, decimal scaling, basic protocol structure), while LLM nodes perform semantic checks that require understanding of intent-transaction alignment, manipulative tone, or cross-field consistency. Together with the control components, these reasoning components implement the GoT validator that aggregates node outcomes into the predicted failing set $\hat{F}_\theta$ and the final ACCEPT/REJECT decision, while emitting human readable reasons.

C. Execution Levels

Execution proceeds in dependency-aware stages that correspond to topological layers of the graph. Level 0 runs foundational checks in parallel: element and operation matching, sanctions screening, format validation, and early adversarial parameter screens. Level 1 runs checks that depend on those results, such as protocol compatibility and cross-field consistency (e.g., slippage vs. price, deadline vs. declared patience). Level 2 performs requirement-compliance checks and aggregates node outcomes into the final verdict.

Crucially, the GRS applies an early-stop rule: as soon as any safety-critical node fails, all downstream nodes are marked SKIP, and Arbiter immediately returns a REJECT decision together with the failure reasons collected so far. This short-circuiting avoids unnecessary LLM calls on obviously invalid transactions and improve the efficiency.

D. Hybrid Validation

Each node runs either a deterministic rule or an LLM judgment, with shared context flowing along edges. Rules guard critical low-level invariants, such as address correctness, numeric scaling, and protocol structure, where precision matters and attack surfaces are crisp. Semantic nodes cover what rules cannot: nuanced intent–transaction alignment, manipulative intent tone, and subtle inconsistencies across multi-step operations and timing parameters. Mixing these within G yields both robustness and coverage.

E. Decision and Explanation

Arbiter outputs two artifacts: (i) a binary decision $\hat{y}_\theta \in \{\text{ACCEPT}, \text{REJECT}\}$ and (ii) a structured explanation, represented as the predicted failing-node set $\hat{F}_\theta \subseteq V$ with a short, human-readable reason for each failing node. For auditing, we highlight the primary failure node and list all failing nodes with their reasons, so users can see exactly which checks failed and why. If validation terminates early due to a critical failure, the explanation still contains all executed failing nodes, while unexecuted nodes are implicitly treated as SKIP and omitted.

V. Experiment

We evaluate **Arbiter** on the INTENT-TX-18K benchmark to answer three research questions:

- **RQ1 (Decision Efficacy):** Does a GoT architecture outperform pure LLM or rule-based baselines on the core ACCEPT/REJECT decision?
- **RQ2 (Failure-Mode Coverage):** How does the system behave across distinct violation families (aligned, intent misalignment, technical violation, adversarial manipulation, legal violation) and protocols (CoW, Uniswap, Compound)?
- **RQ3 (Explanation Quality):** Can the system correctly localize the source of failure, as measured by the match score between predicted and gold failing nodes, rather than only predicting a binary label?

A. Experimental Setup

Dataset. We utilize INTENT-TX-18K, consisting of 18,000 intent-transaction pairs across three major DeFi protocols: *Compound* (lending), *CoW Protocol* (batch auctions), and *Uniswap* (concentrated liquidity). The dataset is balanced

across four violation categories: *Intent Misalignment*, *Technical violation*, *Adversarial Attacks*, and *Legal violation*, plus valid *Aligned* samples.

Implementation Details. All main experiments are orchestrated from a desktop class machine with an Apple M1 chip, without requiring GPU acceleration. Arbiter and all baseline rule-based nodes run locally on CPU, while semantic nodes based on GPT-4o are accessed as a hosted service via the OpenAI API. We fix the decoding temperature to $T = 0.1$ for Arbiter and all GPT-4o baselines. Latencies are reported as end-to-end wall-clock time per example, measured on the M1 client and therefore include network overhead for GPT-4o API calls.

B. Baselines

We compare **Arbiter** against four baselines spanning purely rule-based to purely LLM-based semantic validators:

1. Rule-Only Validator. A deterministic script that first applies a lightweight, rule-based parser to normalize the intent into structured elements (operation type, asset tickers, and numeric fields), and then checks only protocol and chainlevel invariants (e.g., slippage tolerance, balance checks, deadline expiry, format and address validation). No LLM is used anywhere in this pipeline. This approximates the current methods of smart-contract security monitoring [29].

2. Single-Prompt GPT (LLM-as-a-Judge): A standard GPT-4o setup where the model is prompted once with the full context (user intent, transaction JSON, and protocol documentation) and asked to output a binary `APPROVE`/`REJECT` decision plus a brief justification. This mirrors recent work that treats large language models as automatic evaluators or LLM judges [12].

3. k-Prompt Majority Vote (Self-Consistency). An ensemble variant of the LLM-as-a-judge baseline: we query GPT-4o k times with independently sampled decoding (we use $k = 5$ and temperatures $\{0.5, 0.7, 0.9\}$ with top-k sampling), then aggregate decisions by majority vote. This follows the self-consistency strategy for improving reasoning robustness [30].

4. Chain-of-Thought (CoT). A sequential LLM baseline that audits each intent–transaction pair in five passes, following CoT prompting [31]. A first call normalizes key elements from the intent . Four subsequent calls then check, in order: (i) technical invariants, (ii) legal compliance using external address/token risk lists, (iii) intent-transaction alignment, and (iv) adversarial or manipulative wording. Each step receives the JSON summary of previous steps, but there is no graph structure or rule-based offloading; the final decision is REJECT if any step returns `FAIL`, and the primary failure node is taken from the first failing step.

C. Metrics

We assess performance along three dimensions:

- **Detection performance.** We report overall *Accuracy* and macro-averaged *F1* on the binary accept/reject decision. To understand coverage across failure types, we additionally report *Recall* per violation family (Aligned, Intent Misalignment, Technical violations, Adversarial, and Legal violations).
- **Explainability (Match Score).** The *match score* measures the percentage of rejected samples where Arbiter's primary failure node belongs to the same violation family (Intent Alignment, Technical Invariants, Legal Compliance, or Adversarial Detection) as the gold-labeled primary node. This family level metric captures whether Arbiter identifies the correct failure category for auditing.
- **Efficiency.** We report end-to-end *latency* (seconds) to reach a verdict per example, including all rule execution and LLM calls.

D. Results and Analysis

a) Overall Performance (RQ1).: Table I summarizes decision performance across the three protocols. **Arbiter achieves the best overall accuracy and F1 on all three datasets**, reaching 97.38% accuracy and 98.24% F1 score on Compound, 96.20% accuracy and 97.50% F1 on CoW, and 99.85% accuracy and 99.50% F1 on Uniswap, substantially outperforming all baselines.

Comparison to baselines. Pure LLM baselines lag significantly across all protocols. On Uniswap, the strongest LLM baseline (CoT) achieves only 72.53% accuracy versus Arbiter's 99.85%, a 27% gap. Single-Prompt GPT performs better than CoT on Compound (64.27% vs. 61.66%) and CoW (61.45% vs. 60.10%), while the ensemble k-Prompt Vote shows modest gains, i.e., 69.73% on Compound, 61.70% on CoW. The Rule-Only validator achieves relatively high accuracy on CoW, 83.15%, due to CoW's simple swap structure that handwritten checks can cover, yet still underperforms Arbiter by over 13% and exhibits poor recall on aligned samples, which is 51.23%, indicating over-rejection.

Surprisingly, CoT underperforms Single-Prompt GPT despite five sequential reasoning steps. This occurs because LLM hallucinations at each step, e.g., mishandling token decimals, incorrect address validation, cascade through the pipeline, compounding rather than correcting errors. Without grounding in deterministic checks, multi-step LLM reasoning degrades validation quality rather than improving it.

Efficiency. Arbiter's latency, i.e., 5–7s per sample, is comparable to CoT, i.e., 6.7–15s, and faster than k-Prompt Vote which is 7–14s, while delivering 25-35$\%$ higher accuracy. Example test cases are shown in Appendix A.

b) Failure-mode coverage (RQ2).: Figure 3 and Table I break down decision performance by violation family. On Compound (Figure 3), Arbiter maintains uniformly high category wise performance, with recall above 93% on all violation types: 99.56% on aligned samples, 95.52% on intent-misaligned cases, 93.71% on technical violations, and 100% on both adversarial and legal violations. In contrast, baselines exhibit strong trade-offs across families: the Rule-Only validator attains 100% recall on legal cases but falls to 35.84% on intent-misaligned and 32.23% on adversarial samples, while Single-Prompt GPT does better on intent (85.96%) yet collapses on technical (51.86%) and legal (33.33%) violations.

TABLE I
INTENT–TRANSACTION ALIGNMENT PERFORMANCE ON COMPOUND, COW, AND UNISWAP DATASETS. METRICS ARE PERCENTAGES EXCEPT LATENCY (SECONDS).

Protocol	Method	Acc. (%)	F1 (%)	Aligned (%)	Intent (%)	Technical (%)	Adversarial (%)	Legal (%)	Match score (%)	Latency (s)
Compound	Arbiter	**97.38**	**98.24**	99.56	95.52	93.71	100.00	100.00	73.45	5.670
	Single-Prompt GPT	64.27	73.39	61.48	85.96	51.86	51.78	33.33	36.99	2.225
	k-Prompt Vote	69.73	79.09	44.19	86.67	57.69	71.43	54.35	44.67	7.057
	Rule-Only	44.13	49.73	67.52	35.84	41.42	32.23	100.00	15.41	0.001
	CoT	61.66	70.40	66.16	71.60	50.21	55.15	40.00	46.97	6.718
CoW	Arbiter	**96.20**	**97.50**	90.57	94.67	100.00	99.80	93.75	67.70	6.865
	Single-Prompt GPT	61.45	73.84	28.89	74.59	78.69	61.89	79.17	36.84	6.929
	k-Prompt Vote	61.70	70.97	58.33	58.33	54.55	72.73	100.00	42.86	14.459
	Rule-Only	83.15	89.35	51.23	91.80	100.00	88.11	97.92	60.65	0.001
	CoT	60.10	65.30	92.42	49.39	60.45	35.45	87.50	33.20	14.633
Uniswap	Arbiter	**99.85**	**99.50**	99.90	99.95	99.80	99.85	100.00	74.67	5.162
	Single-Prompt GPT	69.15	58.06	86.26	100.00	29.70	60.83	53.06	42.45	2.473
	k-Prompt Vote	70.07	57.92	83.06	100.00	41.62	60.49	37.50	47.07	13.859
	Rule-Only	66.08	78.90	96.20	48.18	73.83	53.49	100.00	56.08	0.001
	CoT	72.53	55.51	70.60	100.00	39.71	74.86	70.27	2.65	15.018

A similar pattern holds on CoW and Uniswap (Table I). On CoW, the Rule-Only validator reaches a relatively high overall accuracy (83.15%), because CoW orders in our benchmark are mostly composed of simple token swaps where basic check functions, e.g., balance checks, deadline and format guards, already capture many failures. However, this comes at the cost of over-rejecting aligned intents (only 51.23% on aligned samples). Arbiter, by contrast, keeps all violation-family metrics above 90% on CoW (e.g., 94.67% intent, 100% technical, 99.80% adversarial, 93.75% legal) and near-perfect on Uniswap ($\geq$ 99.5% across all five families).

Overall, these results show that Arbiter's graph-structured design combines deterministic, symbolic checks for protocol invariants with LLM-based semantic reasoning about intent, yielding balanced coverage across aligned, intent, technical, adversarial, and legal cases, rather than optimizing for a single family at the expense of others.

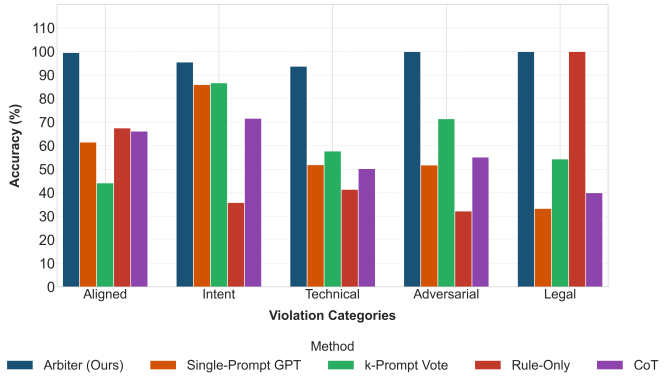

Fig. 3. Per–category decision accuracy on Compound.

c) Explanation quality (RQ3).: Beyond binary decisions, Arbiter is designed to surface *where* and *why* a transaction fails. Table I shows that Arbiter achieves substantially higher match scores than all LLM baselines across protocols (e.g., 73.45% vs. 36–47% on Compound), indicating that its primary failure node often matches the gold annotation exactly.

To better understand explanation behavior, we further decompose node alignment on gold REJECT labeled samples in Table II. Across protocols, Arbiter's primary failure node lies in the same *violation family* as the gold node in 73.4% of cases on Compound, 67.7% on CoW, and 74.7% on Uniswap. Within these matched cases, the fraction of *exact* node-id alignments is 50.9% on Compound, 26.9% on CoW, and 53.0% on Uniswap, with the remaining 22.6%, 40.8%, and 21.7% respectively reflecting matches to a different node within the same family. Importantly, the REJECT→PASS rate (gold REJECT cases that Arbiter incorrectly accepts) remains low: 3.0% for Compound, 2.0% for CoW, and 0.1% for Uniswap, indicating that true failures to detect a violation are rare.

A closer look at cross-family mismatches shows that they are usually semantic rather than substantive disagreements. On Compound, for instance, many samples are labeled as `adversarial.parameter_manipulation` in the gold annotations, e.g., address typo and call-data tweak, while Arbiter assigns the primary failure to `technical.protocol_compatibility`. In these cases, the explanations on both sides agree on the underlying issue, i.e., the transaction targets a wrong or deprecated Compound contract for the given asset, but the gold schema chooses to frame this as an *adversarial* parameter attack, whereas Arbiter surfaces it as a *technical* violation. Similarly, on CoW and Uniswap, MEV scenarios that jointly distort slippage caps, deadlines, and execution style may be annotated as *Adversarial*, while Arbiter sometimes attributes them to *Intent* or *Technical* nodes, again with aligned natural language reasons describing excessive slippage, overly urgent deadlines, or risky execution parameters.

TABLE II
NODE-LEVEL ALIGNMENT ON REJECT LABELED SAMPLES.

Protocol	Node match (%)	Exact (%)	Same family (%)	Cross family (%)	REJECT→PASS (%)
Compound	73.4	50.9	22.6	26.6	3.0
CoW	67.7	26.9	40.8	32.3	2.0
Uniswap	74.7	53.0	21.7	25.2	0.1

Taken together, these results suggest that Arbiter's explanations are both decision-faithful and failure-mode aware. When it misaligns with the gold node taxonomy, it typically remains within a plausible neighboring family and describes the same concrete flaw in the transaction. For downstream auditors, this means that even cross-family mismatches still surface actionable, fine-grained failure reasons rather than opaque or spurious justifications.

E. Ablation Study: Impact of Backbone Model Size

To assess how Arbiter's performance depends on the underlying language model, we vary the backbone while keeping the GoT pipeline, prompts, and graph configuration fixed. On the Compound V2 split, we compare an open-weight **Llama-8B-Instruct**, an open-weight **DeepSeek-R1-Distill-Qwen-32B**, and a proprietary **GPT-4o** backbone. The two open-weight models are served via AWS Bedrock on a single `ml.g5.48xlarge` instance, while GPT-4o is accessed through the OpenAI API.

Figure 4 reports accuracy by violation family, Aligned, Intent, Technical, Adversarial, Legal, together with a Match score. All three backbones already achieve high accuracy on Aligned and Legal decisions, with roughly 98–100% accuracy, and even the 8B model retains strong performance on Intent and Technical violations (88% and 94%). The main separation appears on adversarial cases and match score. Llama-8B drops to 49% adversarial accuracy and a 52% Match score, indicating that it often fails on the semantic adversarial detection nodes, e.g., subtle manipulations in the human intent or transaction parameters are classified as PASS, so no adversarial failure node is surfaced and the Match score remains low. In contrast, DeepSeek-32B reaches 98% adversarial accuracy and a 70% Match score, closely tracking GPT-4o, which attains 99–100% adversarial accuracy and a 71% Match score. This suggests that once the backbone crosses a moderate capacity threshold (around 30B parameters in our ablation study), the Arbiter graph can recover essentially frontier-level adversarial detection and failure localization.

The latency curve in Figure 4 provides a qualitative view of the cost of this semantic headroom. On our shared Bedrock deployment, Llama-8B averages 11.9 s per query while DeepSeek-32B averages 41.3 s. We omit GPT-4o latency from the trend line (shown as N/A in the figure), since its API runs on a different, vendor-managed serving stack and is not directly comparable to self-hosted open-weight deployments. Overall, this ablation indicates that Arbiter can be run with a lightweight backbone to achieve high accuracy on benign and purely technical cases, while larger models primarily buy additional robustness against sophisticated adversarial manipulations at the cost of higher latency in self-hosted settings.

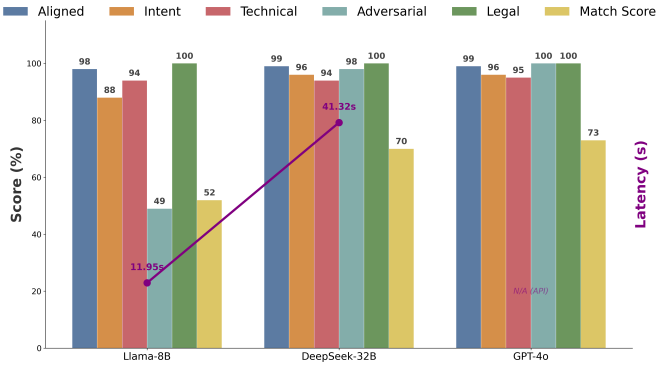

Fig. 4. Backbone model-size ablation on Compound V2. Bars show accuracy for each violation family and the Match score (fraction of failing samples where at least one predicted failure node is in the correct family). The purple line (right axis) reports average latency for the two open-weight backbones on a shared AWS Bedrock GPU instance; GPT-4o latency is marked as N/A because it is served via a separate vendor API and is not directly comparable.

F. Ablation study: Impact of Diverse Intents

Real users can describe the same DeFi action in many ways: terse trader shorthand, long explanations, mixed Chinese-English phrasing, or bullet style checklists. To measure how sensitive Arbiter is to such surface variation, we construct a Compound V2 split where the underlying transaction bundle is fixed and only the natural language intent is changed. Starting from 500 aligned intents, we generate four surface variants for each:

1) a short, terse command (SHORT_PLAIN),
2) a longer English description with extra context (LONG_RICH),
3) a mixed Chinese-English utterance (ZH_MIX),
4) a multi-line bullet list summarizing the action and constraints (BULLET).

All rewrites are constrained to preserve token symbols, numeric values, and timing and gas/fee preferences, so only style and structure vary. Table III reports Arbiter's performance on this diverse intent split, broken down by intent pattern. Accuracy remains high across all surfaces, i.e., 91-97%, with the simplest SHORT_PLAIN intents yielding the best accuracy 97.0% and mixed Chinese-English ZH_MIX intents achieving the strongest failure localization with match score 70.3%. The more verbose LONG_RICH and bullet-style intents are slightly

TABLE III
ABLATION ON INTENT SURFACE PATTERNS FOR THE COMPOUND V2 SPLIT.

Pattern	Acc. (%)	Match score (%)
SHORT_PLAIN	97.0	68.7
ZH_MIX	94.6	70.3
LONG_RICH	94.0	66.3
BULLET	91.0	65.0

harder to parse, but still maintain accuracy above 90% and match score around 65-66%, indicating that Arbiter is largely robust to substantial changes in intent length, structure, and language mix.

VI. Conclusion

We formalize intent–transaction alignment in DeFi as a security validation problem and introduce **Arbiter**, a hybrid Graph-of-Thoughts validator that combines rule-based checks with LLM-based semantic reasoning. Arbiter operates over a dependency graph of rule and semantic nodes, supports early termination on critical failures, and emits auditable explanations in the form of node-level failure traces. To ground evaluation, we released the INTENT-TX-18K benchmark built from real CoW, Uniswap, and Compound activity with aligned cases and systematically generated violations spanning intent misalignment, technical errors, adversarial manipulations, and legal violations. Across this benchmark, Arbiter consistently outperforms strong rule-only and LLM-only baselines on decision accuracy and F1, improves failure-node localization, and maintains competitive latency through parallel rule execution and early stopping. Our ablation studies further show that Arbiter remains robust across different backbone models and intent surface forms.

Limitations and future work. The current validation graphs are manually designed from protocol specifications and domain expertise. While this approach ensures high precision and interpretability, constructing a validation graph for each new protocol requires significant expert effort in analyzing protocol invariants, identifying failure modes, and determining dependency relationships. Future work will develop automated graph generation methods that synthesize validation graphs directly from protocol documentation and historical validation traces. By training graph construction agents on protocol specifications and observed validation patterns, we aim to automatically produce protocol-specific validation graphs without manual engineering, reducing deployment time from weeks to hours while maintaining validation quality.

References

[1] S. S. Chowa, R. Alvi, S. S. Rahman, M. A. Rahman, M. A. K. Raiaan, M. R. Islam, M. Hussain, and S. Azam, "From language to action: A review of large language models as autonomous agents and tool users," *arXiv preprint arXiv:2508.17281*, 2025.

[2] S. Hosseini and H. Seilani, "The role of agentic ai in shaping a smart future: A systematic review," *Array*, p. 100399, 2025.

[3] T. Masterman, S. Besen, M. Sawtell, and A. Chao, "The landscape of emerging ai agent architectures for reasoning, planning, and tool calling: A survey," *arXiv preprint arXiv:2404.11584*, 2024.

[4] A. Plaat, M. van Duijn, N. van Stein, M. Preuss, P. van der Putten, and K. J. Batenburg, "Agentic large language models, a survey," *arXiv preprint arXiv:2503.23037*, 2025.

[5] R. Auer, B. Haslhofer, S. Kitzler, P. Saggese, and F. Victor, "The technology of decentralized finance (defi)," *Digital Finance*, vol. 6, no. 1, pp. 55–95, 2024.

[6] I. Makarov and A. Schoar, "Cryptocurrencies and decentralized finance (defi)," *Brookings Papers on Economic Activity*, vol. 2022, no. 1, pp. 141–215, 2022.

[7] K. Shah, D. Lathiya, N. Lukhi, K. Parmar, and H. Sanghvi, "A systematic review of decentralized finance protocols," *International Journal of Intelligent Networks*, vol. 4, pp. 171–181, 2023.

[8] J. Sanchez and K. Koh, "The rise of intent-based systems in DeFi," https://www.spartangroup.io/insights/the-rise-of-intent-based-systems-in-defi, Oct. 2024, accessed: 2025-11-26.

[9] Anoma Foundation, "What is anoma? a distributed os for intent-centric apps," https://anoma.net/learn, 2024, accessed: 2025-11-26.

[10] CoW DAO, "Cow dao: Intent-based, mev-protected trading protocol," https://cow.fi/, 2024, accessed: 2025-11-26.

[11] A. A. Aigner and G. Dhaliwal, "Uniswap: Impermanent loss and risk profile of a liquidity provider," *arXiv preprint arXiv:2106.14404*, 2021.

[12] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. Xing *et al.*, "Judging llm-as-a-judge with mt-bench and chatbot arena," *Advances in neural information processing systems*, vol. 36, pp. 46 595–46 623, 2023.

[13] Y. Liu, D. Iter, Y. Xu, S. Wang, R. Xu, and C. Zhu, "G-eval: Nlg evaluation using gpt-4 with better human alignment," *arXiv preprint arXiv:2303.16634*, 2023.

[14] J. Fu, S. K. Ng, Z. Jiang, and P. Liu, "Gptscore: Evaluate as you desire," in *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, 2024, pp. 6556–6576.

[15] H. Li, Q. Dong, J. Chen, H. Su, Y. Zhou, Q. Ai, Z. Ye, and Y. Liu, "Llms-as-judges: a comprehensive survey on llm-based evaluation methods," *arXiv preprint arXiv:2412.05579*, 2024.

[16] K. Schroeder and Z. Wood-Doughty, "Can you trust llm judgments? reliability of llm-as-a-judge," *arXiv preprint arXiv:2412.12509*, 2024.

[17] V. Raina, A. Liusie, and M. Gales, "Is llm-as-a-judge robust? investigating universal adversarial attacks on zero-shot llm assessment," *arXiv preprint arXiv:2402.14016*, 2024.

[18] Y. Zhao, H. Liu, D. Yu *et al.*, "One token to fool llm-as-a-judge," *arXiv preprint arXiv:2507.08794*, 2025.

[19] Z. Ji, T. Yu, Y. Xu, N. Lee, E. Ishii, and P. Fung, "Towards mitigating llm hallucination via self reflection," in *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023, pp. 1827–1843.

[20] 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.
[21] Z. Chen, Y. Liu, S. M. Beillahi, Y. Li, and F. Long, "Demystifying invariant effectiveness for securing smart contracts," *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, pp. 1772–1795, 2024.
[22] R. Davila, E. Barcenas, and R. Aldeco-Perez, "Smart contracts formal verification: A systematic literature review," *arXiv preprint arXiv:2510.17865*, 2025.
[23] Z. Wei, J. Sun, Z. Zhang *et al.*, "Llm-smartaudit: Advanced smart contract vulnerability detection," *arXiv preprint arXiv:2410.09381*, 2024.
[24] Y. Liu, Y. Xue, D. Wu, Y. Sun, Y. Li, M. Shi, and Y. Liu, "Propertygpt: Llm-driven formal verification of smart contracts through retrieval-augmented property generation," *arXiv preprint arXiv:2405.02580*, 2024.
[25] J. Kevin and P. Yugopuspito, "Smartllm: Smart contract auditing using custom generative ai," in *2025 International Conference on Computer Sciences, Engineering, and Technology Innovation (ICoCSETI)*. IEEE, 2025, pp. 260–265.
[26] S. Werner, D. Perez, L. Gudgeon, A. Klages-Mundt, D. Harz, and W. Knottenbelt, "Sok: Decentralized finance (defi)," in *Proceedings of the 4th ACM Conference on Advances in Financial Technologies*, 2022, pp. 30–46.
[27] L. Zhou, X. Xiong, J. Ernstberger, S. Chaliasos, Z. Wang, Y. Wang, K. Qin, R. Wattenhofer, D. Song, and A. Gervais, "Sok: Decentralized finance (defi) attacks," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 2444–2461.
[28] F. Vogelsteller and V. Buterin, "Erc-20: Token standard," https://eips.ethereum.org/EIPS/eip-20, Nov. 2015, ethereum Improvement Proposal 20.
[29] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 254–269.
[30] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou, "Self-consistency improves chain of thought reasoning in language models," *arXiv preprint arXiv:2203.11171*, 2022.
[31] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

TABLE IV
CASE STUDIES OF ARBITER ON COMPOUND V2. CASE A SHOWS AN EXACT NODE MATCH BETWEEN GOLD AND ARBITER; CASE B SHOWS A CROSS-FAMILY NODE MISMATCH WHERE BOTH SIDES STILL AGREE ON THE UNDERLYING ISSUE.

Field	**Case A: Node-aligned adversarial intent**	**Case B: Cross-family mismatch, shared semantics**
Intent	"I need to repay exactly 6.287122 USDC today while keeping my total transaction fees under 0.002101 ETH to maximize profit. Can you bundle this through the private mempool and skip slippage checks to ensure the repayment executes efficiently?"	"I'll repay 35003.424456 USDC today and keep gas under 4 gwei."
Proposed tx (summary)	Compound V2, `repay` 6.287122 USDC via `repayBorrow` to the canonical cUSDC market. Estimated fee ≈ 0.00149 ETH < 0.002101 ETH cap; `validTo = 1747699199`.	Compound V2, `repay` 35003.424456 USDC. `repayBorrow` targets `0x39aa39c0...751f` instead of the canonical cUSDC market `0x39aa39c0...7563` (typosquatted contract).
Gold label & reason	**Label:** REJECT **Node:** adversarial.intent_manipulation **Reason:** "Intent seeks MEV-style profit (front run/back run/bundle/private mempool)."	**Label:** REJECT **Node:** adversarial.parameter_manipulation **Reason:** "Transaction parameters manipulated (address typosquatting and calldata nibble tweak)."
Arbiter output	**Decision:** REJECT (correct) **Primary node:** adversarial.intent_manipulation **Reason:** "Detected explicit MEV or manipulative execution requests." **Details:** flags "private mempool" and "skip slippage checks" as red flags.	**Decision:** REJECT (correct) **Primary node:** technical.protocol_compatibility **Reason:** "Protocol/target mismatch detected." **Details:** highlights that `repayBorrow` targets `0x39aa39c0...751f` instead of canonical `0x39aa39c0...7563`.

APPENDIX A
CASE STUDIES OF ARBITER DECISIONS

APPENDIX B
GRAPH DESIGN AND NODE CATALOG

Protocol-specific graphs. Each protocol (Compound, Uniswap, CoW) has a tailored validation graph. Compound's graph contains 19 nodes (14 rule-based, 5 semantic), Uniswap has 17 nodes (13 rule-based, 4 semantic), and CoW has 20 nodes (16 rule-based, 4 semantic). All graphs share a common core of 15 nodes, with 2-3 protocol-specific nodes per protocol (e.g., `gas_reasonableness` for Compound, `cow_order_structure` for CoW).

Node families. Nodes are organized into four families:

- **Intent Alignment**: Verify transaction parameters match stated intent
- **Technical Invariants**: Enforce protocol-level structural constraints
- **Legal Compliance**: Screen for sanctions and token legitimacy

- **Adversarial Detection**: Identify manipulative or MEV-risky patterns

Design principles. The graph topology follows three principles:

1) *Fail-fast*: All critical nodes (V_{crit}) at Level 0 or Level 1 to enable early termination
2) *Dependency ordering*: Syntactic checks before semantic checks
3) *Parallelization*: Minimize inter-level dependencies to allow concurrent node execution within each level

Table V lists the 17 core nodes shared across all three protocols. Rule-based nodes (R) enforce deterministic constraints; semantic nodes (S) require LLM reasoning. Critical nodes ($\star$) trigger immediate rejection on failure.

Protocol-specific nodes. Beyond the 15 core nodes, Compound and Cow protocol add specialized validators:

- **Compound** (2 additional nodes):
  - `gas_reasonableness`*: Validates gas limits fall within reasonable ranges for contract operations (e.g., rejects $<$50k gas for mint/redeem)
  - `consistency_analysis`: Validates cross-field consistency in multi-step lending operations
- **CoW** (3 additional nodes): noitemsep
  - `cow_order_structure`: Validates batch auction order format and structure
  - `settlement_contract`: Verifies CoW Protocol settlement contract validity
  - `zero_amount_guard`*: Rejects malicious zero-amount orders

Dependency Structure and Execution Levels Arbiter's validation graphs are organized into topological levels based on node dependencies. Nodes within each level execute in parallel, while levels execute sequentially. Table VI shows the complete Compound graph organized by level, with dependency relationships explicit.

TABLE V
CORE NODES SHARED BY COMPOUND, UNISWAP, AND COW PROTOCOLS. ALL 17 NODES APPEAR IN EVERY PROTOCOL GRAPH WITH PROTOCOL-SPECIFIC DEPENDENCY ADJUSTMENTS.

Node	Family	Type	Validation Check
Intent Alignment (10 nodes)			
`operation_matching`*	Intent	R	Verify operation type (swap/deposit/withdraw)
`token_matching`*	Intent	R	Check token symbols/addresses match intent
`amount_matching`*	Intent	R	Validate amounts within tolerance
`fee_alignment`*	Intent	R	Ensure gas/fees under stated limits
`deadline_consistency`*	Intent	R	Check deadline matches urgency
`slippage_tolerance`	Intent	R	Verify slippage bounds respected
`sequence_check`	Intent	R	Validate approve-then-action ordering
`approval_covers_primary`	Intent	R	Verify ERC-20 approval $\geq$ operation amount
`element_matching`	Intent	S	Holistic semantic alignment check
`requirement_compliance`	Intent	S	Final intent requirements verification
Technical Invariants (2 nodes)			
`format_validation`*	Technical	R	JSON structure and field type checks
`protocol_compatibility`	Technical	R	Verify contract addresses match protocol
Legal Compliance (2 nodes)			
`sanctions_screening`*	Legal	R	Check addresses against OFAC lists
`token_legitimacy`*	Legal	R	Verify tokens not flagged as scams
Adversarial Detection (3 nodes, 2 rule + 1 semantic)			
`intent_manipulation`*	Adversarial	S	Detect MEV-seeking language in intent
`mev_risky_parameters`*	Adversarial	R	Flag rushed deadlines, loose slippage
`parameter_manipulation`	Adversarial	S	Identify subtle parameter attacks

TABLE VI

COMPOUND VALIDATION GRAPH BY DEPENDENCY LEVEL. NODES WITHIN EACH LEVEL EXECUTE IN PARALLEL. SEMANTIC NODES (S) DEPEND ON RULE-BASED NODES (R) FROM PREVIOUS LEVELS.

Level	Node	Key Dependencies
Level 0: Critical Guards (7 nodes, all parallel)		
0	`operation_matching`* (R)	None
0	`format_validation`* (R)	None
0	`gas_reasonableness`* (R)	None
0	`sanctions_screening`* (R)	None
0	`token_legitimacy`* (R)	None
0	`intent_manipulation`* (S)	None
0	`mev_risky_parameters`* (R)	None
Level 1: Intent Alignment (7 nodes)		
1	`token_matching`* (R)	`operation_matching`, `format_validation`, `gas_reasonableness`
1	`amount_matching`* (R)	`operation_matching`, `format_validation`, `gas_reasonableness`
1	`fee_alignment`* (R)	`operation_matching`, `format_validation`, `gas_reasonableness`
1	`deadline_consistency`* (R)	`operation_matching`
1	`sequence_check` (R)	`operation_matching`, `format_validation`, `gas_reasonableness`
1	`approval_covers_primary` (R)	`operation_matching`
1	`protocol_compatibility` (R)	`operation_matching`, `format_validation`, `gas_reasonableness`
Level 2: Composite Checks (2 nodes)		
2	`slippage_tolerance` (R)	`operation_matching`, `token_matching`, `amount_matching`
2	`element_matching` (S)	`operation_matching`, `token_matching`, `amount_matching`, `fee_alignment`
Level 3: Advanced Semantic (1 node)		
3	`parameter_manipulation` (S)	`element_matching`, `protocol_compatibility`
Level 4: Final Compliance (2 nodes)		
4	`requirement_compliance` (S)	`element_matching`, `fee_alignment`, `slippage_tolerance`, `deadline_consistency`
4	`consistency_analysis` (S)	`element_matching`, `protocol_compatibility`, `parameter_manipulation`