INTENT2TX: BENCHMARKING LLMS FOR TRANSLATING NATURAL LANGUAGE INTENTS INTO ETHEREUM TRANSACTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Models (LLMs) are poised to revolutionize Web3 interfaces, yet current benchmarks fail to bridge the gap between high-level user intents and functionally correct on-chain execution. We present INTENT2TX, the first benchmark for intent-to-transaction translation, comprising 30,246 single-step and 1,758 multi-step instances reconstructed from 300 days of real-world Ethereum mainnet activity. Each instance pairs a transaction deterministically decoded from a successful mainnet call with a human-written or LLM-generated natural language intent, grounding language in complex, state-dependent protocol interactions across 11 DeFi categories. To validate functional correctness, we propose an execution-aware evaluation framework that employs differential state analysis on forked mainnet environments, moving beyond brittle surface-level metrics. Our evaluation of 16 state-of-the-art LLMs reveals a "reasoning-to-execution" gap: even syntactically perfect outputs frequently fail to achieve intended state transitions. INTENT2TX provides a framework for developing verifiable and autonomous agents in intent-centric Web3 ecosystems. Code and data: `https://anonymous.4open.science/r/Intent2Tx_Bench-97FF`.

1 INTRODUCTION

In the Web2 ecosystem, the transition from explicit commands to goal-oriented interfaces is well-established; for instance, a user might say "book the fastest flight to London" rather than manually interacting with fragmented underlying APIs. LLMs are now catalyzing a similar shift in Web3, evolving into the primary interface that lowers the barrier to decentralized application (dApp) interaction. By moving from a manual transaction-based model to an "intent-centric" paradigm, LLMs allow users to express complex financial goals—such as "swapping ETH for USDC and then adding liquidity to a profitable pool"—in natural language.

However, translating natural language intents into executable on-chain transactions is significantly more complex than standard Web2 tool-calling. While existing AI research in function calling and API selection provides a foundation (Guo et al., 2024b; Schick et al., 2023; Patil et al., 2025), Web3 introduces three unique hurdles: (1) Structural Heterogeneity, where diverse DeFi protocols use radically different router logics for similar purposes (Carter & Jeng, 2021); (2) Low Readability, as transaction signatures often involve nested tuples and raw hex data that lack the semantic clarity of RESTful APIs (Kitzler et al., 2023); and (3) State-Dependency, where the validity of a transaction is inextricably linked to the real-time, irreversible state of the Ethereum Virtual Machine (EVM) (Hu et al., 2023).

Among benchmarks dealing with Web3 intents, two works are closely related to ours: the Intent-Tx-18K dataset (Yao et al.) and EVM-QuestBench (Yang et al., 2026). However, the former is primarily designed to train intent-transaction consistency detectors; it relies on raw hex data from limited protocols (e.g., Uniswap, CoW (Xu et al., 2023a)) without human-readable function signatures, rendering it unsuitable for generative tasks. The latter, while targeting transaction generation, provides inputs that resemble explicit, stepwise execution traces rather than goal-oriented intents. Such an approach fails to reflect real-world LLM applications where the objective is to abstract away, not

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

dictate, the underlying technical steps. Consequently, there is a pressing need for a benchmark that bridges **high-level intent** with **readable and executable on-chain calls**.

To address these limitations, we introduce the **Intent-to-Transaction (INTENT2TX) benchmark**. We define the task as follows: given a high-level natural language intent (describing the what, not the how), the model must output a structured on-chain call comprising the contract address, function name, parameters, and value. Our benchmark encompasses two difficulty levels: **Single-step** (an atomic operation that can be expressed in a single transaction) and **Multi-step** (a series of atomic operations requiring planning). The transactions are deterministically decoded from 300 days of Ethereum mainnet transaction logs (2025.03–2026.01), ensuring they reflect the most current and diverse protocol landscape, while the paired intents come from two sources. The first is Decentralized Autonomous Organizations (DAOs)(Wang et al., 2019), on-chain communities that govern a protocol by voting on proposals: each proposal states its goal in natural language and carries the exact calls to execute once approved, so it naturally pairs a human-written intent with its transaction. Since such proposals cannot reflect the full transaction distribution, the second source supplements them with LLM reverse-generated intents under manual validation. We have systematically evaluated the performance of state-of-the-art LLMs on this task, as well as the dataset's utility as a fine-tuning corpus.

Our contributions are as follows: (1) First Intent-to-Transaction Benchmark: We present the first comprehensive benchmark that maps high-level intents to both readable and executable transactions, covering a wide array of DeFi protocols. (2) General Extraction Pipeline: We propose a method for constructing intent-transaction pairs from real on-chain logs, combining human-written governance intents with LLM reverse generation and providing a scalable way to synthesize training data from the blockchain's history. (3) Execution-Aware Evaluation and Systematic Analysis: We introduce a simulation-based evaluation protocol using differential state analysis on forked mainnet environments, and conduct a comprehensive study of 16 LLMs, revealing that no model is reliable without demonstrations (direct pass@1 ≤ 0.11), that code specialization does not help, and that fine-tuning on common operations induces both positive and negative transfer to long-tail categories.

2 BACKGROUND AND RELATED WORK

DeFi Primitives and the Intent-centric Paradigm. The DeFi ecosystem has evolved into a diverse landscape of financial primitives, spanning Decentralized Exchanges (DEXs), lending markets, liquid staking, and yield aggregators (Alamsyah et al., 2024). Traditionally, interacting with these protocols follows a transaction-based (imperative) model, where users must explicitly dictate the execution path—specifying contract addresses, function selectors, and ABI-encoded parameters. This requires a deep understanding of the EVM mechanics (Hildenbrandt et al., 2018). Recently, the intent-centric (declarative) paradigm (Goes et al., 2023) has emerged to abstract this complexity. Unlike transactions that define how to execute (e.g., "call swapExactTokensForTokens on Uniswap V2"), intents focus on what the desired outcome is (e.g., "exchange 1 ETH for at least 2500 USDC"). While infrastructure like account abstraction (ERC-4337 (Buterin et al., 2021)) and off-chain solvers handle the execution of signed intents, a critical intelligence gap remains: the semantic translation of fuzzy natural language into the precise, structured calls required by these frameworks. Our work targets this gap, evaluating LLMs' ability to act as the primary translator across diverse Web3 operational categories.

Benchmarks for Generating Structured Data. The task of transforming natural language into structured formats is a cornerstone of LLM evaluation. Current benchmarks predominantly focus on Text-to-SQL, code generation (e.g., Spider (Yu et al., 2018), HumanEvalPro (Yu et al., 2024)), and tool-use/API calling (e.g., StableToolBench (Guo et al., 2024b)). In the context of blockchain transactions, however, the structured output space becomes substantially more demanding. Transaction schemas involve high-dimensional parameters, including 256-bit integers and deeply nested tuples (Lin et al., 2020), while also requiring strict adherence to domain-specific constraints such as slippage (Chemaya et al., 2024) and deadline (Pacheco et al., 2023), where even minor deviations can invalidate execution. In addition, similar financial operations may map to heterogeneous function signatures across different DeFi protocols (He et al., 2020), further increasing the complexity of generation. Our work extends structured data generation into the highly adversarial and high-stakes domain of Web3.

Conventions for Evaluating Executable Output. Reference-based metrics (e.g., BLEU) are increasingly replaced by execution-based evaluation in code synthesis to ensure functional correctness (Chen et al., 2021; Austin et al., 2021). In blockchain settings, this paradigm is essential: transaction validity depends on the on-chain state at execution time (e.g., balances, protocol conditions, and temporal constraints), and even structurally correct outputs may fail or behave incorrectly outside their original environment. Most existing benchmarks, however, assume stateless, sandboxed functions and do not capture shared external state or irreversible actions (Guo et al., 2024a). To address this, we introduce Differential State Analysis, which evaluates LLM outputs via simulated execution on a forked Ethereum mainnet and compares post-execution state changes (e.g., balance deltas) with ground truth.

3 INTENT2TX DATASET

3.1 OVERVIEW

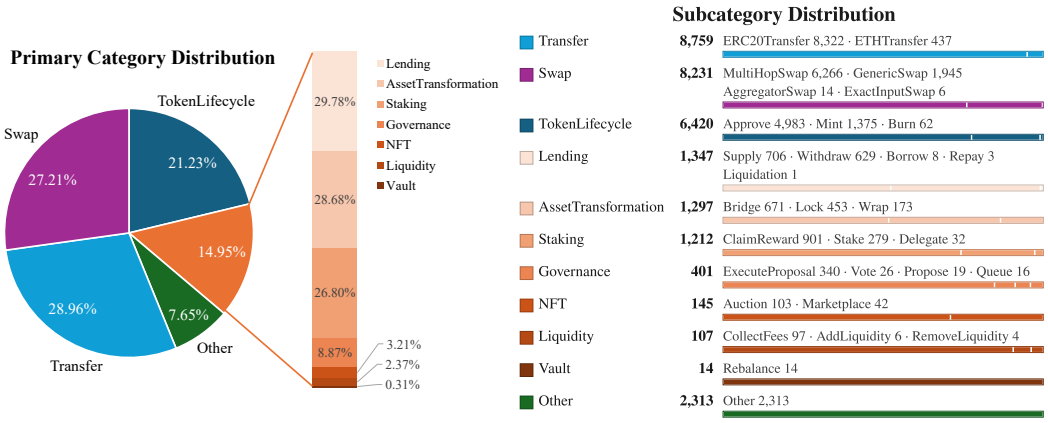


Figure 1: Primary (left) and subcategory (right) distribution of the Single-step instances. Multi-step instances (composed actions or multi-call proposals, Section 3.3) have no categories of their own.

The INTENT2TX dataset consists of **30,246** Single-step instances and **1,758** Multi-step instances, each derived from real-world Ethereum mainnet transaction logs spanning from March 2025 to January 2026. The transactions are deterministically decoded from these logs, while the natural language intents are either written by humans in DAO governance proposals or reverse-generated by an LLM (Section 3.3).

Each instance maps a high-level user intent to an executable transaction schema. For a single-step instance, the input is a natural language request describing a financial or operational goal (e.g., "Swap 1,000 USDC for ETH with minimal slippage"). The corresponding output is a structured on-chain call which contains contract name, address, function, parameters and ETH value.

As illustrated in Figure 1, the primary category distribution is dominated by core DeFi activities (Transfer, Swap, and TokenLifecycle), in line with the most frequent behaviors of retail users, followed by a significant "long-tail" of more complex protocols such as Lending, Staking, Governance, and Liquidity Management. The subcategory distribution (Figure 1, right) further reveals high-granularity actions like MultiHopSwap, Bridge, ClaimReward, and ExecuteProposal. By grounding our benchmark in 300 days of live on-chain history, INTENT2TX reflects real-world parameter distributions, varying protocol architectures, and evolving contract patterns, posing a challenge for LLM-based intent translation.

3.2 TASK DEFINITION

Problem Formalization. The goal of the INTENT2TX task is to map a high-level natural language intent I into a sequence of structured, executable blockchain transactions t. Formally, the task is

defined as finding a mapping function $f(I) \rightarrow [t_1, t_2, \ldots, t_k]$, where $k = 1$ for atomic, single-step operations and $k > 1$ for complex, multi-step workflows. Each transaction t_i must contain all necessary metadata and parameters to be successfully executed on-chain.

Transaction Schema. To ensure the output is both human-readable and machine-executable, we define a standardized JSON-based Transaction Schema for t_i:

```
{ "contract": string,            // contract name (e.g., "UniswapV2Router02")
  "contract_address": string,    // contract address (e.g., "0x7a250d5630...")
  "function": string,            // function name (e.g., "transfer")
  "params": { "<param_name>": { "type": string, "val": any }, ... },
                                 // Solidity type (e.g., "uint256") and inferred value
  "value": float }               // Ether amount sent with the transaction (in ETH)
```

This design prioritizes (1) Generality: explicit function names and types remove the need for pre-loaded ABIs; (2) Execution-Readiness: `contract_address` and typed `params` convert directly into raw calldata for simulation; and (3) LLM-Friendliness: semantic names (*contract*, *function*) instead of raw hex codes leverage the pre-trained linguistic knowledge of LLMs.

Task Difficulty. We categorize the benchmark based on whether an intent can be fulfilled within a single on-chain transaction. *Single-step*: The intent can be expressed and executed using a single Ethereum transaction ($k = 1$). This setting evaluates the model's ability to identify the correct contract function and infer precise parameters (e.g., an ERC20 transfer or a token swap). *Multi-step*: The intent requires a sequence of multiple transactions ($k > 1$), where intermediate steps must be correctly ordered and parameterized (e.g., *Approve* $\rightarrow$ *Swap* $\rightarrow$ *Add Liquidity*). This setting tests the model's ability to decompose high-level goals into executable steps and handle inter-transaction dependencies.

3.3 Data Construction Pipeline

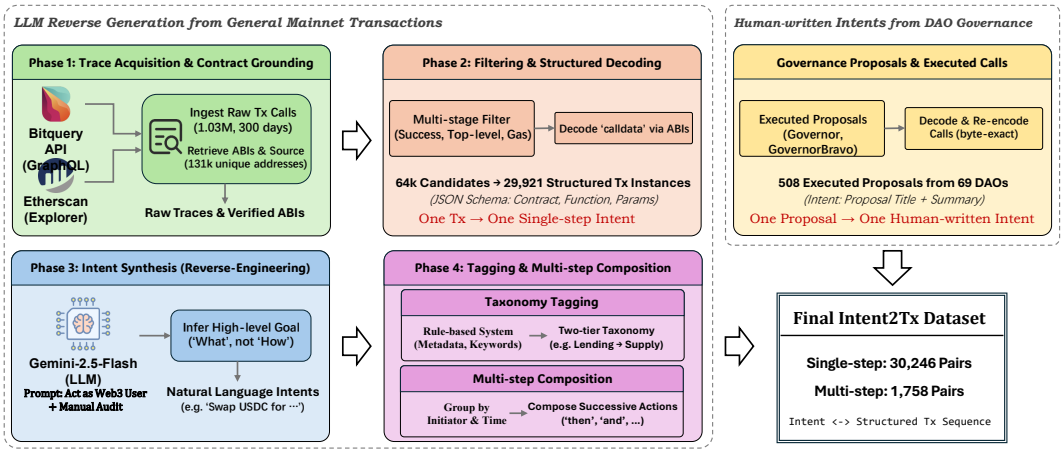


Figure 2: Construction of INTENT2TX from **two complementary intent sources**: human-written DAO governance proposals (right) and LLM reverse generation from mainnet transactions (left).

Human-written Intents from DAO Governance. In on-chain governance (Figure 2, right), a human-written proposal and its executable calls are authored together. We collected the OpenZeppelin Governor and GovernorBravo proposals(Kovacs & Costa, 2025) executed in the same time window with one to five calls, using the proposal title and first paragraph as the intent; each call is decoded with the target's verified ABI and re-encoded to match the on-chain calldata byte by byte. This yields 508 proposals from 69 DAOs (325 Single-step and 183 Multi-step instances). However, these intents are limited in quantity and mainly cover treasury transfers and protocol configuration rather than everyday swaps, lending, or staking, so they cannot fully represent the distribution of Ethereum transactions.

LLM Reverse Generation. We therefore supplement them with intents reverse-generated from general mainnet transactions. This construction follows the four-phase pipeline in Figure 2 (left), transforming raw, low-level blockchain traces into high-level intent-transaction pairs.

Phase 1: Trace Acquisition and Contract Grounding. We initially ingested over 1.03 million raw Ethereum transaction calls spanning a 300-day window via the Bitquery GraphQL API. To ensure the transactions could be decoded into a structured format, we extracted 131,273 unique contract addresses and retrieved their corresponding source code and ABIs from Etherscan, a well-known blockchain explorer.

Phase 2: Filtering and Structured Decoding. Raw traces are often cluttered with system calls, failed transactions, and "spam" interactions with zero functional value (Zhang et al., 2020). We applied a multi-stage filter to retain only successful and top-level atomic calls that interacted with verified contracts and consumed enough gas. Utilizing the retrieved ABIs, we decoded the raw `calldata` of 64,424 candidate transactions into the **Transaction Schema** defined in the previous section. This resulted in 29,921 structured transaction instances.

Phase 3: Intent Synthesis via Reverse-Engineering. To generate natural language user intents, we followed prior work that uses LLMs for dataset construction (Patil et al., 2024; Xu et al., 2023b; Liu et al., 2023). We employed a "reverse-engineering" approach using `gemini-2.5-flash`. Rather than asking the model to simply describe the code, we prompted it to act as a Web3 user and infer the high-level goal (*intent*) behind the structured schema. For common actions (e.g., Swaps), the model was instructed to use colloquial language; for niche DeFi protocols, it maintained technical precision. Only the intents are model-generated; all transaction targets come from the deterministic decoding in Phase 2. To ensure quality, two authors experienced with Ethereum transactions and DeFi protocols independently audited 900 stratified examples against their decoded transactions, with disagreements resolved by a third author; 6.1% of the audited examples were initially rejected, then revised and re-checked before inclusion (Cohen's $\kappa = 0.85$; see Appendix B). The prompt, tuned over several rounds of small-scale trials, is given in Appendix A.1.

Phase 4: Taxonomy Tagging and Multi-step Composition. The final dataset was enriched with a two-tier taxonomy (e.g., *Lending* $\rightarrow$ *Supply*). We implemented a rule-based tagging system based on contract metadata and function keywords to categorize all **Single-step** instances. The labeling results are shown in Figure 1. To construct the **Multi-step** benchmark, we grouped transactions by the initiator's (`from`) address and execution timestamp. Successive actions performed by the same user within a short window (e.g., *Approve* followed by *AddLiquidity*) were composed into complex intents using natural conjunctions ("then", "and"), resulting in 1,575 multi-step planning tasks. Examples of data from the benchmark are shown in Appendix E.

4 Experiments and Evaluation Framework

We evaluate INTENT2TX through four experimental axes: benchmarking 16 diverse LLMs, analyzing data scaling laws, assessing cross-category generalization, and validating functional correctness via on-chain simulation. Our model suite includes general-purpose LLMs and code-optimized variants (e.g., GPT-5.2 Codex, Qwen3 Coder) to investigate domain-specific synthesis capabilities. All experiments were conducted on a high-performance server equipped with 192-core AMD Ryzen Threadripper PRO 9995WX CPUs, an NVIDIA RTX PRO 6000 GPU, and 256GB of RAM. The full list of evaluated models is detailed in Appendix G.

4.1 Research Questions and Experimental Settings

RQ1: How well do LLMs perform on INTENT2TX? We compare two inference settings: (1) *Direct inference*, where the model generates transactions from the system prompt (Appendix A.2), user intent, and an address book (Appendix A.3) without in-context examples; and (2) *Oracle retrieval-augmented inference*, where the top-$k = 3$ similar instances are retrieved as demonstrations. Similarity is computed via Jaccard overlap over <contract, function> pairs, with a fallback to down-weighted text similarity. Because this similarity uses the gold <contract, function> labels of the test instance, this setting is an oracle-routing upper bound on the benefit of retrieval rather than a deployable end-to-end setting. All models are evaluated on both single-transaction and multi-transaction tasks under the same stratified sampling scheme.

RQ2: How does training data scale affect performance? We conduct a scaling experiment to study how training data size affects generalization in intent-to-transaction generation. We fine-tune a Qwen3-14B model (Yang et al., 2025) using parameter-efficient LoRA (Li et al., 2023) adaptation

implemented in LLaMA-Factory (Zheng et al., 2024) on progressively larger subsets of the training data, with sizes of 200, 800, 2k, 8k, 14k, 20k, and 28,776 instances.

To ensure a balanced distribution, all training subsets are constructed via stratified sampling over intent categories. A fixed set of 1,470 held-out instances serves as the test set across all configurations.

Our hypothesis is that increasing the scale and diversity of training data improves the model's ability to generalize to unseen intents and transaction compositions. This setup allows us to analyze scaling trends under a controlled data distribution and consistent evaluation protocol.

RQ3: Can models generalize to unseen operation categories? We evaluate whether models fine-tuned on common operation categories can generalize to rarer and more complex categories. Specifically, we train on a subset of frequent categories, including {Transfer, Swap, TokenLifecycle, Lending, Staking, AssetTransformation}, and test on a disjoint set of less frequent categories: {Governance, Liquidity, NFT, Vault, Other}. The fine-tuning configuration follows the same setup as in the scaling experiment. By enforcing a strict category split, this experiment isolates cross-category transfer ability and evaluates whether models can compose previously learned primitives to handle unseen operation types.

RQ4: How well do generated transactions execute on-chain? We note that multiple structured transaction sequences may achieve the same user intent, even if they differ from the reference output (e.g., invoking alternative contracts or equivalent functions). To account for this, we design a simulation-based evaluation framework that measures execution-level equivalence rather than exact match.

Given both the reference (G) and model-generated (P) transactions, we first extract the contract address, function, and parameters, and reconstruct the corresponding Ethereum transactions via ABI encoding. We then execute both transactions on a forked Ethereum mainnet using `anvil` (Tudorache, 2025), where the fork point is set to one block prior to the original transaction.

We evaluate two criteria: (1) *executability*, i.e., whether the transaction can be successfully executed, and (2) *state consistency*, defined as whether the resulting account-level state changes (e.g., token balance deltas) are consistent between G and P, within a tolerance threshold ($< 1\%$).

This framework enables a more faithful assessment of functional correctness by capturing semantic equivalence in execution outcomes, rather than relying solely on syntactic matching.

4.2 Evaluation Metrics

To assess the functional alignment between the prediction P and the gold standard G, we define a multi-dimensional scoring framework. This framework evaluates the model's performance across structural, logical, and parameter-level accuracy. When fork simulation is available, executability and state equivalence (RQ4) are the primary measures of functional correctness; the static metrics below serve as complementary diagnostics for two reasons. First, execution success and Pass@1 are all-or-nothing at the instance level, so a near-miss and a completely wrong transaction can both score zero, whereas the static components distinguish them and localize the failure (output format, contract/function routing, or argument binding). Second, fork execution is expensive and can fail for environmental reasons unrelated to the model, so static metrics enable diagnosis on the full dataset and across all experiments.

Format Score (S_{fmt}) measures basic instruction-following by verifying JSON validity: $S_{fmt}(G, P) = 1$ if P is a valid JSON object and 0 otherwise.

Logic Score (S_{log}) evaluates the model's ability to identify the correct interaction target, including the contract address, function signature, and contract name:

$$S_{log}(G, P) = 0.4 \cdot \mathbb{I}(addr_P = addr_G) + 0.4 \cdot \mathbb{I}(func_P = func_G) + 0.2 \cdot \mathbb{I}(name_P = name_G)$$

where $\mathbb{I}(\cdot)$ is the indicator function.

Param Score (S_{prm}) assesses the granularity of parameter extraction. Let $K_{all} = K_G \cup K_P \cup \{Value_{eth}\}$ represent the union of standard and predicted parameter keys. The score is calculated as $S_{prm}(G, P) = \frac{1}{|K_{all}|} \sum_{k \in K_{all}} s(k)$. Individual parameter accuracy $s(k)$ is defined by a base score of 0.4 for key existence, with additional increments of 0.3 for correct type matching and 0.3 for value matching.

Pass@1 (S_{pass}) serves as a strict "Hard Match" indicator for end-to-end usability, requiring near-perfect logic and perfect parameter alignment: $S_{pass}(G, P) = \mathbb{I}(S_{log} \geq 0.8 \wedge S_{prm} = 1.0)$.

The **Final Weighted Score** for an instance i combines these sub-metrics to reflect overall reliability:

$$S_{final} = \omega_1 S_{fmt} + \omega_2 S_{log} + \omega_3 S_{prm} + \omega_4 S_{pass}$$

Following empirical requirements for stable on-chain execution, we assign weights of $\omega_1 = 0.1, \omega_2 = 0.3, \omega_3 = 0.2$, and $\omega_4 = 0.4$.

For Multi-step tasks where G contains T sequential actions, the metrics are extended by taking the arithmetic mean of each step, $S_{metric_avg} = \frac{1}{T}\sum_{t=1}^{T} S_{metric}(G_t, P_t)$. The total performance is reported as the average score across all N samples in the benchmark.

5 Results and Analysis

5.1 Retrieval Augmentation Significantly Improves Performance (RQ1)

Table 1: Results under direct and oracle retrieval settings: format (F: S_{fmt}), logic (L: S_{log}), parameter (P: S_{prm}), pass@1 (A: S_{pass}), and Final (S_{final}).

	Direct, Single-step					Direct, Multi-step					Oracle Retrieval, Single-step					Oracle Retrieval, Multi-step				
Model	F	L	P	A	Final	F	L	P	A	Final	F	L	P	A	Final	F	L	P	A	Final
Claude Haiku 4.5	0.97	0.44	0.47	0.01	0.33	0.87	0.39	0.45	0.04	0.31	1.00	0.85	0.86	0.35	0.67	0.97	0.71	0.74	0.27	0.57
Claude Opus 4.5	0.99	0.53	0.51	0.01	0.37	0.98	0.55	0.55	**0.11**	**0.42**	1.00	0.87	0.90	0.41	**0.70**	0.96	0.73	0.74	0.29	0.58
Codestral	0.93	0.43	0.42	0.01	0.31	0.80	0.38	0.40	0.05	0.29	0.94	0.76	0.79	0.32	0.61	0.84	0.68	0.67	0.24	0.52
DeepSeek 3.2	0.96	0.45	0.50	0.04	0.35	0.94	0.40	0.39	0.05	0.31	0.99	0.82	0.85	0.33	0.65	0.97	0.73	0.76	**0.32**	**0.59**
GLM 4.6	0.99	0.47	0.46	0.02	0.34	0.88	0.41	0.41	0.02	0.30	1.00	0.86	0.88	0.38	0.69	0.96	0.72	0.75	0.22	0.55
GLM 4 32B	0.95	0.40	0.42	0.02	0.31	0.88	0.35	0.38	0.01	0.27	0.99	0.82	0.85	0.35	0.65	0.91	0.66	0.66	0.20	0.50
GPT 5.2	1.00	0.48	0.54	**0.06**	**0.38**	0.88	0.39	0.45	0.04	0.31	1.00	0.81	0.86	0.39	0.67	0.94	0.63	0.68	0.22	0.51
GPT 5.2 Codex	1.00	0.37	0.36	0.01	0.29	0.60	0.30	0.33	0.03	0.23	1.00	0.79	0.83	0.38	0.66	0.86	0.65	0.70	0.32	0.55
Gemini 2.5 Flash	1.00	0.41	0.49	0.01	0.32	0.96	0.41	0.50	0.04	0.33	1.00	0.85	0.89	**0.42**	**0.70**	1.00	0.75	0.77	0.27	0.58
Gemini 2.5 Pro	1.00	0.50	0.54	0.05	**0.38**	0.95	0.48	0.49	0.10	0.38	0.99	0.84	0.87	0.41	0.69	0.95	0.72	0.75	0.31	**0.59**
Kimi 2	0.99	0.47	0.48	0.01	0.34	0.94	0.50	0.46	0.03	0.35	1.00	0.83	0.87	0.37	0.67	0.96	0.70	0.73	0.24	0.55
Llama 3.1 8B	0.91	0.33	0.32	0.00	0.25	0.90	0.32	0.37	0.00	0.26	0.81	0.64	0.65	0.19	0.48	0.79	0.60	0.58	0.10	0.41
Llama 3.3 70B	0.87	0.38	0.39	0.00	0.28	0.89	0.36	0.41	0.02	0.29	0.93	0.74	0.78	0.26	0.58	0.92	0.66	0.70	0.19	0.51
Ministral 14B	0.39	0.15	0.17	0.00	0.12	0.46	0.18	0.21	0.01	0.15	0.81	0.67	0.68	0.25	0.52	0.69	0.48	0.49	0.16	0.37
Qwen3 235B	1.00	0.41	0.39	0.01	0.30	0.90	0.35	0.41	0.03	0.29	0.94	0.78	0.82	0.37	0.64	0.96	0.66	0.69	0.28	0.54
Qwen3 Coder	1.00	0.40	0.43	0.02	0.31	0.93	0.33	0.38	0.03	0.28	0.93	0.78	0.80	0.35	0.63	0.85	0.63	0.64	0.23	0.49

Table 1 reports results under direct and oracle retrieval-augmented settings for both single-step and multi-step tasks. Retrieval consistently yields substantial improvements across all models and metrics; since the retrieval setting uses gold routing labels, these gains should be read as an upper bound.

Retrieval mainly improves routing and argument binding: S_{log} and S_{prm} rise by 0.30–0.40 for most models (e.g., Claude Haiku 4.5: 0.44→0.85 and 0.47→0.86 on single-step tasks), and pass@1 rises to 0.30–0.42 for strong models. Without demonstrations, however, no model is reliable: the best direct pass@1 is only 0.06 on single-step and 0.11 on multi-step tasks, even though most models achieve near-perfect format scores. The benchmark thus measures far more than structured-output syntax. Multi-step tasks remain harder under retrieval (e.g., 0.70 vs. 0.58 Final for Claude Opus 4.5), indicating that composing multiple transactions is a key bottleneck (see Appendix F).

Code specialization does not help and can even hurt. Under direct inference, GPT 5.2 Codex trails GPT 5.2 (Final 0.29 vs. 0.38). Its outputs show that the gap stems from abstention rather than weaker code ability: in 26% of single-step cases, Codex returns an empty object or an `error` message requesting missing details (e.g., the contract address or calldata), compared with at most 1.1% for any other model. With retrieval, this rate falls to 2.9% and Codex matches GPT 5.2 (0.66 vs. 0.67). Such caution may be desirable in high-stakes settings but is penalized by reference-based scoring, suggesting that justified abstention deserves explicit treatment in future evaluations.

5.2 Scaling Training Data Yields Consistent Gains (RQ2)

Figure 3 shows that increasing training data generally improves downstream performance, suggesting that more diverse intent–transaction pairs enhance generalization. The largest gains appear in logic

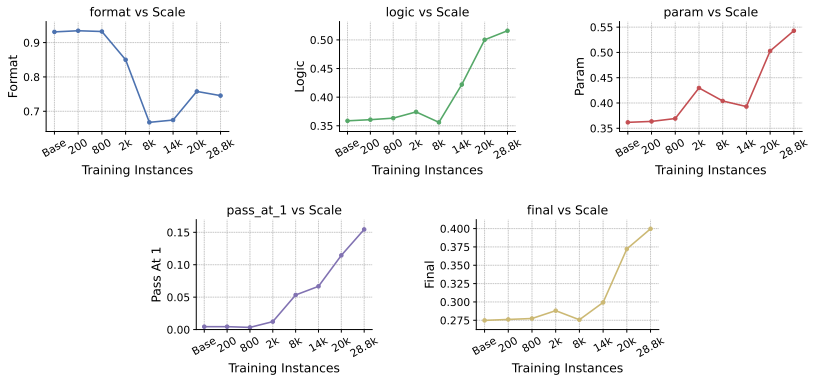


Figure 3: Scaling behavior across training set sizes. Top: Format, Logic, and Param; bottom: Pass@1 and Final.

(S_{log}) and parameter accuracy (S_{prm}), indicating that larger corpora help models better recover contract-function mappings and parameter values.

We also observe a slight drop in format accuracy (S_{fmt}) as data scale increases. This likely reflects a trade-off in dataset complexity: smaller subsets contain simpler and more uniform instances, whereas larger subsets introduce more diverse transaction structures and parameter schemas, making strict JSON adherence harder. Unlike typical saturation behavior, performance continues to improve at larger scales, especially for pass@1 (S_{pass}) and the final score. Overall, these results suggest that data quality and diversity are as important as scale for intent-to-transaction generation.

5.3 Models Exhibit Limited Generalization to Unseen Categories (RQ3)

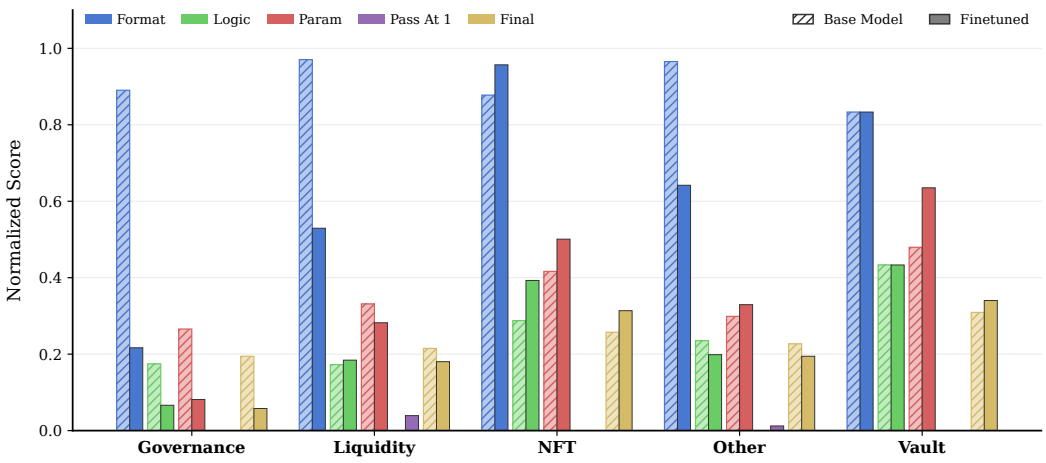


Figure 4: Generalization performance comparison across different categories. The *hatched* bars represent the Base model, while the *solid* colored bars denote the cross-category fine-tuned model.

Figure 4 compares the base model with the cross-category fine-tuned model on five held-out operation categories. Transfer is mixed: NFT and Vault, which share mint, burn, deposit, and withdrawal primitives with the training categories, improve in logic, parameter, and final scores, whereas Final drops on Governance (0.195→0.058), Liquidity (0.215→0.180), and Other (0.227→0.195).

These declines are driven mainly by collapsing format accuracy (e.g., Governance: 0.890→0.217). Inspecting the outputs reveals one dominant failure mode: the fine-tuned model tries to reproduce opaque `bytes` payloads (e.g., calldata described as "pre-calculated data") and falls into degenerate repetition until the generation limit, never closing the JSON object. Such outputs account for 92% of

its 1,152 format failures; on instances whose reference contains a `bytes` parameter, the failure rate rises from 5.2% to 59.1% (vs. 4.5% to 15.8% otherwise). As 89% of Governance instances contain such parameters, this category collapses most.

Positive transfer nevertheless coexists within the same run. The base model solves none of the 2,855 held-out cases under pass@1, whereas the fine-tuned model newly solves 31 (27 in Other, 4 in Liquidity). For example, for a request to resolve a `PermissionedDisputeGame`, the base model predicts the plausible but nonexistent `resolveDispute`, while the fine-tuned model selects the ABI-valid `resolve`. Fine-tuning on frequent operations thus transfers category-agnostic skills but destabilizes generation on rare, long-payload schemas.

5.4 Execution-based Evaluation Reveals the Potential of Models (RQ4)

Table 2: Simulation execution performance. Exec denotes executability, and StateEq denotes state equivalence with the ground truth. All results are computed over instances with successful ground-truth execution. Retrieval denotes the oracle retrieval setting (Section 4).

	Direct		Retrieval			Direct		Retrieval	
Model	Exec	StateEq	Exec	StateEq	Model	Exec	StateEq	Exec	StateEq
Claude Haiku 4.5	0.572	0.513	0.681	0.647	Gemini 2.5 Flash	0.662	**0.616**	0.713	0.692
Claude Opus 4.5	0.513	0.472	0.729	**0.706**	Gemini 2.5 Pro	**0.681**	0.605	0.733	0.702
Codestral	0.529	0.481	0.634	0.626	Kimi 2	0.627	0.560	0.673	0.655
DeepSeek 3.2	0.542	0.480	0.669	0.635	Llama 3.1 8B	0.221	0.208	0.505	0.492
GLM 4.6	0.655	0.592	0.704	0.672	Llama 3.3 70B	0.323	0.310	0.592	0.573
GLM 4 32B	0.427	0.377	0.662	0.643	Ministral 14B	0.198	0.187	0.537	0.523
GPT 5.2	0.618	0.541	**0.744**	0.697	Qwen3 235B	0.447	0.409	0.673	0.653
GPT 5.2 Codex	0.437	0.394	0.720	0.697	Qwen3 Coder	0.554	0.520	0.721	0.697

Table 2 reports execution-based results, measuring executability (Exec) and state equivalence (StateEq) with the ground truth. This setup evaluates whether generated transactions can actually fulfill user intent on-chain beyond surface-level correctness.

We observe a clear gap between syntactic validity and execution success. Even strong models achieve only moderate performance under direct inference (Exec/StateEq $\approx$ 0.4–0.6), indicating that structurally correct outputs do not guarantee correct on-chain behavior. Retrieval augmentation consistently improves both metrics across models, with top systems (e.g., GPT 5.2, Gemini 2.5 Pro) reaching over 0.73 in Exec and around 0.70 in StateEq, suggesting that grounding generation in relevant transaction patterns reduces critical errors.

However, a non-trivial gap between Exec and StateEq persists: many transactions execute successfully but fail to produce the intended state changes, indicating that models often approximate rather than fully realize the desired behavior.

Overall, execution-based evaluation provides a more faithful measure of intent fulfillment than static matching metrics.

6 Conclusion and Future Work

We presented Intent2Tx, a benchmark that pairs transactions decoded from 300 days of Ethereum mainnet activity with human-written governance intents and LLM reverse-generated intents, together with an execution-aware evaluator based on forked-mainnet simulation. Across 16 LLMs, no model translates intents reliably without demonstrations (direct pass@1 $\leq$ 0.11), code specialization does not help, and fine-tuning on common operations transfers category-agnostic skills but destabilizes generation on rare, long-payload schemas. Because incorrect transactions can cause irreversible financial losses, simulation-based verification should serve not only as an evaluation metric but also as a safety guardrail before execution. Human-written intents still cover mainly governance actions and the benchmark is limited to the Ethereum mainnet; future work will add human-written intents for everyday DeFi operations, extend to other EVM chains, and train with execution feedback.

STATEMENTS

AI USE STATEMENT

In this work, we used generative AI tools for the following tasks that require disclosure. (1) *Generating synthetic data.* Apart from the human-written intents collected from DAO governance proposals, the natural language intents in INTENT2TX were generated by `gemini-2.5-flash` from deterministically decoded mainnet transactions, using the fixed prompt in Appendix A.1 (Section 3.3). The transaction targets themselves were not generated by AI. The authors manually audited 900 stratified generated intents and revised all rejected ones (Appendix B), and we analyze potential bias from the generator in Appendix C. (2) *Supporting data analysis.* An AI coding assistant was used to help write scripts that analyze our experimental logs, such as categorizing abstention outputs and format failures in the analyses of Table 1 and Figure 4. The authors checked all reported statistics against the raw logs. (3) *Implementing methods.* The data-construction and evaluation code was written manually by the authors with AI-assisted code completion; all code was reviewed and tested by the authors. We have not used generative AI tools to develop the benchmark methodology, design the experiments, or formulate hypotheses, and tasks involving theoretical models, mathematical claims, proofs, or translation are not applicable to this work. Note that large language models are also the objects of study in this work: we evaluate 16 LLMs on INTENT2TX (Appendix G), which does not constitute AI assistance to the research itself. Additionally, we used generative AI tools to edit the paper for readability and to help revise parts of the text. We have reviewed all AI-assisted work, and all AI-edited text was checked and revised by the authors. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

Data source and privacy. INTENT2TX is built from publicly available Ethereum mainnet transactions obtained through the Bitquery API, together with verified contract source code and ABIs from Etherscan, and from executed DAO governance proposals recorded on-chain. Ethereum addresses are pseudonymous but may be linkable to real-world identities. We retain addresses because they are required to construct and execute transactions, but we do not collect any off-chain personal information and do not attempt to link addresses to individuals; the released data contain no information beyond what is already public on-chain. The human-written intents are taken from public governance proposals; all other intents are LLM-generated and do not originate from real users. The dataset and code are released under the Apache License 2.0.

Human involvement. The manual validation of generated intents (Appendix B) was carried out by the authors. No crowdworkers or external participants were involved, and no personal data were collected from annotators.

Potential harms and safe use. Incorrect transactions can cause irreversible financial loss. All model outputs in this work were executed only in local forked environments, and no transaction was broadcast to the Ethereum mainnet. Our results show that current LLMs cannot reliably translate intents into correct transactions; we therefore caution against deploying LLM-generated transactions without simulation-based verification and explicit user confirmation. As with other agent benchmarks, progress on INTENT2TX could also facilitate automated on-chain agents that are misused. We consider this risk limited, because the benchmark contains only successful, ordinary interactions with verified contracts, filters out spam transactions, and contains no exploit techniques.

Bias and limitations. Most intents are English sentences written by a single LLM generator, and the human-written intents cover mainly governance actions; the benchmark covers only the Ethereum mainnet and only verified contracts. Results may therefore not transfer to human-written requests, other languages, other blockchains, or unverified contracts, and should be interpreted within this scope.

REPRODUCIBILITY STATEMENT

We release the INTENT2TX dataset and our code at the anonymous repository linked in the abstract, including the data-construction scripts, the inference and scoring scripts, the fine-tuning configurations, and the fork-based simulation evaluator. Section 3.3 describes each phase of the data-construction

pipeline, and Appendices B–D report the human validation procedure and dataset statistics. The prompts used for intent generation and inference, and the address book provided to models, are given in Appendices A.1–A.3. Section 4 specifies the inference settings, the fine-tuning setup, the simulation protocol, the evaluation metrics, and the hardware; the evaluated models are listed in Appendix G. All models are queried with temperature 0. Results of proprietary API models may still vary slightly across model versions and over time.

REFERENCES

Andry Alamsyah, Gede Natha Wijaya Kusuma, and Dian Puteri Ramadhani. A review on decentralized finance ecosystems. *Future Internet*, 16(3):76, 2024.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Vitalik Buterin, Yoav Weiss, Dror Tirosh, Shahaf Nacson, Alex Forshtat, Kristof Gazso, and Tjaden Hess. Erc-4337: Account abstraction using alt mempool. Technical report, Ethereum Improvement Proposals, 2021.

Nic Carter and Linda Jeng. Defi protocol risks: The paradox of defi. *Regtech, suptech and beyond: innovation and technology in financial services" riskbooks–forthcoming Q*, 3, 2021.

Nir Chemaya, Dingyue Liu, Robert McLaughlin, Nicola Ruaro, Christopher Kruegel, and Giovanni Vigna. The power of default: Measuring the effect of slippage tolerance in decentralized exchanges. In *International Conference on Financial Cryptography and Data Security*, pp. 192–208. Springer, 2024.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Christopher Goes, Awa Sun Yin, and Adrian Brink. Anoma: a unified architecture for full-stack decentralised applications. *Anoma Research Topics, Aug*, 2023.

Zhen Guo, Adriana Meza Soria, Wei Sun, Yikang Shen, and Rameswar Panda. Api pack: A massive multi-programming language dataset for api call generation. *arXiv preprint arXiv:2402.09615*, 2024a.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 11143–11156, 2024b.

Ningyu He, Lei Wu, Haoyu Wang, Yao Guo, and Xuxian Jiang. Characterizing code clones in the ethereum smart contract ecosystem. In *International Conference on Financial Cryptography and Data Security*, pp. 654–675. Springer, 2020.

Everett Hildenbrandt, Manasvi Saxena, Nishant Rodrigues, Xiaoran Zhu, Philip Daian, Dwight Guth, Brandon Moore, Daejun Park, Yi Zhang, Andrei Stefanescu, et al. Kevm: A complete formal semantics of the ethereum virtual machine. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, pp. 204–217. IEEE, 2018.

Xiaowen Hu, Bernd Burgstaller, and Bernhard Scholz. Evmtracer: dynamic analysis of the parallelization and redundancy potential in the ethereum virtual machine. *IEEE Access*, 11:47159–47178, 2023.

Stefan Kitzler, Friedhelm Victor, Pietro Saggese, and Bernhard Haslhofer. Disentangling decentralized finance (defi) compositions. *ACM Transactions on the Web*, 17(2):1–26, 2023.

Anna Kovacs and Erik Costa. Decentralized governance models for blockchain-based organizations. *Blockchain, Web3 & Digital Trust Journal p-ISSN 3117-597X and e-ISSN 3117-5988*, 2(4):10–18, 2025.

Yixiao Li, Yifan Yu, Chen Liang, Pengcheng He, Nikos Karampatziakis, Weizhu Chen, and Tuo Zhao. Loftq: Lora-fine-tuning-aware quantization for large language models. *arXiv preprint arXiv:2310.08659*, 2023.

Dan Lin, Jiajing Wu, Qi Yuan, and Zibin Zheng. Modeling and understanding ethereum transaction records via a complex network approach. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 67(11):2737–2741, 2020.

Wei Liu, Weihao Zeng, Keqing He, Yong Jiang, and Junxian He. What makes good data for alignment? a comprehensive study of automatic data selection in instruction tuning. *arXiv preprint arXiv:2312.15685*, 2023.

Michael Pacheco, Gustavo Oliva, Gopi Krishnan Rajbahadur, and Ahmed Hassan. Is my transaction done yet? an empirical study of transaction processing times in the ethereum blockchain platform. *ACM Transactions on Software Engineering and Methodology*, 32(3):1–46, 2023.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37: 126544–126565, 2024.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

Alexandru-Gabriel Tudorache. A detailed set of ideas for designing a quantum computing framework based on smart contracts, configured using foundry and qiskit. *EMITTER International Journal of Engineering Technology*, 13(1):139–155, 2025.

Shuai Wang, Wenwen Ding, Juanjuan Li, Yong Yuan, Liwei Ouyang, and Fei-Yue Wang. Decentralized autonomous organizations: Concept, model, and applications. *IEEE Transactions on Computational Social Systems*, 6(5):870–878, 2019.

Jiahua Xu, Krzysztof Paruch, Simon Cousaert, and Yebo Feng. Sok: Decentralized exchanges (dex) with automated market maker (amm) protocols. *ACM Computing Surveys*, 55(11):1–50, 2023a.

Qiantong Xu, Fenglu Hong, Bo Li, Changran Hu, Zhengyu Chen, and Jian Zhang. On the tool manipulation capability of open-source large language models. *arXiv preprint arXiv:2305.16504*, 2023b.

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

Pei Yang, Wanyi Chen, Ke Wang, Lynn Ai, Eric Yang, and Tianyu Shi. Evm-questbench: An execution-grounded benchmark for natural-language transaction code generation. *arXiv preprint arXiv:2601.06565*, 2026.

Duanyi Yao, Siddhartha Jagannath, Baltasar Aroso, Vyas Krishnan, and Ding Zhao. Auditable llm arbiter for defi security: A hybrid graph-of-thoughts approach to intent–transaction alignment.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 3911–3921, 2018.

Zhaojian Yu, Yilun Zhao, Arman Cohan, and Xiao-Ping Zhang. Humaneval pro and mbpp pro: Evaluating large language models on self-invoking code generation. *arXiv preprint arXiv:2412.21199*, 2024.

Jiarui Zhang, Yukun Cheng, Xiaotie Deng, Bo Wang, Jan Xie, Yuanyuan Yang, and Mengqian Zhang. Preventing spread of spam transactions in blockchain by reputation. In *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*, pp. 1–6. IEEE, 2020.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, and Zheyan Luo. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 3: system demonstrations)*, pp. 400–410, 2024.

A PROMPTS

A.1 GUIDING MODEL TO DERIVE INTENT FROM TRANSACTIONS

```
You are a Web3 User Intent Generator. Your goal is to reverse-engineer a
    raw blockchain transaction into a realistic user request.

### Logic: Imagine the User's Persona
1. **For Common Actions (Transfer/Swap/Wrap)**:
   - Act as a casual user. Use colloquial language.
   - Example: "Send all my ETH to my ledger", "Swap 100 USDT for some
       Pepe", "Wrap 2 Ether".
   - *Key*: Focus on the "what" and "how much", keep it natural.

2. **For Complex/DeFi Actions (Staking/Liquidity/Vaults)**:
   - Act as a DeFi user. Be more precise about the protocol and the
       action.
   - You can even provide specific details(contract address, function
       signature) when the transaction is complex or contract is obscure.
        However, you should be careful about the extent to which you
       provide this information.
   - Example: "Deposit 500 USDC into the Aave V3 pool", "Claim my rewards
        from the Lido dashboard".

3. **For Low-level/Niche/Rare Contract Calls**:
   - Act as a power user or developer. If the contract is rare, include
       specific IDs or addresses provided in the parameters.
   - If the transaction contains a long hex string (like data or secret)
       that cannot be simplified, explicitly mention that 'pre-calculated
        data' or 'specific secret' should be used.
   - Example: "Vote 'Yes' on Proposal #42", "Mint 3 NFTs from the
       whitelist contract at 0x7f18bb4dd92cf2404c54cba1a9be4a1153bdb078",
        "Call the emergency withdraw on the vault".

### Core Rules:
- **No Technical Metadata**: Do NOT mention 'bytes', 'calldata' or '
    uint256'.
- **Param Integration**: If a parameter looks like a 'recipient', 'amount
    ', or 'token symbol', incorporate it into the sentence.
- **Vary the Tone**: Choose professional, casual, and urgent tones
    according to the transaction content.
- **Output Only**: Provide the sentence only, no preamble.

### User Persona Examples:
- [Input: Transfer ETH] -> "Give 0.5 ETH to my friend at 0
    x942a6a136f84d491134b9e03a35dafa701d7ea21"
- [Input: Uniswap Swap] -> "I want to exchange 1000 USDC for as much LINK
     as possible."
- [Input: Niche Mint] -> "Mint 5 tokens from the contract 0
    x7f18bb4dd92cf2404c54cba1a9be4a1153bdb078 on the mainnet."
```

A.2 SYSTEM PROMPT FOR INTENT TO TRANSACTION TASK

```
You are a DeFi execution planner.
Your task is to convert a user's high-level intent into a SINGLE on-chain
     action represented as a JSON object.
Output requirements:
- Output JSON only, no explanations.
- Follow the exact schema defined below.
- For each parameter, you must provide both its Solidity type and the
    specific value inferred from the intent.
JSON schema:\n"
{\n"
  "contract": string,              // contract name (e.g., "
      UniswapV2Router02")
  "contract_address": string,      // contract address (e.g., "0
      x7a250d5630b4cf539739df2c5dacabf6959a1e")
  "function": string,              // function name (e.g., "transfer")
  "params": {                      // map of parameters
    "<param_name>": {\n'
      "type": string,              // Solidity type (e.g., "address", "
          uint256")
      "val": any                   // specific value inferred from the
          intent
    }\n'
    ... // more parameters if needed
  },
  "value": float                   // Ether amount sent with the
      transaction (in ETH)
}
```

A.3 Address book provided to LLMs

We provide an address book to all models at inference time to ensure a fair and controlled evaluation. The address book contains mappings between protocol names and on-chain addresses, which are necessary for constructing executable transactions but are not the primary focus of our benchmark.

Without this information, models—especially smaller ones—would be penalized for memorization or hallucination errors unrelated to the core task of intent understanding and transaction composition. By standardizing access to such external knowledge, we isolate the models' reasoning and planning capabilities.

This design follows a common practice in tool-use and agent evaluation, where auxiliary resources are provided to reduce irrelevant variance and improve comparability across models of different scales.

```
You may refer to the following address book for commonly used Ethereum
    protocol addresses. This can help you avoid mistakes when inputting
    contract addresses for popular tokens and protocols.

## Additional Information: Helpful Ethereum Address Book

This address book provides a mapping between common protocol names (just
    a description, not contract name), token symbols, and their on-chain
    addresses.

### Tokens
| Name / Symbol | Contract Address |
| :--- | :--- |
| **USDT** | `0xdAC17F958D2ee523a2206206994597C13D831ec7` |
| **USDC** | `0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eb48` |
| **DAI** | `0x6B175474E89094C44Da98b954EedeAC495271d0F` |
| **WBTC** | `0x2260FAC5E5542a773Aa44fBCfeDf7C193bc2C599` |
| **WETH** | `0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2` |
| **PYUSD** | `0x6c3ea9036406852006290770BEdFcAbA0e23A0e8` |
```

```
| **LINK** | `0x514910771af9ca656af840dff83e8264ecf986ca` |
| **UNI** | `0x1f9840a85d5af5bf1d1762f925bdaddc4201f984` |
| **fwWETH** | `0xa250cc729bb3323e7933022a67b52200fe354767` |
| **SPX** | `0xe0f63a424a4439cbe457d80e4f4b51ad25b2c56c` |
| **PAXG** | `0x45804880de22913dafe09f4980848ece6ecbaf78` |
| **AAVE** | `0x7fc66500c84a76ad7e9c93437bfc5ac33e2ddae9` |
| **ENA** | `0x57e114b691db790c35207b2e685d4a43181e6061` |
| **XAUt** | `0x68749665ff8d2d112fa859aa293f07a622782f38` |
| **CRV** | `0xd533a949740bb3306d119cc777fa900ba034cd52` |
| **USDS** | `0xdc035d45d973e3ec169d2276ddab16f1e407384f` |

### DEX / Routers
| Name / Symbol | Contract Address |
| :--- | :--- |
| **Uniswap_V2_Router** | `0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D` |
| **Uniswap_V3_Router_2** | `0x68b3465833fb72A70ecDF485E0e4C7bD8665Fc45`
    |
| **Universal_Router** | `0x3fc91a3afd70395cd496c647d5a6cc9d4b2b7fad` |
| **SushiSwap_Router** | `0xd9e1cE17f2641f24aE83637ab66a2cca9C378B9F` |
| **1inch_Aggregation_Router_V5** | `0
    x1111111254EEB25477B68fb85Ed929f73A960582` |
| **1inch_Aggregation_Router_V6** | `0
    x111111125421cA6dc452d289314280a0f8842A65` |
| **0x_Protocol_V4** | `0xDef1C0ded9bec7F1a1670819833240f027b25EfF` |
| **Curve_Router** | `0x99a58482BD75cbab83b27EC03CA68fF489b5788f` |
| **Balancer_V2_Vault** | `0xBA12222222228d8Ba445958a75a0704d566BF2C8` |
| **Kyberswap_V2** | `0x6131B5fae19EA4f9D964eAc0408E4408b66337b5` |

### Lending / Staking
| Name / Symbol | Contract Address |
| :--- | :--- |
| **Aave_V2_Lending_Pool** | `0x7d2768dE32b0b80b7a3454c06BdAc94A69DDc7A9`
    |
| **Aave_V3_Lending_Pool** | `0x87870Bca3F3fD6335C3F4ce8392D69350B4fA4E2`
    |
| **Lido_stETH** | `0xae7ab96520DE3A18E5e111B5EaAb095312D7fE84` |
| **Rocket_Pool_RETH** | `0xae78736Cd615f374D3085123A210448E74Fc6393` |
| **Spark_V1_Lending_Pool** | `0xC13e21B648A5Ee794902342038FF3aDAB66BE987
    ` |
| **Morpho_Blue_V2** | `0xBBBBBbbBBb9cC5e90e3b3Af64bdAF62C37EEFFCb` |
| **Compound_V2** | `0x3d9819210A31b4961b30EF54bE2aeD79B9c9Cd3B` |
| **Compound_V3_USDC_Market** | `0
    xc3d688B66703497DAA19211EEdff47f25384cdc3` |
| **Compound_V3_WETH_Market** | `0
    xA17581A9E3356d9A858b789D68B4d866e593aE94` |

### NFT
| Name / Symbol | Contract Address |
| :--- | :--- |
| **Opensea_Seaport** | `0x00000000000000ADc04C56Bf30aC9d3c0aAF14dC` |
| **Blur_Blend** | `0x29469395eAf6f95920E59F858042f0e28D98a20B` |
| **CryptoPunks** | `0xb47e3cd837dDF8e4c57F05d70Ab865de6e193BBB` |

### Bridges / Infrastructure
| Name / Symbol | Contract Address |
| :--- | :--- |
| **Across_Spoke_Pool** | `0x5c7BC2d53d245f5021E17E266bb792010103510c` |
| **Optimism_L1_Standard_Bridge** | `0
    x99C9fc46f92E8a1c0deC1b1747d010903E884bE1` |
| **Arbitrum_L1_Standard_Bridge** | `0
    x8315177aB297bA92A06054cE80a67Ed4DBd7ed3a` |
| **Base_L1_Standard_Bridge** | `0
    x3154Cf16ccdb4C6d922629664174b904d80F2C35` |
| **Polygon_RootChainManager** | `0
    xA0c68C638235ee32657e8f720a23ceC1bFc77C77` |
```

```
| **Stargate_Route** | `0x8731d54E9D02c286767d56ac03e8037C07e01e98` |
| **Hop_Ethereum_Bridge** | `0xb8901acB165ed027E32754E0FFe830802919727f`
    |
| **Synapse_Bridge** | `0x2796317b0fF8538F253012862c06787Adfb8cEb6` |
| **Celer_cBridge_V2** | `0x5427FEFA711Eff984124bFBB1AB6fbf5E3DA1820` |
| **zkSync_Era_L1_DiamondProxy** | `0
    x32400084C286CF3E17e7B677ea9583e60a000324` |
```

B HUMAN VALIDATION OF GENERATED INTENTS

Transaction targets in INTENT2TX are deterministically decoded from successful mainnet calls; apart from the human-written governance intents, the natural language intents are generated by `gemini-2.5-flash`. To validate these LLM-generated intents, we audited 900 stratified examples during dataset construction: 600 single-step examples covering all 11 categories and 300 multi-step examples stratified by sequence length.

Two authors experienced with Ethereum transactions, Solidity ABIs, and DeFi protocols independently compared each intent with its decoded transaction along five dimensions: (1) semantic faithfulness to the transaction, (2) coverage of user-relevant constraints (e.g., amounts, recipients, tokens), (3) absence of unsupported information, (4) goal-level wording that describes the "what" rather than the "how", and (5) linguistic clarity. Disagreements were discussed with a third author until all three reached consensus. Raw agreement between the two annotators is 98.3%, with Cohen's $\kappa = 0.85$.

Table 3 reports the rejection rates. Multi-step examples required more correction than single-step examples. All rejected examples were revised according to the consensus decision and checked again before inclusion in the final dataset.

Table 3: Results of the human validation audit on LLM-generated intents.

Audit subset	Audited	Rejected	Rejection rate
Single-step	600	28	4.7%
Multi-step	300	27	9.0%
Overall	900	55	6.1%

C ANALYSIS OF INTENT-GENERATOR BIAS

Because `gemini-2.5-flash` both generates the non-governance intents and is one of the evaluated models, its results may benefit from familiarity with its own wording style. We note that the LLM-generated pairs were newly constructed and not public before evaluation, so no evaluated model could have seen them during pretraining; the concern is therefore stylistic affinity rather than memorization. Table 4 compares Gemini 2.5 Flash with the strongest models on single-step tasks, using the results in Tables 1 and 2.

Table 4: Single-step results of the intent generator (Gemini 2.5 Flash) and the strongest evaluated models. Values are taken from Tables 1 and 2; the rank of Gemini 2.5 Flash among all 16 models is given in parentheses. Retrieval denotes the oracle retrieval setting.

	Direct		Retrieval	
Model	Final	StateEq	Final	StateEq
Gemini 2.5 Flash	0.32 (8th)	0.616 (1st)	0.70 (1st, tied)	0.692 (6th)
Gemini 2.5 Pro	0.38	0.605	0.69	0.702
Claude Opus 4.5	0.37	0.472	0.70	0.706
GPT 5.2	0.38	0.541	0.67	0.697

The results do not show a consistent advantage for the generator. Under direct inference, Gemini 2.5 Flash ranks only 8th in Final score, below Gemini 2.5 Pro, Claude Opus 4.5, and GPT 5.2. It achieves the highest direct StateEq, but only by a margin of 0.011 over Gemini 2.5 Pro. Under retrieval, it ties with Claude Opus 4.5 in Final score and ranks 6th in StateEq. Its advantage thus depends on the setting and metric, and does not appear across all settings. We nevertheless cannot fully exclude generator-style affinity, and we regard the use of a single LLM generator as a limitation of the current benchmark.

D INTENT DISCLOSURE AND DUPLICATION STATISTICS

Explicit target disclosure. To quantify how much of the target transaction is stated explicitly in the intents, we measured the fraction of LLM-generated intents that contain each target field. We used strict case-insensitive raw substring matching: the complete contract name, the exact contract address, the complete function name, or an unmodified raw parameter value must appear contiguously in the intent, without whitespace normalization, token splitting, semantic matching, or unit conversion. Table 5 reports the results.

Table 5: Percentage of LLM-generated intents that explicitly contain each target field (strict case-insensitive raw substring matching).

Subset	Contract/ protocol name	Contract address	Function name	Any raw parameter
All	45.7%	3.8%	34.7%	74.0%
Single-step	44.8%	3.2%	33.8%	73.2%
Multi-step	62.4%	14.7%	53.0%	89.5%

Explicit mentions of contract addresses are rare (3.8%), and complete function names appear in about one third of the intents. The higher rates for names and parameters mostly reflect how users naturally phrase requests rather than leakage of the execution route: (1) Web3 protocols are commonly referred to by the same proper name as their deployed contracts, so mentioning a service (e.g., "Aave" or "Uniswap") counts as a contract-name match without revealing the address or the specific router; (2) many Solidity function names, such as `transfer`, `swap`, `deposit`, `withdraw`, and `approve`, are ordinary action verbs that naturally appear in goal-level requests; (3) parameter values such as amounts, recipients, token addresses, proposal IDs, and deadlines are user-specified constraints that are necessary to reconstruct the transaction, and cannot be inferred otherwise. Multi-step intents show higher rates because they enumerate several actions and therefore mention more protocols, functions, and values.

Duplication. Exact duplicate intent strings account for 2.86% of the LLM-generated single-step instances. Such duplicates can arise naturally when different users perform the same common operation.

E DATA EXAMPLES

To provide a concrete understanding of the INTENT2TX benchmark, we present representative examples illustrating the mapping from high-level natural language intents to structured, executable on-chain calls. Each instance in our dataset contains metadata (omitted here), a user intent and a corresponding JSON-formatted ground truth, which includes the contract name, address, function signature, and typed parameters.

```
------------------------------ Example 1
    ----------------------------------

Input: User intent: Transfer 4.399312393802940721 LPT to 0
    x0D0707963952f2FBA59dD06f2B425ace40B492FE.

Output:
{
  "contract": "LivepeerToken",
```

```
  "contract_address": "0x58b6a8a3302369daec383334672404ee733ab239",
  "function": "transfer",
  "params": {
    "_to": {
      "type": "address",
      "val": "0x0d0707963952f2fba59dd06f2b425ace40b492fe"
    },
    "_value": {
      "type": "uint256",
      "val": "4399312393802940721"
    }
  },
  "value": 0
}

-------------------------------- Example 2
    ----------------------------------

Input: User intent: Burn my NFT position with ID 950002.

Output:
{
  "contract": "NonfungiblePositionManager",
  "contract_address": "0xc36442b4a4522e871399cd717abdd847ab11fe88",
  "function": "burn",
  "params": {
    "tokenId": {
      "type": "uint256",
      "val": "950002"
    }
  },
  "value": 0
}

-------------------------------- Example 3
    ----------------------------------

Input: User intent: Mint an NFT to my wallet 0
    xa17a980c0f540489e75664b32cb0f9cb17a4940f from the contract at 0
    xa9ab98cb65a9c522a196823ffd04dd3f7276c19f, using metadata from ipfs
    ://QmSkQEgqsPSRN5b6nXf7TW6NyJFpEgDJXV9NF6QjvwNAun/0.after that Create
     a new ERC1155 token with the URI "ipfs://
    QmSfpSWBYbTiRpdXeG1mdkTudg8UYzc1oC6Vr85CLCGdP3/0" and mint 1 of these
     tokens to 0xa17a980c0f540489e75664b32cb0f9cb17a4940f.

Output:
[
  {
    "contract": "ERC721TL",
    "contract_address": "0xa9ab98cb65a9c522a196823ffd04dd3f7276c19f",
    "function": "mint",
    "params": {
      "recipient": {
        "type": "address",
        "val": "0xa17a980c0f540489e75664b32cb0f9cb17a4940f"
      },
      "uri": {
        "type": "string",
        "val": "ipfs://QmSkQEgqsPSRN5b6nXf7TW6NyJFpEgDJXV9NF6QjvwNAun/0"
      }
    },
    "value": 0
  },
  {
    "contract": "ERC1155TL",
```

```
    "contract_address": "0x1b10f094efb8bc1defb12af23313d2e45054a25d",
    "function": "createToken",
    "params": {
      "newUri": {
        "type": "string",
        "val": "ipfs://QmSfpSWBYbTiRpdXeG1mdkTudg8UYzc1oC6Vr85CLCGdP3/0"
      },
      "addresses": {
        "type": "address[]",
        "val": [
          "0xa17a980c0f540489e75664b32cb0f9cb17a4940f"
        ]
      },
      "amounts": {
        "type": "uint256[]",
        "val": [
          "1"
        ]
      }
    },
    "value": 0
  }
]
```


The examples above showcase several key challenges addressed by INTENT2TX:

Precision and Unit Normalization: In `Example 1`, the model must accurately convert a floating-point token amount (4.399... LPT) into its base unit (*wei*) equivalent as a `uint256` string, requiring knowledge of the token's decimal precision (18 decimals for LPT).

Protocol-Specific Mapping: `Example 2` demonstrates the transition from domain-specific jargon ("burn my NFT position") to the specific logic of the *NonfungiblePositionManager* contract, identifying the correct function (`burn`) and the associated `tokenId`.

Multi-step Logic and Composition: `Example 3` illustrates a complex, multi-goal intent. The model must decompose the request into a sequence of atomic operations—first minting an ERC721 token and then creating a new ERC1155 token. This requires not only correct parameter extraction (e.g., handling `ipfs` URIs) but also strategic planning to maintain the execution order.

Complex Type Handling: As seen in the output for the ERC1155 creation, the benchmark requires models to format parameters into complex Solidity types, such as `address[]` and `uint256[]` arrays, ensuring they are ready for ABI encoding.

F CASES OF LLMS GENERATING TRANSACTIONS

To provide a deeper understanding of the model's reasoning capabilities and failure modes, we present several qualitative examples across different model families and task complexities. These cases are extracted from our experimental evaluations, showcasing both successful logical mappings and representative errors in numerical precision, protocol selection, and address handling.

```
---------------------- Claude Haiku 4.5 : Single-step
    ------------------------

Input: User intent: Swap 0.0534 ETH for at least 153.71M units of token
    from address 0x5148e5CfDfDF76987cD1c6033bbCD52DF0A8d75F, sending them
     to 0xb86c24408f7ce28fc4742493aeb9776f163a19bc.

Output:
{
  "contract": "UniswapV2Router02",
  "contract_address": "0x7a250d5630b4cf539739df2c5dacb4c659f2488d",
  "function": "swapExactETHForTokens",
  "params": {
```

```
    "amountOutMin": {
      "type": "uint256",
      "val": "153711954830041625150926594 2"
    },
    "path": {
      "type": "address[]",
      "val": [
        "0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2",
        "0x5148e5cfdfdf76987cd1c6033bbcd52df0a8d75f"
      ]
    },
    "to": {
      "type": "address",
      "val": "0xb86c24408f7ce28fc4742493aeb9776f163a19bc"
    },
    "deadline": {
      "type": "uint256",
      "val": "1764335997"
    }
  },
  "value": 0.0534
}

LLM_Output:
{
  "contract": "UniswapV2Router02",
  "contract_address": "0x7a250d5630b4cf539739df2c5dacb4c659f2488d",
  "function": "swapExactETHForTokens",
  "params": {
    "amountOutMin": {
      "type": "uint256",
      "val": "153710000000000000000000000"
    },
    "path": {
      "type": "address[]",
      "val": [
        "0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2",
        "0x5148e5CfDfDF76987cD1c6033bbCD52DF0A8d75F"
      ]
    },
    "to": {
      "type": "address",
      "val": "0xb86c24408f7ce28fc4742493aeb9776f163a19bc"
    },
    "deadline": {
      "type": "uint256",
      "val": "1742220567"
    }
  },
  "value": 0.0534
}

------------------------ Llama 3.1 8B : Single-step
    -------------------------

Input: User intent: Swap 0.015874411512787778 units of the token at 0
    x13919f59286578600321f83941369e319dbc0bc5 for at least
    1.654878062516271904 ETH, and send it to 0
    x1af8008c5f5bca43305a3d4104d5e1802c49814f. This needs to be done
    before May 5, 2025 at 1:26:51 PM GMT.

Output:
{
  "contract": "UniswapV2Router02",
  "contract_address": "0x7a250d5630b4cf539739df2c5dacb4c659f2488d",
```

```
  "function": "swapExactTokensForETH",
  "params": {
    "amountIn": {
      "type": "uint256",
      "val": "15874411512787778"
    },
    "amountOutMin": {
      "type": "uint256",
      "val": "1654878062516271904"
    },
    "path": {
      "type": "address[]",
      "val": [
        "0x13919f59286578600321f83941369e319dbc0bc5",
        "0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2"
      ]
    },
    "to": {
      "type": "address",
      "val": "0x1af8008c5f5bca43305a3d4104d5e1802c49814f"
    },
    "deadline": {
      "type": "uint256",
      "val": "1746274011"
    }
  },
  "value": 0
}

LLM_Output:
{
  "contract": "UniswapV2Router02",
  "contract_address": "0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D",
  "function": "swapExactETHForTokens",
  "params": {
    "amountIn": {
      "type": "uint256",
      "val": "0.015874411512787778"
    },
    "minAmountOut": {
      "type": "uint256",
      "val": "1.654878062516271904"
    },
    "path": {
      "type": "address[]",
      "val": [
        "0x13919f59286578600321f83941369e319dbc0bc5"
      ]
    },
    "to": {
      "type": "address",
      "val": "0x1af8008c5f5bca43305a3d4104d5e1802c49814f"
    },
    "deadline": {
      "type": "uint256",
      "val": "1649146801"
    }
  },
  "value": "0.0"
}

---------------------- Claude Opus 4.5 : Multi-step
    ------------------------
```

```
Input: User intent: Send 167.659520753773760851 Bounce to 0
    xf5a7b9617422fd252cf1cd9fbc69f42645ad0ead. and Send 15,107,747.49
    SPELL to 0xf5a7b9617422fd252cf1cd9fbc69f42645ad0ead.

Output:
[{"contract": "BounceAuctionToken", "contract_address": "0
    xa9b1eb5908cfc3cdf91f9b8b3a74108598009096", "function": "transfer", "
    params": {"recipient": {"type": "address", "val": "0
    xf5a7b9617422fd252cf1cd9fbc69f42645ad0ead"}, "amount": {"type": "
    uint256", "val": "167659520753773760851"}}, "value": 0.0}, {"contract
    ": "Spell", "contract_address": "0
    x090185f2135308bad17527004364ebcc2d37e5f6", "function": "transfer", "
    params": {"to": {"type": "address", "val": "0
    xf5a7b9617422fd252cf1cd9fbc69f42645ad0ead"}, "amount": {"type": "
    uint256", "val": "151077474931362249382787 90"}}, "value": 0.0}]

LLM_Output:
[{"contract": "BounceAuctionToken", "contract_address": "0
    xa9b1eb5908cfc3cdf91f9b8b3a74108598009096", "function": "transfer", "
    params": {"recipient": {"type": "address", "val": "0
    xf5a7b9617422fd252cf1cd9fbc69f42645ad0ead"}, "amount": {"type": "
    uint256", "val": "167659520753773760851"}}, "value": 0.0}, {"contract
    ": "SpellToken", "contract_address": "0
    x090185f2135308bad17527004364ebcc2d37e5f6", "function": "transfer", "
    params": {"to": {"type": "address", "val": "0
    xf5a7b9617422fd252cf1cd9fbc69f42645ad0ead"}, "amount": {"type": "
    uint256", "val": "151077470000000000000000"}}, "value": 0.0}]

---------------------- Ministral 14B : Multi-step
    --------------------------

Input: User intent: I want to resolve the dispute game.and Resolve up to
    512 claims, starting from claim index 0, on the Permissioned Dispute
    Game contract.

Output:
[{"contract": "PermissionedDisputeGame", "contract_address": "0
    xb86bf0d32b4676bca2053cca5fc6a512bded4b9a", "function": "resolve", "
    params": {}, "value": 0.0}, {"contract": "PermissionedDisputeGame", "
    contract_address": "0xee17cdba5a1490c4f5fe76d906b838832bc0b099", "
    function": "resolveClaim", "params": {"_claimIndex": {"type": "
    uint256", "val": "0"}, "_numToResolve": {"type": "uint256", "val":
    "512"}}, "value": 0.0}]

LLM_Output:
[ { "contract": "PermissionedDisputeGame", "contract_address": "0
    x0000000000000000000000000000000000000000", // Placeholder; replace
    with actual contract address if known "function": "resolve", "params
    ": {}, "value": 0.0 }, { "contract": "PermissionedDisputeGame", "
    contract_address": "0x0000000000000000000000000000000000000000", //
    Placeholder; replace with actual contract address if known "function
    ": "resolveClaim", "params": { "_claimIndex": { "type": "uint256", "
    val": "0" }, "_numToResolve": { "type": "uint256", "val": "512" } },
    "value": 0.0 } ]
```

By comparing the ground truth with the outputs from various models, we identify three primary failure modes:

Function Selection and Parameter Formatting: The *Llama 3.1 8B* case illustrates a logic reversal error. The model mistakenly selected `swapExactETHForTokens` for an intent that required selling a specific token for ETH (`swapExactTokensForETH`). Furthermore, it failed to perform the floating-to-integer conversion, passing a decimal string into a `uint256` field, which is syntactically invalid for ABI encoding.

Address Hallucination and Placeholder Usage: *Ministral 14B* exhibits a "placeholder hallucination" pattern in multi-step tasks. While it correctly understands the sequence of operations (resolving a dispute and its claims), it fails to retrieve or infer the specific contract addresses from its context, defaulting to the null address (`0x00...00`). This highlights the model's dependency on high-quality Retrieval-Augmented Generation (RAG) for long-tail protocol interaction.

Multi-step Decomposition Hurdles: In the *Claude Opus* multi-step example, the model successfully decomposes the intent into two separate `transfer` calls. However, the accumulation of minor numerical discrepancies across steps remains a challenge for even state-of-the-art models, suggesting that current LLMs lack the internal deterministic execution environment needed for high-stakes financial operations.

G LLMs PARTICIPATING IN THE EVALUATION

Table 6: Models evaluated in our benchmark.

Model	**OpenRouter Model Name**	**Type**	**Key Strengths / Notes**
GPT 5.2	openai/gpt-5.2	Closed	Strong reasoning, long context ($\sim$400k), fast inference
GPT 5.2 Codex	openai/gpt-5.2-codex	Closed	Optimized for coding and tool-use workflows
Claude Haiku 4.5	anthropic/claude-haiku-4.5	Closed	Fast, cost-efficient lightweight model
Claude Opus 4.5	anthropic/claude-opus-4.5	Closed	Top-tier coding and agent performance
Gemini 2.5 Flash	google/gemini-2.5-flash	Closed	Low-latency, cost-efficient multimodal model
Gemini 2.5 Pro	google/gemini-2.5-pro	Closed	Long context ($\sim$1M), strong multimodal reasoning
DeepSeek 3.2	deepseek/deepseek-v3.2	Open-weight	Competitive reasoning, low cost, MoE architecture
Kimi 2	moonshotai/kimi-k2-0905	Closed	Long-context dialogue, strong bilingual ability
Llama 3.1 8B	meta-llama/llama-3.1-8b-instruct	Open	Small, efficient, widely deployable
Llama 3.3 70B	meta-llama/llama-3.3-70b-instruct	Open	Strong general-purpose open model
Qwen3 235B	qwen/qwen3-235b-a22b-2507	Open-weight	Large MoE model, strong reasoning performance
Qwen3 Coder	qwen/qwen3-coder-next	Open-weight	Specialized for code generation and reasoning
Ministral 14B	mistralai/ministral-14b-2512	Open	Efficient small model with good latency-performance
Codestral	mistralai/codestral-2508	Open-weight	Code-focused model for completion and generation
GLM 4.6	z-ai/glm-4.6	Closed	General-purpose Chinese/English model
GLM 4 32B	z-ai/glm-4-32b	Open-weight	Mid-scale deployable model