HARNESS ENGINEERING IN LLM TOOL USE VIA AGENT-NATIVE REUSABLE TOOL PRIMITIVES

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) augmented with external tools have demonstrated remarkable capability in solving complex real-world tasks. Existing approaches remain brittle in multi-step and multi-turn settings, where the model must map one tool's output into the next tool's input schema, and degrade further as the tool catalogue grows. We introduce **Tool Primitives**, which replace schema-based invocation with a natural language interface: each tool is wrapped with an LLM that resolves its own schema and executes the underlying function, so callers describe what they want instead of emitting schema-compliant arguments, and dependent calls pass results to one another in natural language. Building on them, we host **ToolFace**, a repository of 25,519 functions from which only the tools retrieved for the current query enter context, bounding what the model reasons over by the retrieval budget rather than by the size of the repository. A primitive, however, answers only the request it is handed. We therefore propose **HEART**, a **H**arness **E**ngineering framework via **A**gent-native, **R**eusable **T**ool Primitives, which decomposes a query into an invocation plan, grounds each step's arguments from the query and from earlier results, and validates every returned result, feeding failures back as targeted revisions.

Experiments on five benchmarks demonstrate that HEART outperforms SFT-based models by 10% on average and surpasses GPT-5.4, Claude-4.6-Sonnet, and Gemini-3.1-Pro by 6% on average while reducing API cost by up to 85%. On 50 real-world tasks, HEART achieves 84% task completion, $3.8\times$ the average of three frontier commercial models (22%).

1 INTRODUCTION

Large Language Models (LLMs) with external tools and APIs has significantly enhanced the capability to solve complex real-world tasks (Huang et al., 2024; Qin et al., 2023). To equip LLMs with tool-use capabilities, existing approaches broadly fall into two paradigms: training-free methods that elicit tool invocation through prompting and in-context demonstrations (Yao et al., 2022; Shen et al., 2023), and fine-tuning-based methods that train models on curated tool-use trajectories via supervised fine-tuning (SFT) and preference optimization (Lin et al., 2024; Liu et al., 2024c; Prabhakar et al., 2025). The latter have driven substantial improvements in structured function calling, with models increasingly evaluated on standardized benchmarks such as BFCLv4 (Patil et al., 2025).

Despite rapid progress, existing LLM tool-use methods remain brittle in complex multi-step and multi-turn reasoning. When tool invocations involve dependent steps, models often fail to correctly map outputs from one tool into the expected input schema of the next, since tools expose heterogeneous APIs and return diverse output formats. This issue becomes even more severe in multi-turn interactions, where relevant context progressively degrades as histories grow longer. Existing benchmarks confirm these limitations: on nested API call benchmarks such as NESTFUL (Basu et al., 2025), even the strongest models achieve only 28% full-sequence match accuracy, while on multi-turn benchmarks such as τ^2-Bench (Barres et al., 2025), performance drops substantially as interaction depth increases. This raises a natural question: **instead of requiring models to speak the language of APIs, can tools be made to speak the language of models?** We answer affirmatively with **Tool Primitives**, which replace rigid schema-based invocation with natural language interfaces for tool calling. Rather than requiring models to memorize exact parameter types and API schemas, each

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.

tool is wrapped with an LLM interface that handles schema resolution and execution internally. Models therefore interact with tools through natural language, making inter-tool communication more uniform and enabling nested tool calling and multi-turn information acquisition without explicit schema knowledge.

Second, a large tool catalogue degrades performance. Tool invocation is typically conditioned on a large catalogue of candidate tools provided in the input context; as the number of available tools grows, model performance degrades substantially. Recent evaluations show accuracy drops of 7%-85% as tool catalogue size scales from 8K to 120K tokens (Kate et al., 2025; Liu et al., 2024b), and this problem is further amplified in real-world settings such as ToolBench (Qin et al., 2023), which encompasses over 16,000 real-world APIs. Motivated by this, we argue that **models need not know the full tool catalogue upfront and tools should instead be retrieved on demand at inference time**. Inspired by Agent Skills (Anthropic, 2025a) and Agent Primitives (Jin et al., 2026), both of which package instructions, metadata, and resources into modular reusable capability blocks, we treat tools analogously and introduce **ToolFace**, a large-scale tool repository containing 25,519 functions each paired with a structured schema and an executable implementation, where each function is further wrapped as a Tool Primitive to enable natural language interaction.

Tool Primitives alone, however, do not determine which tools a query needs, and a primitive that sees only its own call knows neither where its argument values come from nor whether its result answers the query. We therefore introduce **HEART**, a **H**arness **E**ngineering framework via **A**gent-native, **R**eusable **T**ool Primitives, which uses three mechanisms. Request decomposition turns the query into an ordered invocation plan, or asks the user for a missing detail when the context cannot support one. Value resolution grounds each step's arguments from the query and from earlier results into a natural-language request, which the target primitive validates against its own schema. Result validation checks each result against task completion, argument consistency, execution validity, and constraint satisfaction, and on failure returns feedback that enriches the context and triggers a targeted revision. This loop repeats until validation passes or a revision budget is exhausted.

Extensive experiments on five benchmarks spanning large-scale real-world API invocation across single- and multi-tool scenarios (ToolBench (Qin et al., 2023)), nested API call sequences (NESTFUL (Basu et al., 2025)), dual-control conversational agent evaluation (τ^2-Bench Barres et al. (2025)), robustness and fine-grained tool-use evaluation (ACEBench (Chen et al., 2025)) and agentic web search and memory function calling (BFCLv4 (Patil et al., 2025)), demonstrate that HEART consistently outperforms SFT-based models by 10% on average and surpasses three commercial models, including GPT-5.4, Claude-4.6-Sonnet, and Gemini-3.1-Pro, by 6% on average while reducing token cost by up to 85%. Furthermore, we curate 50 real-world tasks on which HEART achieves 84% task completion and demonstrates robustness against prompt injection attacks, reducing attack success rate to 0.0%. Our main contributions are as follows:

- We design **Tool Primitives**, a natural language interface that encapsulates schema resolution and execution within each tool, enabling nested and multi-turn tool calling without requiring explicit API schema knowledge. We further introduce **ToolFace**, a repository of 25,519 functions wrapped as Tool Primitives, enabling dynamic tool retrieval at inference time.
- We introduce **HEART**, a harness engineering framework that makes these primitives practical at repository scale: it retrieves only the tools a query is likely to need, resolves their arguments through natural language rather than schema-compliant dictionaries, and confines recovery to the step whose result failed validation.
- Experiments on five benchmarks show HEART outperforms SFT-based models by 10% and three commercial models by 6% on average, while reducing token cost by up to 85%. On 50 real-world tasks, HEART achieves 84% task completion, $3.8\times$ the average of three frontier models (22%).

2 Related Work

Tool-Use LLMs. Work on tool invocation has advanced along tool coverage and execution complexity. Hammer (Lin et al., 2024) targets efficient on-device deployment in a single-turn regime, while MAGNET (Yin et al., 2025), ToolACE (Liu et al., 2024c), and the xLAM series (Liu et al., 2024d) synthesize multi-turn trajectories covering nested compositions, cross-turn dependencies, and missing-parameter scenarios. These efforts progressively move from single-turn invocation toward sustained multi-turn execution, but share a limitation: the model is still required to emit schema-compliant

arguments for every call, so composition remains bounded by how reliably it can map one tool's output into the next tool's input format.

Benchmarks for Tool Use Evaluation. Evaluation spans structured function calling (Patil et al., 2025; Srinivasan et al., 2023), large-scale real-world APIs (Qin et al., 2023; Guo et al., 2024), compositional and nested call sequences (Basu et al., 2024; 2025), and interactive multi-turn settings (Yao et al., 2024; Barres et al., 2025; Wang et al., 2025; Chen et al., 2025; Lu et al., 2025). These benchmarks cover diverse scenarios but primarily evaluate task completion under a fixed tool environment.

Key Differences. Prior methods expose tools to the model as raw schemas in the input context, whether through prompting or fine-tuning on static trajectories, which couples tool storage, representation, and invocation. HEART decouples them. Tool Primitives give each tool a natural language interface in place of a raw schema, ToolFace serves only the primitives a query retrieves rather than loading a catalogue into context, and a Tool Primitive is defined by the tool it wraps rather than by an assigned role, so the set of primitives is induced by ToolFace and grows with it while invocations chain along the data dependencies between calls rather than a fixed pipeline. HEART adds only what the abstraction leaves open, decomposing the query, grounding each step's arguments, and validating each result before it is accepted. Appendix A gives an extended discussion.

3 METHODOLOGY

3.1 OVERVIEW

HEART is built on ToolFace, a tool repository that decouples tool storage from the model's input context, and Tool Primitives, LLM-wrapped interfaces that accept natural language invocation without exposing raw API schemas. The harness around them decomposes a query into an invocation plan, grounds each step's arguments from the query and from earlier results, and validates each result before it is accepted. An overview of HEART is shown in Fig. 1.

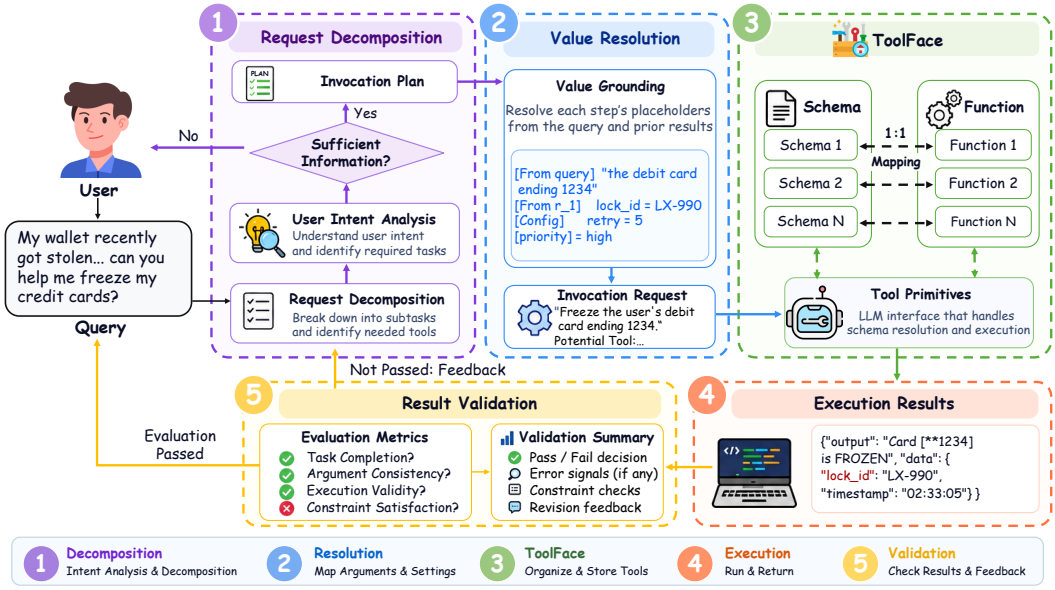

Figure 1: Overview of HEART. **(1) Request decomposition** analyses the user's intent and checks whether the context supports a plan, soliciting missing details if not. **(2) Value resolution** grounds each step's placeholders from the query and from prior results into a natural-language invocation request. **(3) ToolFace** stores tools as schema–function pairs, each wrapped as a Tool Primitive that resolves the schema internally. **(4) Execution** returns the structured result, which **(5) result validation** checks against task completion, argument consistency, execution validity, and constraint satisfaction; on failure it returns feedback that triggers a targeted revision.

3.2 TOOLFACE AND TOOL PRIMITIVES

Tool Primitives. A Tool Primitive is an LLM-based wrapper that encapsulates a single tool from ToolFace, serving as its agent-native interface within the HEART pipeline. Rather than exposing raw

API schemas directly to the calling model, each Tool Primitive accepts natural language invocation requests, internally resolves the corresponding schema from ToolFace, executes the underlying function, and returns a structured result.

Formally, let $\mathcal{T} = \{(s_i, f_i)\}_{i=1}^{N}$ denote the ToolFace repository of N schema–function pairs, where s_i specifies the interface of tool i and f_i is its executable implementation. All Tool Primitives have a single base LLM $\mathcal{M}$, conditioned on different schemas at inference time. A Tool Primitive $\mathcal{P}_i$ is:

$$\mathcal{P}_i(x; c) = \mathcal{M}\big([s_i;\, c;\, x]\big), \tag{1}$$

where x is the natural language request issued by the upstream caller, c is optional context (e.g., a prior primitive's result), $[\cdot]$ denotes prompt concatenation, and c is left empty when no context is present. Upon receiving x and c, $\mathcal{P}_i$ interprets the request, maps it to the argument space defined by s_i, invokes f_i with the resolved arguments, and returns the execution result r_i.

ToolFace. To decouple tools from the LLM's input and enable on-demand retrieval at inference time, we build ToolFace as a centralized tool registry. It stores tools as structured schema–function pairs: we manually author each schema to capture the tool's interface, including parameter types, constraints, and return specifications, and pair it with the corresponding executable implementation.

To build it, we collect 25,519 tools in total: 16,464 live APIs spanning 49 functional categories from ToolBench Qin et al. (2023); 40 mathematical reasoning tools and 4,348 coding tools from NESTFUL Basu et al. (2025); 68 tools across retail, airline, and telecom domains from τ^2-Bench Barres et al. (2025); 4,538 bilingual APIs spanning 8 major domains and 68 sub-categories (including technology, finance, entertainment, and healthcare) from ACEBench Chen et al. (2025); 2 web search and 5 memory tools from BFCLv4 Patil et al. (2025); and 54 manually crafted tools to support human-like interactions such as form filling, button clicking, and webpage analysis.

3.3 Harness Engineering via Agent-Native Reusable Tool Primitives

Tool Primitives and ToolFace determine how a tool is called and which tools exist, but not which of them a query needs, where their argument values come from, or what to do when a call returns the wrong thing. HEART supplies these three in turn. Request decomposition turns the user query q into an invocation plan $\Pi = (\pi_1, \ldots, \pi_K)$ over K steps, or returns a targeted clarification request when the current context $\mathcal{C}_t$ cannot support one. Value resolution grounds the values of each step π_k from $\mathcal{C}_t$ into a natural-language invocation request x_k, together with execution hyperparameters, and dispatches it to the corresponding Primitive $\mathcal{P}_k$, which validates and binds the described arguments against its own schema. Result validation then evaluates each result r_k against four criteria, task completion, argument consistency, execution validity, and constraint satisfaction; on failure it emits structured feedback ϕ_k that enriches $\mathcal{C}_t$ and triggers a targeted revision.

Each mechanism exists because the abstraction leaves something undetermined. Decomposition is needed because a primitive answers the request it is handed and cannot ask what the user actually wants: the sufficiency check resolves ambiguity before execution begins, rather than committing to a plan built on missing arguments and discovering the gap only after a call has failed. Value resolution is needed because a step's values come from two places the primitive cannot see, the user query and the results of earlier steps, so placeholders must be grounded before dispatch even though schema binding itself happens inside the primitive. Validation is needed because a primitive reports whether its own call executed, not whether the result answers the query; it supplies that judgment and closes the loop by feeding its diagnosis back into the plan:

$$\mathcal{C}_{t+1} = \mathcal{C}_t \cup \{\phi_k\}, \quad \Pi' = \textsc{Decompose}(q,\, \mathcal{C}_{t+1}), \tag{2}$$

so a failed call is re-planned with enriched context instead of terminating the trajectory. Revision repeats for at most B rounds, after which the current result is returned to the user.

All three mechanisms are realized by prompting the same backbone and differ only by system prompt; Sec. 4.3 quantifies what each contributes and how performance varies with B. Complete specifications, criterion definitions, the end-to-end execution flow, and prompts are given in Appendix B, Appendix C, Appendix D, and Appendix E.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Benchmarks. We evaluate HEART on five benchmarks: ToolBench (Qin et al., 2023) covers large-scale real-world API invocation across single- and multi-tool scenarios. NESTFUL (Basu et al., 2025) evaluates nested API call sequences where outputs of one call serve as inputs to the next. τ^2-Bench (Barres et al., 2025) assesses agents under dual-control conversational settings where both agent and user invoke tools in a shared environment. ACEBench (Chen et al., 2025) targets robustness and fine-grained tool-use evaluation across complex multi-tool workflows. BFCLv4 (Patil et al., 2025) focuses on agentic web search and memory function calling.

Baselines. Beyond benchmark-specific baselines, we include frontier models including GPT-5.4 (OpenAI, 2025b), Claude-4.6-Sonnet (Anthropic, 2026), and Gemini-3.1-Pro (DeepMind, 2025) as shared baselines across all benchmarks.

Implementation Details. We use Qwen3-8B as the backbone LLM throughout HEART, for decomposition, value resolution, validation, and the Tool Primitives themselves. All experiments are conducted with a maximum revision budget of $B = 3$. Tool retrieval from ToolFace is performed via semantic search over tool schema descriptors; retrieval quality and its sensitivity to repository size are reported in Appendix F.4. We report average results over three runs, and over eight on τ^2-Bench.

4.2 MAIN RESULTS

On ToolBench. We compare HEART against ToolBench (Qin et al., 2023)'s baselines including Vicuna (Zheng et al., 2023), Alpaca (Taori et al., 2023), ToolLLaMA (Qin et al., 2023), GPT-3.5-Turbo (Ouyang et al., 2022), Claude-2 (Anthropic, 2023), Text-Davinci-003 (Brown et al., 2020), and GPT-4 (Achiam et al., 2023). We report Pass Rate and Win Rate across six single- and multi-tool (I1, I2, I3) scenarios.

HEART tops the ToolBench leaderboard (Table 1) with an average Pass Rate of 75.1% and Win Rate of 75.7%. These figures represent gains of $+1.7\%$/$+1.9\%$ over the strongest commercial baseline Claude-4.6-Sonnet (DFSDT) and $+7.8\%$/$+12.6\%$ over the strongest fine-tuned baseline ToolLLaMA (DFSDT-Retriever). The performance advantage of HEART is most pronounced on I3-Inst (77.2% Pass, 89.4% Win), the hardest subset involving multi-tool instructions requiring cross-API dependencies, where HEART surpasses Claude-4.6-Sonnet with DFSDT by $+3.4\%$ on Pass Rate and $+2.9\%$ on Win Rate. This consistent advantage across difficulty levels demonstrates that HEART's structured decomposition stage-resolution stage-validation stage pipeline generalizes effectively to large-scale real-world API invocation without relying on explicit search strategies such as DFSDT.

Table 1: Performance comparison (%) of different methods on ToolBench.

Model	Method	I1-Inst.		I1-Tool		I1-Cat.		I2-Inst.		I2-Cat.		I3-Inst.		Avg.	
		Pass	Win	Pass	Win	Pass	Win	Pass	Win	Pass	Win	Pass	Win	Pass	Win
Vicuna	ReACT & DFSDT	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Alpaca	ReACT & DFSDT	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ToolLLaMA	ReACT	25.0	45.0	29.0	42.0	33.0	47.5	30.5	50.8	31.5	41.8	25.0	55.0	29.0	47.0
	DFSDT	57.0	55.0	61.0	55.3	62.0	54.5	77.0	68.5	77.0	58.0	66.0	69.0	66.7	60.0
	DFSDT-Retriever	64.0	62.3	64.0	59.0	60.5	55.0	81.5	68.5	68.5	60.8	65.0	73.0	67.3	63.1
GPT-3.5-Turbo	ReACT	41.5	–	44.0	–	44.5	–	42.5	–	46.5	–	22.0	–	40.2	–
	DFSDT	54.5	60.5	65.0	62.0	60.5	57.3	75.0	72.0	71.5	64.8	62.0	69.0	64.8	64.3
Claude-2	ReACT	5.5	31.0	3.5	27.8	5.5	33.8	6.0	35.0	6.0	31.5	14.0	47.5	6.8	34.4
	DFSDT	20.5	38.0	31.0	44.3	18.5	43.3	17.0	36.8	20.5	33.5	28.0	65.0	22.6	43.5
Text-Davinci-003	ReACT	12.0	28.5	20.0	35.3	20.0	31.0	8.5	29.8	14.5	29.8	24.0	45.0	16.5	33.2
	DFSDT	43.5	40.3	44.0	43.8	46.0	46.8	37.0	40.5	42.0	43.3	46.0	63.0	43.1	46.3
GPT-4	ReACT	53.5	60.0	50.0	58.8	53.5	63.5	67.0	65.8	72.0	60.3	47.0	78.0	57.2	64.4
	DFSDT	60.0	67.5	71.5	67.8	67.0	66.5	79.5	73.3	77.5	63.3	71.0	84.0	71.1	70.4
GPT-5.4	ReACT	54.5	62.0	52.8	58.5	54.2	63.5	65.4	65.8	71.5	62.0	46.5	76.5	57.5	64.7
	DFSDT	61.2	67.5	64.5	65.2	63.8	64.2	76.5	71.2	76.2	65.8	68.4	81.2	68.8	69.2
Claude-4.6-Sonnet	ReACT	57.4	65.2	55.2	61.5	57.8	67.5	70.2	68.4	75.4	64.5	51.2	81.5	61.2	68.1
	DFSDT	65.4	70.8	70.5	69.5	68.4	70.2	81.5	75.6	80.5	69.4	73.8	86.5	73.4	73.8
Gemini-3.1-Pro	ReACT	56.2	63.8	53.8	60.5	56.5	65.4	68.4	67.5	73.2	63.2	49.5	79.5	59.6	66.6
	DFSDT	63.8	69.2	67.5	66.8	66.2	67.5	79.5	73.4	78.4	67.8	71.2	84.5	71.1	71.5
HEART	/	**67.5**	**72.8**	**71.5**	**70.8**	**69.5**	**72.1**	**83.4**	**77.5**	**81.6**	**71.5**	**77.2**	**89.4**	**75.1**	**75.7**

On NESTFUL. Following NESTFUL (Basu et al., 2025), we compare HEART against a broad set of baselines, including the xLAM series (Zhang et al., 2024; Liu et al., 2024d), Hammer (Lin et al., 2024), ToolACE (Liu et al., 2024c), Llama-3.1 (Grattafiori et al., 2024), Mixtral (Jiang et al., 2024),

Granite-20B (Abdelaziz et al., 2024), DeepSeek-V3 (Liu et al., 2024a), and GPT-4o (Hurst et al., 2024). We report F1 score for function name prediction (F1 Func.) and parameter name prediction (F1 Param.), partial sequence match accuracy (Part. Acc.), full sequence match accuracy (Full Acc.), and win rate (Win Rate), which measures whether all predicted APIs are executable and lead to the gold answer, under one-shot and three-shot ICL settings.

Table 2: Results on NESTFUL under one-shot and three-shot in-context learning (ICL).

Model	#Params	One-shot ICL					Three-shot ICL				
		F1 Func.	F1-Param.	Part Acc.	Full Acc.	Win Rate	F1 Func.	F1-Param.	Part Acc.	Full Acc.	Win Rate
xLAM-1b-fc-r	1B	0.19	0.08	0.09	0.00	0.01	0.22	0.09	0.09	0.03	0.02
xLAM-2-1b-fc-r	1B	0.41	0.13	0.13	0.00	0.00	0.43	0.12	0.13	0.00	0.00
xLAM-7b-fc-r	7B	0.49	0.17	0.15	0.00	0.03	0.55	0.23	0.23	0.15	0.14
xLAM-2-8b-fc-r	8B	0.48	0.15	0.14	0.00	0.01	0.47	0.17	0.15	0.04	0.04
Hammer2.0-7b	7B	0.56	0.24	0.21	0.07	0.16	0.61	0.30	0.29	0.22	0.25
Hammer2.1-7b	7B	0.10	0.05	0.05	0.01	0.01	0.16	0.10	0.11	0.08	0.08
Llama-3.1-8B-Instruct	8B	0.64	0.19	0.17	0.06	0.06	0.63	0.22	0.22	0.16	0.11
ToolACE-8B	8B	0.43	0.13	0.13	0.00	0.00	0.50	0.15	0.13	0.00	0.00
ToolACE-2-Llama-3.1-8B	8B	0.28	0.10	0.13	0.00	0.00	0.29	0.10	0.13	0.00	0.00
Granite-20B-FunctionCalling	20B	0.64	0.20	0.17	0.02	0.05	0.61	0.25	0.26	0.21	0.20
Mixtral-8x7B-Instruct-v0.1	46.7B	0.22	0.07	0.05	0.00	0.01	0.32	0.13	0.14	0.09	0.09
xLAM-8x7b-fc-r	46.7B	0.40	0.15	0.16	0.01	0.01	0.43	0.16	0.17	0.02	0.03
Llama-3.1-70B-Instruct	70B	0.41	0.19	0.15	0.04	0.09	0.33	0.17	0.15	0.07	0.11
Mixtral-8x22B-Instruct-v0.1	141B	0.49	0.21	0.17	0.06	0.07	0.65	0.29	0.28	0.21	0.23
xLAM-8x22b-fc-r	141B	0.53	0.21	0.22	0.12	0.03	0.50	0.23	0.25	0.17	0.06
Llama-3.1-405B-Instruct-fp8	405B	0.41	0.14	0.08	0.03	0.10	0.41	0.18	0.13	0.07	0.14
DeepSeek-V3	685B	0.69	0.36	0.27	0.09	0.43	0.69	0.42	0.37	0.29	0.60
GPT-4o (2024-08-06)	UNK	0.73	0.41	0.38	0.28	0.59	0.74	0.41	0.38	0.28	0.60
GPT-5.4	UNK	0.78	0.48	0.50	0.40	0.66	0.80	0.50	0.52	0.42	0.68
Claude-4.6-Sonnet	UNK	0.77	0.47	0.49	0.39	0.65	0.79	0.49	0.51	0.41	0.67
Gemini-3.1-Pro	UNK	0.76	0.46	0.48	0.38	0.63	0.78	0.48	0.50	0.40	0.65
HEART	8B * 4	**0.82**	**0.52**	**0.55**	**0.44**	**0.75**	**0.84**	**0.56**	**0.58**	**0.47**	**0.79**

Table 3: Performance comparison of different methods across domains on the τ^2-Bench benchmark.

Method	Domain	Pass[1]	Pass[2]	Pass[3]	Pass[4]	Tokens / Cost ($)	Latency (s)
GPT-4.1	Retail	0.75	0.64	0.58	0.53	7,401 / 0.0592	14.28
	Airline	0.56	0.45	0.42	0.40	7,825 / 0.0626	15.62
	Telecom	0.34	0.26	0.22	0.19	6,862 / 0.0549	34.41
GPT-o4-mini	Retail	0.71	0.59	0.52	0.46	5127 / 0.0031	4.24
	Airline	0.59	0.48	0.42	0.38	5389 / 0.0032	4.68
	Telecom	0.42	0.33	0.29	0.26	4856 / 0.0029	26.05
GPT-4.1-mini	Retail	0.66	0.52	0.44	0.39	5,214 / 0.0083	6.12
	Airline	0.51	0.39	0.32	0.26	5,563 / 0.0089	6.81
	Telecom	0.44	0.30	0.22	0.18	5,021 / 0.0080	21.94
Claude-3.7-sonnet	Retail	0.79	0.69	0.63	0.60	6,482 / 0.0972	11.45
	Airline	0.50	0.41	0.38	0.36	6,745 / 0.1012	12.32
	Telecom	0.49	0.37	0.30	0.25	6,038 / 0.0906	30.88
GPT-5.4	Retail	0.70	0.53	0.43	0.37	7,842 / 0.1176	16.24
	Airline	0.64	0.55	0.50	0.48	8,126 / 0.1219	18.42
	Telecom	0.56	0.40	0.34	0.29	7,319 / 0.1098	35.11
Claude-4.6-sonnet	Retail	0.80	0.72	0.66	0.61	7,153 / 0.1073	14.12
	Airline	0.72	0.62	0.55	0.50	7,412 / 0.1112	16.52
	Telecom	0.58	0.47	0.39	0.33	6,728 / 0.1009	33.48
Gemini-3.1-Pro	Retail	0.68	0.54	0.46	0.40	6,914 / 0.0830	9.15
	Airline	0.54	0.41	0.33	0.29	7,258 / 0.0871	10.82
	Telecom	0.55	0.42	0.34	0.30	6,492 / 0.0779	28.46
HEART	Retail	**0.82**	**0.78**	**0.76**	**0.75**	21,684 / 0.0152	18.72
	Airline	**0.73**	**0.68**	**0.65**	**0.63**	24,531 / 0.0172	21.34
	Telecom	**0.63**	**0.57**	**0.53**	**0.50**	20,887 / 0.0146	37.95

Table 2 shows that HEART achieves the best performance on all five metrics under both one-shot and three-shot ICL. Against the strongest commercial baseline GPT-5.4, HEART improves Full Acc. by $+4\%/+5\%$ and Win Rate by $+9\%/+11\%$ under one-shot/three-shot. Strikingly, several tool-calling models, including all xLAM variants up to 8B and both ToolACE models, achieve Full Acc. $= 0.00$ under one-shot, confirming that flat tool-use trajectories do not generalize to nested API sequences. We also observe that raw model scale does not compensate for the structured decomposition: DeepSeek-V3 (685B) reaches only 0.09 Full Acc., far below HEART's 0.44.

On τ^2-Bench. Following the τ^2-Bench (Barres et al., 2025), we include GPT-4.1 (OpenAI, 2025a), GPT-o4-mini (OpenAI, 2025c), GPT-4.1-mini (OpenAI, 2025a), Claude-3.7-Sonnet (Anthropic, 2025b). and report Pass^k ($k = 1, 2, 3, 4$), the fraction of tasks for which all k independent trials

succeed, following the official τ^2-Bench protocol. We also report average per task token consumption, API cost (USD) (Detailed in Appendix F.1), and latency (seconds).

Table 3 shows that HEART achieves the best Pass^k scores across all domains and all k. The performance gap widens as k increases: while Pass^1 margins over the strongest commercial baseline Claude-4.6-Sonnet remain modest, HEART maintains higher Pass^4 scores (e.g., $+51.5\%$ in Telecom), demonstrating that iterative revision sustains task completion under extended multi-turn interactions where commercial models degrade. Telecom consistently poses the greatest challenge; the best baseline reaches only 0.33 on Pass^4, while HEART achieves 0.50. Despite higher token consumption from the additional LLM call each invocation step carries, HEART reduces API cost by up to $7.7\times$ relative to GPT-5.4.

Table 4: Performance comparison (%) of different methods across domains on the ACEBench.

Model	Normal						Special	Agent	Overall
	Atom	Single-Turn	Multi-Turn	Similar API	Preference	Summary			
GPT-4o	93.4	84.5	77.0	85.0	83.0	87.6	93.0	63.8	85.4
GPT-4-Turbo	93.2	84.8	77.5	86.0	86.0	88.0	86.7	67.5	84.5
Qwen-Max	91.2	80.5	68.0	83.0	83.0	84.2	74.0	64.3	78.4
GPT-4o-Mini	86.5	76.0	66.5	77.0	78.0	79.9	79.0	33.3	72.5
Gemini-1.5-Pro	84.5	76.8	64.5	80.0	78.0	79.0	78.7	25.5	70.7
Claude-3.5-Sonnet	76.9	72.5	62.5	71.0	72.0	72.9	77.4	39.5	68.9
GPT-5.4	94.0	86.0	81.0	88.0	85.0	89.5	94.0	69.5	86.0
Claude-4.6-Sonnet	93.8	85.5	79.5	87.0	84.0	88.3	91.3	66.8	84.3
Gemini-3.1-Pro	92.5	83.0	76.0	85.0	82.0	86.6	89.0	64.5	82.3
HEART	**94.2**	**87.5**	**83.5**	**89.0**	**86.0**	**89.8**	**95.0**	**72.3**	**86.9**

Table 5: Performance comparison (%) of different methods on BFCLv4.

Model	Web Search			Memory			
	Base	No Snippet	Overall	KV	Vector	Recursive Sum	Overall
xLAM-2-32B-FC-R	37.00	14.00	25.50	6.45	10.32	45.81	20.86
xLAM-2-70B-FC-R	15.00	17.00	13.00	2.58	10.97	29.68	14.41
xLAM-2-8B-FC-R	11.00	2.00	6.50	5.81	15.48	20.65	13.98
xLAM-2-3B-FC-R	3.00	2.00	2.50	5.81	5.81	22.58	11.40
xLAM-2-1B-FC-R	0.00	0.00	0.00	3.87	3.87	3.87	3.87
Hammer2.1-7B	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Hammer2.1-3B	0.00	0.00	0.00	2.58	3.87	2.58	3.01
LLaMA-3.1-8B-Instruct	6.00	0.00	3.00	7.74	5.81	18.71	10.75
LLaMA-3.2-3B-Instruct	2.00	0.00	1.00	3.23	3.23	12.26	6.24
Qwen3-8B	15.00	9.00	12.00	5.16	7.10	31.61	14.62
Qwen3-32B	25.00	18.00	21.50	12.26	25.81	41.94	26.67
GPT-4.1	67.00	69.00	68.00	16.13	18.06	37.42	23.87
GPT-4.1-Mini	62.00	52.00	57.00	22.58	16.13	41.94	26.88
Gemini-1.5-Pro	60.00	64.00	62.00	19.35	24.52	58.71	34.19
Claude-3.5-Sonnet	60.00	64.00	62.00	13.55	47.10	55.48	38.71
GPT-5.4	81.00	83.00	82.00	57.42	58.71	51.61	55.91
Claude-4.6-Sonnet	85.00	81.00	83.00	64.19	67.42	73.23	68.28
Gemini-3.1-Pro	80.00	78.00	79.00	59.35	62.58	63.23	61.72
HEART	**87.00**	**85.00**	**86.00**	**68.47**	**69.10**	**70.58**	**69.38**

On ACEBench. We report performance across six Normal subcategories (Atom, Single-Turn, Multi-Turn, Similar API, Preference, Summary), the Special category, the Agent category, and an Overall score. We compare HEART against commercial models in Table 4; results against open-source baselines including Qwen2.5 series (Yang et al., 2024), Llama-3.1 (Grattafiori et al., 2024), DeepSeek-V3 (Liu et al., 2024a), Phi-3 (Abdin et al., 2024), and tool-calling models such as Hammer (Lin et al., 2024), xLAM (Liu et al., 2024d), and Watt-Tool-8B (AI, 2024) are in Appendix F.2.

Table 4 shows that HEART achieves the highest Overall score (86.9%), surpassing GPT-5.4 (86.0%) by $+0.9\%$. While margins on Normal subcategories are modest, the clearest advantages appear on Special (95.0% vs. 94.0%) and Agent (72.3% vs. 69.5%), which require handling out-of-distribution tool formats and multi-step agentic reasoning. On Multi-Turn (83.5%), HEART outperforms GPT-5.4 by $+2.5\%$, reflecting the benefit of iterative revision.

On BFCLv4. Following BFCLv4 (Patil et al., 2025), we compare HEART against a diverse set of baselines, including the xLAM series (Liu et al., 2024d), Hammer (Lin et al., 2024), LLaMA models (Grattafiori et al., 2024), Qwen3 (Yang et al., 2025), GPT-4.1 (OpenAI, 2025a), GPT-4.1-Mini (OpenAI, 2025a), Gemini-1.5-Pro (Team et al., 2024), and Claude-3.5-Sonnet (Anthropic,

2024). We evaluate Web Search function calling under Base and No-Snippet settings, and Memory function calling across KV Retrieval, Vector Retrieval, and Recursive Summarization, reporting the averaged Overall score for each.

Table 5 reports results across all models. HEART achieves the best performance on both Web Search Overall (86.0%) and Memory Overall (69.38%), surpassing Claude-Sonnet-4.6 by $+3.0\%$ and $+1.10\%$ respectively. The memory advantage over GPT-5.4 is more substantial, with HEART outperforming by $+13.47\%$ on Memory Overall, including gains of $+11.05\%$, $+10.39\%$, and $+18.97\%$ on KV, Vector, and Recursive Summarization. SFT-based models fail almost entirely: Hammer2.1-7B scores 0.0% across all metrics, and the strongest SFT baseline xLAM-2-32B-FC-R reaches only 25.5%/20.86%, showing no generalization to dynamic agentic tool scenarios.

4.3 Ablation Study

Component Contribution. We evaluate four HEART ablations on ToolBench and NESTFUL, each removing one core component: the decomposition stage, the resolution stage, ToolFace & Tool Primitives, or the validation stage. Appendix F.7 specifies each variant. Table 6 reports results.

The most dramatic degradation is observed under **w/o ToolFace & Tool Primitives**, which reduces ToolBench Avg. Pass Rate from 75.1 to 16.1 and NESTFUL Full Acc. from 0.44 to 0.06, confirming that the natural language interface for tools is the most critical component in HEART. Removing the validation stage produces the second-largest drop (Pass Rate 47.6, Full Acc. 0.15), demonstrating that feedback-driven revision is essential for recovering from execution failures. The w/o Decomposition and w/o Resolution variants show moderate but consistent degradation, confirming that both contribute necessary intermediary layers. The spread across these variants is itself informative: removing the interface costs 59.0 points of ToolBench Pass Rate, whereas removing any single harness stage costs between 12.7 and 27.5. The gains therefore come from the abstraction rather than from the stages assembled around it, and the budget-matched control in Appendix F.8 shows that this is not simply the extra LLM call the interface carries.

Table 6: Ablation study on component contribution evaluated on ToolBench and NESTFUL.

Variant	ToolBench		NESTFUL	
	Avg. Pass	Win Rate	Full Acc.	Win Rate
w/o Decomposition	57.2	64.1	0.26	0.51
w/o Resolution	62.4	67.3	0.28	0.52
w/o ToolFace & Tool Primitives	16.1	27.9	0.06	0.06
w/o Validation	47.6	49.6	0.15	0.33
HEART	**75.1**	**75.7**	**0.44**	**0.75**

Removing Tool Primitives also removes one LLM call per invocation step. To separate the interface from the reasoning capacity carrying it, we reallocate the same call budget to resolution stage self-refinement over raw schemas; this recovers only part of the gap ($16.1 \rightarrow 34.2$ Avg. Pass), leaving full HEART at 75.1 (Appendix F.8). Holding the backbone and tool catalogue fixed at Qwen3-8B and ToolFace, HEART also leads the strongest matched-backbone scaffold (DFSDT) by $+19.3$ Avg. Pass on ToolBench and $+0.31$ Full Acc. on NESTFUL (Appendix F.6), indicating the gains track the harness rather than the backbone.

Revision Budget. We study the effect of the maximum revision budget B by varying $B \in \{0, 1, 2, 3, 5\}$ and evaluating on ToolBench and NESTFUL, where $B = 0$ disables revision entirely, reducing HEART to a single-pass execution pipeline without feedback-driven recovery.

Table 7: Effect of revision budget B on ToolBench and NESTFUL.

Budget B	ToolBench		NESTFUL	
	Avg. Pass	Win Rate	Full Acc.	Win Rate
$B = 0$	47.6	49.6	0.15	0.33
$B = 1$	51.4	57.9	0.27	0.49
$B = 2$	65.9	66.8	0.42	0.68
$B = 3$	75.1	75.7	0.44	0.75
$B = 5$	75.3	76.2	0.44	0.75

Table 7 reports results. Performance improves consistently from $B = 0$ to $B = 3$ across both benchmarks, with the largest gains concentrated between $B = 1$ and $B = 2$: ToolBench Pass Rate jumps from 51.4 to 65.9 and NESTFUL Full Acc. from 0.27 to 0.42, indicating that the second revision round resolves the majority of recoverable failures. $B = 5$ yields negligible further improvement, confirming saturation at $B = 3$, which we adopt as the default.

4.4 Case Study

End-to-End Real World Task. To evaluate HEART in realistic deployment conditions, we construct a benchmark of 50 end-to-end real-world tasks spanning five domains: *Travel Planning*, *Healthcare*,

E-commerce, *Finance*, and *Local Services* with 10 tasks per domain. Each task is grounded in a concrete user persona (e.g., a student booking a flight, a group organizer finding catering) and expressed as a request requiring multi-step tool invocation to resolve. Tasks range from single-tool lookups (e.g., retrieving gas prices) to multi-tool workflows involving search, comparison, and conditional booking across heterogeneous APIs. Details are in Appendix H.

Table 8 shows that HEART achieves 84% task completion, outperforming GPT-5.4 (20%), Claude-4.6-Sonnet (24%), and Gemini-3.1-Pro (22%) by over 60%. The gap reveals a critical divide between recommendation and execution: commercial models reliably retrieve options but fail when tasks require acting on behalf of the user, whereas HEART's grounded argument mapping and stateful execution enable it to complete tasks that require acting in the world. Here the commercial models are evaluated through their product interfaces with their own built-in tool ecosystems; granting them the ToolFace tools via native function calling raises their completion to 22–32%, so tool access accounts for only a small fraction of the gap (Appendix F.9).

Table 8: Task Completion Rate (%) on 50 tasks across five domains. Each domain contains 10 tasks evaluated by human annotators.

Domain	GPT-5.4	Claude-4.6-Sonnet	Gemini-3.1-Pro	HEART
Travel Planning	20	20	10	**80**
Healthcare	20	30	30	**70**
E-commerce	10	10	20	**80**
Finance	30	30	30	**100**
Local Services	20	30	20	**90**
Overall	20	24	22	**84**

Robustness against Prompt Injection Attack. Existing retrieve-then-select paradigms expose tool metadata in the model's input prompt, creating an attack surface for prompt injection. Following ToolHijacker (Shi et al., 2025), we consider an attacker who injects a malicious tool document into the retrieved candidate set to manipulate tool selection, and report ACC under no attack alongside ASR under Gradient-Free and Gradient-Based strategies on ToolBench.

Table 9: Tool selection accuracy (ACC%) and attack success rate (ASR%) on ToolBench. HEART achieves 0.0% ASR under both attack strategies because tool schemas are stored in ToolFace and invoked via Tool Primitives, never appearing in the LLM's input prompt and thus eliminating the attack surface by construction.

Attack	Metric	GPT-5.4	Claude-4.6-Sonnet	Gemini-3.1	HEART
No Attack	ACC	99.0	99.0	98.6	–
Gradient-Free	ASR	82.2	74.4	78.8	**0.0**
Gradient-Based	ASR	70.6	66.0	72.2	**0.0**

As shown in Table 9, all models remain highly vulnerable, with ASR reaching up to 70% under Gradient-Free attack. HEART achieves 0.0% ASR under both strategies. Tool schemas are stored in ToolFace and executed via Tool Primitives, never appearing in the LLM's input prompt. The attack surface that ToolHijacker exploits does not exist.

4.5 Discussion

The results highlight three properties of Tool Primitives. **(1) Natural language invocation.** Primitives resolve schemas internally; removing this capability drops ToolBench Pass from 75.1 to 16.1, while a budget-matched control reaches only 34.2 (Table 6; Appendix F.8). Keeping schemas outside the caller context also yields 0.00% injection ASR (Table 9). **(2) Inter-tool communication.** Passing results between Primitives gives HEART 0.44 Full Acc. on NESTFUL, versus 0.00 for SFT models and 0.09 for DeepSeek-V3 (Table 2). **(3) Execution isolation.** Structured failures enable validation and retry: HEART maintains Pass^4 on τ^2-Bench, with most failures resolved within $B = 3$ rounds and validation reaching $0.97/0.96$ precision/recall (Table 3; Table 7; Appendix F.5). Limitations are in Appendix G.

5 Conclusion

We propose HEART, a harness engineering framework that addresses key limitations of existing LLM tool-use systems, including scalability to large tool catalogs, robustness in multi-turn interaction, compositional reasoning, and failure recovery. By introducing Tool Primitives and a harness design with request decomposition, value resolution, and result validation, HEART transforms tool use into a structured and iterative reasoning process. Experiments across multiple benchmarks show that HEART consistently improves performance, efficiency, and robustness over existing approaches, including strong resilience to real-world challenges such as prompt injection. We hope our proposed HEART framework can be the real heart of LLM tool use.

AI USE STATEMENT

AI assistants were used as auxiliary tools for manuscript preparation, including language polishing, clarity improvement, organization, and limited experimental workflows. All experimental design, methodological decisions, analyses, reported results, and final content were reviewed and verified by the authors.

ETHICS STATEMENT

This work involves no human subjects and no personally identifying data. The 50 real-world tasks we curate are authored by the authors and contain placeholder user details rather than information about any third party; all executions were carried out on accounts belonging to the authors. ToolFace wraps publicly documented APIs and the tool inventories of five public benchmarks, used in accordance with their respective licenses and terms of service.

REPRODUCIBILITY STATEMENT

HEART is training-free: all roles are instantiated from an off-the-shelf backbone and differ only by system prompt, so reproduction requires no weight updates. The backbone, revision budget, retrieval configuration, and averaging protocol are specified in Sec. 4.1; the role specifications, validation criteria, end-to-end execution flow, and the complete prompt for every role are given in Appendices B, C, D, and E. All five benchmarks are public and evaluated under their official protocols and metrics, and the per-token prices used for the cost analysis are listed in Appendix F.1. Appendix H links an anonymous repository containing the ToolFace schemas and executable implementations, the retrieval index and its configuration, all prompts, and the 50 real-world tasks with their tool lists.

REFERENCES

Ibrahim Abdelaziz, Kinjal Basu, Mayank Agarwal, Sadhana Kumaravel, Matthew Stallone, Rameswar Panda, Yara Rizk, GP Shrivatsa Bhargav, Maxwell Crouse, Chulaka Gunasekara, et al. Granite-function calling model: Introducing function calling abilities via multi-task learning of granular tasks. In Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track, pp. 1131–1139, 2024.

Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, Alon Benhaim, Misha Bilenko, Johan Bjorck, Sébastien Bubeck, Martin Cai, Qin Cai, Vishrav Chaudhary, Dong Chen, Dongdong Chen, Weizhu Chen, Yen-Chun Chen, Yi-Ling Chen, Hao Cheng, Parul Chopra, Xiyang Dai, Matthew Dixon, Ronen Eldan, Victor Fragoso, Jianfeng Gao, Mei Gao, Min Gao, Amit Garg, Allie Del Giorno, Abhishek Goswami, Suriya Gunasekar, Emman Haider, Junheng Hao, Russell J. Hewett, Wenxiang Hu, Jamie Huynh, Dan Iter, Sam Ade Jacobs, Mojan Javaheripi, Xin Jin, Nikos Karampatziakis, Piero Kauffmann, Mahoud Khademi, Dongwoo Kim, Young Jin Kim, Lev Kurilenko, James R. Lee, Yin Tat Lee, Yuanzhi Li, Yunsheng Li, Chen Liang, Lars Liden, Xihui Lin, Zeqi Lin, Ce Liu, Liyuan Liu, Mengchen Liu, Weishung Liu, Xiaodong Liu, Chong Luo, Piyush Madan, Ali Mahmoudzadeh, David Majercak, Matt Mazzola, Caio César Teodoro Mendes, Arindam Mitra, Hardik Modi, Anh Nguyen, Brandon Norick, Barun Patra, Daniel Perez-Becker, Thomas Portet, Reid Pryzant, Heyang Qin, Marko Radmilac, Liliang Ren, Gustavo de Rosa, Corby Rosset, Sambudha Roy, Olatunji Ruwase, Olli Saarikivi, Amin Saied, Adil Salim, Michael Santacroce, Shital Shah, Ning Shang, Hiteshi Sharma, Yelong Shen, Swadheen Shukla, Xia Song, Masahiro Tanaka, Andrea Tupini, Praneetha Vaddamanu, Chunyu Wang, Guanhua Wang, Lijuan Wang, Shuohang Wang, Xin Wang, Yu Wang, Rachel Ward, Wen Wen, Philipp Witte, Haiping Wu, Xiaoxia Wu, Michael Wyatt, Bin Xiao, Can Xu, Jiahang Xu, Weijian Xu, Jilong Xue, Sonali Yadav, Fan Yang, Jianwei Yang, Yifan Yang, Ziyi Yang, Donghan Yu, Lu Yuan, Chenruidong Zhang, Cyril Zhang, Jianwen Zhang, Li Lyna Zhang, Yi Zhang, Yue Zhang, Yunan Zhang, and Xiren Zhou. Phi-3 technical report: A highly capable language model locally on your phone. https://arxiv.org/abs/2404.14219, 2024.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. arXiv preprint arXiv:2303.08774, 2023.

Watt AI. Watt-tool-8b. https://huggingface.co/watt-ai/watt-tool-8B, 2024.

Anthropic. Claude 2. https://www.anthropic.com/news/claude-2, 2023.

Anthropic. Claude 3.5 sonnet. https://www.anthropic.com/news/claude-3-5-sonnet, 2024.

Anthropic. Agent skills. https://platform.claude.com/docs/en/agents-and-tools/agent-skills/overview, 2025a.

Anthropic. Claude 3.7 sonnet and claude code. https://www.anthropic.com/news/claude-3-7-sonnet, 2025b.

Anthropic. Introducing claude sonnet 4.6. https://www.anthropic.com/news/claude-sonnet-4-6, 2026.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. arXiv preprint arXiv:2506.07982, 2025.

Kinjal Basu, Ibrahim Abdelaziz, Subhajit Chaudhury, Soham Dan, Maxwell Crouse, Asim Munawar, Vernon Austel, Sadhana Kumaravel, Vinod Muthusamy, Pavan Kapanipathi, et al. Api-blend: A comprehensive corpora for training and benchmarking api llms. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 12859–12870, 2024.

Kinjal Basu, Ibrahim Abdelaziz, Kiran Kate, Mayank Agarwal, Maxwell Crouse, Yara Rizk, Kelsey Bradford, Asim Munawar, Sadhana Kumaravel, Saurabh Goyal, et al. Nestful: A benchmark for evaluating llms on nested sequences of api calls. In Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pp. 33526–33535, 2025.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. Advances in neural information processing systems, 33:1877–1901, 2020.

Chen Chen, Xinlong Hao, Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Yuefeng Huang, Xiangcheng Liu, Wang Xinzhi, et al. Acebench: A comprehensive evaluation of llm tool usage. Findings of the Association for Computational Linguistics: EMNLP, 2025:12970–12998, 2025.

Google DeepMind. Gemini 3.1 pro: A smarter model for your most complex tasks. https://blog.google/innovation-and-ai/models-and-research/gemini-models/gemini-3-1-pro/, 2025.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. arXiv preprint arXiv:2407.21783, 2024.

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, 2024.

Shijue Huang, Wanjun Zhong, Jianqiao Lu, Qi Zhu, Jiahui Gao, Weiwen Liu, Yutai Hou, Xingshan Zeng, Yasheng Wang, Lifeng Shang, et al. Planning, creation, usage: Benchmarking llms for comprehensive tool utilization in real-world complex scenarios. In Findings of the Association for Computational Linguistics: ACL 2024, pp. 4363–4400, 2024.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. arXiv preprint arXiv:2410.21276, 2024.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. arXiv preprint arXiv:2401.04088, 2024.

Haibo Jin, Kuang Peng, Ye Yu, Xiaopeng Yuan, and Haohan Wang. Agent primitives: Reusable latent building blocks for multi-agent systems. arXiv preprint arXiv:2602.03695, 2026.

Kiran Kate, Tejaswini Pedapati, Kinjal Basu, Yara Rizk, Vijil Chenthamarakshan, Subhajit Chaudhury, Mayank Agarwal, and Ibrahim Abdelaziz. Longfunceval: Measuring the effectiveness of long context models for function calling. arXiv preprint arXiv:2505.10570, 2025.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. Api-bank: A comprehensive benchmark for tool-augmented llms. In Proceedings of the 2023 conference on empirical methods in natural language processing, pp. 3102–3116, 2023.

Qiqiang Lin, Muning Wen, Qiuying Peng, Guanyu Nie, Junwei Liao, Jun Wang, Xiaoyun Mo, Jiamu Zhou, Cheng Cheng, Yin Zhao, et al. Hammer: Robust function-calling for on-device language models via function masking. arXiv preprint arXiv:2410.04587, 2024.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. arXiv preprint arXiv:2412.19437, 2024a.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. Transactions of the association for computational linguistics, 12:157–173, 2024b.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, et al. Toolace: Winning the points of llm function calling. arXiv preprint arXiv:2409.00920, 2024c.

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, et al. Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets. Advances in Neural Information Processing Systems, 37:54463–54482, 2024d.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, et al. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. In Findings of the Association for Computational Linguistics: NAACL 2025, pp. 1160–1183, 2025.

OpenAI. Introducing gpt-4.1 in the api. https://openai.com/index/gpt-4-1/, 2025a.

OpenAI. Introducing gpt-5.4. https://openai.com/index/introducing-gpt-5-4/, 2025b.

OpenAI. Introducing openai o3 and o4-mini. https://openai.com/index/introducing-o3-and-o4-mini/, 2025c.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. Advances in neural information processing systems, 35:27730–27744, 2022.

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.

Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo Zhang, Tulika Awalgaonkar, Shiyu Wang, Zhiwei Liu, Haolin Chen, Thai Hoang, Juan Carlos Niebles, et al. Apigen-mt: Agentic pipeline for multi-turn data generation via simulated agent-human interplay. arXiv preprint arXiv:2504.03601, 2025.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. arXiv preprint arXiv:2307.16789, 2023.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. Advances in Neural Information Processing Systems, 36:38154–38180, 2023.

Jiawen Shi, Zenghui Yuan, Guiyao Tie, Pan Zhou, Neil Zhenqiang Gong, and Lichao Sun. Prompt injection attack to tool selection in llm agents. arXiv preprint arXiv:2504.19793, 2025.

Venkat Krishna Srinivasan, Zhen Dong, Banghua Zhu, Brian Yu, Damon Mosk-Aoyama, Kurt Keutzer, Jiantao Jiao, and Jian Zhang. Nexusraven: a commercially-permissive language model for function calling. In NeurIPS 2023 Foundation Models for Decision Making Workshop, 2023.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`, 2023.

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. arXiv preprint arXiv:2403.05530, 2024.

Jun Wang, Jiamu Zhou, Xihuai Wang, Xiaoyun Mo, Haoyu Zhang, Qiqiang Lin, Jincheng Jincheng, Muning Wen, Weinan Zhang, and Qiuying Peng. Hammerbench: Fine-grained function-calling evaluation in real mobile assistant scenarios. In Findings of the Association for Computational Linguistics: ACL 2025, pp. 3350–3376, 2025.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report. arXiv preprint arXiv:2412.15115, 2024.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In The eleventh international conference on learning representations, 2022.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. arXiv preprint arXiv:2406.12045, 2024.

Fan Yin, Zifeng Wang, I-Hung Hsu, Jun Yan, Ke Jiang, Yanfei Chen, Jindong Gu, Long T. Le, Kai-Wei Chang, Chen-Yu Lee, Hamid Palangi, and Tomas Pfister. Magnet: Multi-turn tool-use data synthesis and distillation via graph translation, 2025. URL `https://arxiv.org/abs/2503.07826`.

Yinger Zhang, Hui Cai, Xierui Song, Yicheng Chen, Rui Sun, and Jing Zheng. Reverse chain: A generic-rule for llms to master multi-api planning. In Findings of the Association for Computational Linguistics: NAACL 2024, pp. 302–325, 2024.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. Advances in neural information processing systems, 36:46595–46623, 2023.

A Extended Related Work

Tool-Use LLMs. Prior work falls into two complementary axes: tool coverage and execution complexity. Early approaches to tool invocation remain confined to simple, stateless execution. For example, Hammer (Lin et al., 2024) focuses on efficient on-device deployment but operates within a single-turn regime where tools are invoked once and immediately resolved. Moving beyond isolated calls, MAGNET (Yin et al., 2025) synthesizes multi-turn trajectories via graph translation, using local dependency graphs and node operations (Insert, Merge, Split) to generate training data covering nested function compositions, long-range cross-turn dependencies, and missing-parameter scenarios. ToolACE (Liu et al., 2024c) and the xLAM series (Liu et al., 2024d) move beyond isolated single-turn queries, curating multi-turn conversational datasets that capture extended back-and-forth interactions between users and agents. While these efforts progressively advance from single-turn invocation toward sustained multi-turn execution, they share a common limitation: the model is still required to emit schema-compliant arguments for every call, so composition remains bounded by how reliably it can map one tool's output into the next tool's input format.

Benchmarks for Tool Use Evaluation. Evaluating LLM tool use spans several dimensions. BFCLv4 (Patil et al., 2025) evaluates structured function calling across diverse APIs, while StableToolBench (Guo et al., 2024) improves ToolBench (Qin et al., 2023) by replacing unstable live APIs with deterministic virtual servers. NexusRaven (Srinivasan et al., 2023) focuses on schema-compliant JSON generation, API-Bank (Li et al., 2023) studies tool-augmented dialogue scenarios, and API-BLEND (Basu et al., 2024) evaluates compositional planning across heterogeneous APIs. NESTFUL (Basu et al., 2025) further targets nested API call sequences, showing that even GPT-4o achieves only 28% full-sequence match accuracy. For interactive and multi-turn settings, τ-Bench (Yao et al., 2024) and τ^2-Bench (Barres et al., 2025) evaluate agents under rule-constrained conversational environments, while HammerBench (Wang et al., 2025), ACEBench (Chen et al., 2025), and ToolSandbox (Lu et al., 2025) extend evaluation to mobile applications, complex multi-tool workflows, and stateful execution settings. While these benchmarks cover diverse tool-use scenarios, they primarily evaluate task completion under fixed tool environments.

Key Differences. Aforementioned methods typically expose tools directly to the model as raw schemas within the input context, whether through prompting-based invocation or fine-tuning on static tool-use trajectories. This tightly couples tool storage, representation, and invocation, leading to the challenges discussed earlier. HEART departs from this paradigm in three ways. First, HEART introduces Tool Primitives, which provide tools with natural language interfaces instead of exposing raw API schemas directly to the model. Second, HEART hosts ToolFace, enabling retrieval of only relevant Tool Primitives at inference time rather than loading large tool catalogues into context. Third, a Tool Primitive is defined by the tool it wraps rather than by an assigned role, so the set of primitives is induced by ToolFace and grows with it, and the path along which invocations are chained follows the data dependencies between calls rather than a fixed pipeline. HEART supplies only what this abstraction leaves open: which primitives a query needs, where their argument values come from, and how a failed result re-enters planning.

B Mechanism Specifications

We expand on the specifications of the three mechanisms introduced in the main text.

B.1 Query Decomposition

The decomposition serves as HEART's entry point, responsible for two sequential functions: **intent analysis** and **information sufficiency checking**. Upon receiving a user query q, decomposition first decomposes q into a structured intent representation that identifies the target task, relevant tool categories, and any explicit constraints or preferences expressed by the user. It then evaluates whether the current context $\mathcal{C}_t$—comprising the user query, any previously acquired information, and execution history—provides sufficient information to construct a fully specified invocation plan.

Formally, the sufficiency decision is

$$\delta_t = \textsc{Decompose}(q, \mathcal{C}_t) \in \{\texttt{sufficient}, \texttt{insufficient}\}. \tag{3}$$

If the sufficiency check returns `insufficient`, decomposition generates a targeted clarification request c_t and returns it to the user, initiating another interaction round. This loop continues until the context is judged sufficient, at which point decomposition produces an invocation plan $\Pi = (\pi_1, \pi_2, \ldots, \pi_K)$, an ordered sequence of K tool invocation steps, each π_k specifying the tool to be invoked at step k to address the user's request.

B.2 Value Resolution

Given an invocation plan Π from decomposition, value resolution is responsible for **parameter mapping** and **tool dispatch**. For each step $\pi_k \in \Pi$, which specifies the target Tool Primitive but contains no concrete argument values, value resolution resolves the required arguments from the current context $\mathcal{C}_t$ and constructs a natural language invocation request x_k that encodes the target tool, the intended operation, and the resolved parameter values, along with execution hyperparameters $\texttt{H-Params}_k$ such as retry count and priority level.

Formally,

$$\beta_k = \textsc{Resolve}(\pi_k,\, \mathcal{C}_t) = \big(x_k,\, \texttt{H-Params}_k\big), \tag{4}$$

where x_k encodes the target tool, the intended operation, and the resolved parameter values, and $\texttt{H-Params}_k$ specifies execution hyperparameters. This indirection sidesteps two persistent failure modes of dispatching to raw tool implementations, fragile parameter formatting and degraded selection accuracy under large tool catalogues, which directly motivates the Tool Primitive abstraction.

Note that $\mathcal{C}_t$ serves a different role here than in decomposition: whereas decomposition consults $\mathcal{C}_t$ to assess information sufficiency, value resolution uses $\mathcal{C}_t$ purely as a value source for parameter resolution, ensuring the two components are complementary.

Directly dispatching from value resolution to raw tool implementations faces two persistent limitations. First, precise parameter formatting is fragile: even when value resolution correctly identifies the target tool and user intent, small deviations in argument type, naming, or value encoding can cause schema validation failures. Second, with large tool catalogues, value resolution must reason over raw API schemas—a verbose, low-level representation that degrades tool selection accuracy and increases the likelihood of incomplete or malformed bindings. These limitations motivate the Tool Primitive abstraction, which absorbs schema resolution and argument binding within each Primitive. The value resolution thus dispatches x_k to the corresponding Tool Primitive $\mathcal{P}_k$, which internally validates and maps the described arguments against schema s_k, and executes the underlying function f_k.

B.3 Result Validation

The validation evaluates each execution result r_k returned by a Tool Primitive against four structured criteria before the result is accepted or escalated for revision. Detailed criterion definitions are provided in Appendix C.

The validation decision is

$$v_k = \textsc{Validate}(r_k,\, \pi_k,\, s_k) \in \{\texttt{pass},\, \texttt{fail}\}. \tag{5}$$

If $v_k = \texttt{pass}$, execution proceeds to the next step π_{k+1} or, if $k = K$, the final result is returned to the user. If $v_k = \texttt{fail}$, validation generates structured feedback ϕ_k that diagnoses which criterion failed and why, and returns ϕ_k to decomposition to enrich the context and trigger targeted revision:

$$\mathcal{C}_{t+1} = \mathcal{C}_t \cup \{\phi_k\}, \quad \Pi' = \textsc{Decompose}(q,\, \mathcal{C}_{t+1}). \tag{6}$$

Rather than reverting to a brittle fallback upon tool failure, HEART diagnoses the failure mode and re-plans with enriched context, enabling structured error recovery across the full invocation lifecycle.

C Validation Criteria

The validation evaluates each execution result r_k along four structured criteria:

- **Task Completion**: Does r_k satisfy the objective specified in invocation step π_k?
- **Argument Consistency**: Are the arguments resolved by value resolution consistent with the user's original intent and the constraints encoded in schema s_k?

- **Execution Validity**: Did the tool execute without runtime errors, and does the return value conform to the expected output schema?
- **Constraint Satisfaction**: Are any task-level or domain-specific constraints—such as rate limits, access permissions, or business rules—respected by the execution outcome?

Each criterion is assessed independently; failure in any single criterion yields $v_k = \texttt{fail}$ and triggers feedback generation. This decomposition enables validation to produce actionable, criterion-specific diagnostics rather than opaque pass/fail signals.

D End-to-End Execution Flow

Algorithm 1 summarizes the full HEART pipeline. Starting from user query q, decomposition iteratively acquires missing information until context is sufficient, then produces invocation plan Π. The value resolution sequentially binds and dispatches each step, passing inter-step results through Tool Primitive interfaces. The validation evaluates each result and either advances execution or returns structured feedback for revision. This loop continues until all K steps pass validation or a maximum revision budget is exhausted.

Algorithm 1 Pseudo-code of HEART

Require: User query q, ToolFace $\mathcal{T}$, max revise budget B
Ensure: Final execution result or structured failure report

$\mathcal{C} \leftarrow \{q\}, b \leftarrow 0$
while DECOMPOSE$(q, \mathcal{C}) = \texttt{insufficient}$ **do**
  $c \leftarrow$ DECOMPOSE.CLARIFY$(q, \mathcal{C})$
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{\text{USER}(c)\}$
end while
$\Pi \leftarrow$ DECOMPOSE.PLAN$(q, \mathcal{C})$
while $b \leq B$ **do**
  for each step $\pi_k \in \Pi$ **do**
    $\beta_k \leftarrow$ RESOLVE$(\pi_k, \mathcal{C})$
    $r_k \leftarrow \mathcal{P}_k(\beta_k \mid r_{k-1})$
    if VALIDATE$(r_k, \pi_k, s_k) = \texttt{fail}$ **then**
      $\phi_k \leftarrow$ VALIDATE.FEEDBACK(r_k, π_k, s_k)
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{\phi_k\}$
      $\Pi \leftarrow$ DECOMPOSE.PLAN$(q, \mathcal{C})$
      $b \leftarrow b + 1$; **break**
    end if
  end for
  return r_K
end while
return FAILUREREPORT$(\mathcal{C})$

E Prompt Templates

E.1 Decomposition Prompt

decomposition System Prompt

Role. You are decomposition in the tool-calling framework. Your responsibilities are (1) intent analysis and (2) information sufficiency checking.
Intent Analysis. Given the user query, identify:
- The target task and its objective.
- The relevant tool categories required to fulfill the task.
- Any explicit constraints or preferences expressed by the user (e.g., priority, domain, format).

Information Sufficiency Checking. Determine whether the current context provides all information needed to construct a complete invocation plan. Output one of:

- `SUFFICIENT` — proceed to generate an invocation plan.
- `INSUFFICIENT` — generate a targeted clarification question asking only for the missing information.

Invocation Plan Format. When context is sufficient, output an ordered JSON list of invocation steps:

```
{
  "status": "SUFFICIENT",
  "plan": [
    {
      "step": 1,
      "tool_category": "<category>",
      "tool_hint": "<tool name or description>",
      "objective": "<what this step should accomplish>",
      "dependencies": []
    },
    {
      "step": 2,
      ...
      "dependencies": [1]
    }
  ]
}
```

Clarification Format. When context is insufficient:

```
{
  "status": "INSUFFICIENT",
  "clarification": "<single targeted question>"
}
```

Revision. When provided with validation feedback, revise the plan to address the diagnosed failure. Output a revised plan in the same format, annotated with ``replanned'': true.

E.2 Value Resolution Prompt

value resolution System Prompt

Role. You are value resolution in the tool-calling framework. Your responsibilities are (1) parameter mapping and (2) tool dispatch.

Parameter Mapping. For each step in the invocation plan, given the current context (the user query, prior clarifications, and results of previously executed steps):

- Resolve the value of every required parameter of the step from the context. Values may originate from the original user query, earlier clarification turns, or the structured result of an upstream step (referenced as `$step_j.<field>`).
- Group all resolved values that belong to the same step together; do not mix values across steps.
- Encode the resolved bindings into a single natural-language invocation request that names the target Tool Primitive, states the intended operation, and embeds the resolved values inline. Do not emit a raw schema-compliant parameter dictionary — the Tool Primitive performs schema-level argument binding internally.
- Never invent a value that is not grounded in the context, and never override the target Tool Primitive chosen by decomposition.

Tool Dispatch. Attach execution-level hyperparameters for each step:

- `retry`: integer in $[1, 5]$. Use higher values for safety- or finance-critical actions and lower values for read-only lookups.
- `priority`: one of $\{$`low`, `normal`, `high`$\}$. Use `high` when the user signals urgency or when the step lies on the critical path of a time-sensitive task.

- `timeout_s`: integer seconds; default 30.

Dispatch Format. Output an ordered JSON list, one entry per step in the invocation plan, in the original step order:

```
{
  "status": "READY",
  "dispatch": [
    {
      "step": 1,
      "target_tool": "<tool name or tool hint>",
      "invocation_request": "<natural-language request>",
      "resolved_bindings": { "<param>": "<value>", ... },
      "h_params": {
        "retry": <int>,
        "priority": "<low|normal|high>",
        "timeout_s": <int>
      }
    },
    {
      "step": 2,
      ...
    }
  ]
}
```

Cross-Step Consistency. When two or more steps share the same parameter (e.g., the same card identifier or user ID), resolve the value once from the context and reuse it consistently across all affected steps' bindings.

E.3 Tool Primitive Prompt

Tool Primitive System Prompt

Role. You are a Tool Primitive in the tool-calling framework. You wrap a single tool from ToolFace and serve as its agent-native interface. You are instantiated with two pieces of information: the tool's **schema** (specifying parameter names, types, constraints, and return format) and the tool's executable **function**.

Schema Resolution. Given the natural-language invocation request from value resolution and any optional context (e.g., the structured result of a prior Tool Primitive passed as upstream context):

- Interpret the request to extract the intended operation and the values described for each parameter.
- Map each described value to the argument space defined by your schema, performing type coercion (e.g., string-to-integer, date normalization), enum matching, and unit conversion as needed.
- Apply schema-level constraints: required-field checks, value-range validation, format validation (e.g., regex patterns), and any default values specified by the schema.
- If the request references upstream context, extract the relevant field from that context and bind it to the corresponding parameter.

Function Execution. Once all arguments are resolved and validated, invoke the underlying function with the bound argument dictionary. Capture the raw return value, runtime errors (if any), and execution metadata (e.g., latency, status code).

Success Format. If schema resolution succeeds and the function returns without runtime error, output:

```
{
  "status": "SUCCESS",
  "tool": "<tool name from schema>",
```

```
  "bound_arguments": { "<param>": "<resolved value>", ... },
  "result": { ... },
  "summary": "<one-sentence natural-language summary of
  the result>"
}
```

The `summary` field provides a natural-language rendering of the result so that it can be passed as upstream context to a downstream Tool Primitive without requiring value resolution or decomposition to manage intermediate state.

Failure Format. If schema resolution fails (e.g., a required parameter cannot be extracted from the request, a value violates a schema constraint, or type coercion is impossible), or if the function raises a runtime error, do not retry silently. Surface a structured failure to validation:

```
{
  "status": "FAILURE",
  "tool": "<tool name from schema>",
  "failure_type": "<SCHEMA_RESOLUTION
                   | CONSTRAINT_VIOLATION
                   | RUNTIME_ERROR>",
  "details": "<which parameter or constraint failed, and why>",
  "bound_arguments": { "<param>": "<resolved value>", ... }
}
```

Execution Isolation. Errors must remain local to this Tool Primitive — never call another tool, never modify the invocation plan, and never attempt to recover from a failure on your own. The validation will diagnose the failure and trigger revision if needed.

E.4 Result Validation Prompt

validation System Prompt

Role. Role. You are validation in the tool-calling framework. Your responsibilities are (1) evaluation of the execution result against four criteria and (2) feedback generation for decomposition when evaluation fails. You receive the execution result, the corresponding step in the invocation plan, and the tool's schema.

Evaluation Criteria. Assess the execution result against four criteria:

1. **Task Completion**: Does the result satisfy the objective specified in the step? Is the output semantically aligned with what the user requested?
2. **Argument Consistency**: Are the resolved arguments consistent with the user's original intent and the constraints encoded in the tool's schema? Were any required fields omitted or incorrectly substituted?
3. **Execution Validity**: Did the tool execute without runtime errors? Does the return value conform to the expected output schema?
4. **Constraint Satisfaction**: Are task-level or domain-specific constraints respected (e.g., rate limits, access permissions, business rules)?

Pass Format. Issue `PASS` only when all four criteria are satisfied:

```
{
  "status": "PASS",
  "step": <int>,
  "criteria": {
    "task_completion":         "pass",
    "argument_consistency":    "pass",
    "execution_validity":      "pass",
    "constraint_satisfaction": "pass"
  }
}
```

Fail Format. If any criterion fails, return structured feedback to decomposition:

```
{
  "status": "FAIL",
  "step": <int>,
  "criteria": {
    "task_completion":          "pass" | "fail",
    "argument_consistency":     "pass" | "fail",
    "execution_validity":       "pass" | "fail",
    "constraint_satisfaction": "pass" | "fail"
  },
  "feedback": "<structured diagnosis identifying which
               criterion failed, why it failed, and what
               information decomposition needs to revise
               successfully>"
}
```

Feedback Guidelines.

- Feedback must be actionable: specify the failure mode precisely (e.g., "wrong argument type for `card_id`: expected `int`, received `str`") rather than vague descriptions.
- Do not speculate about failures not evidenced in the execution result or the schema.
- When multiple criteria fail simultaneously, list all failed criteria and prioritize the root cause in the diagnosis.

F Additional Experiments

F.1 Model Pricing

Table 10 lists the input/output token prices for all models used in our cost analysis.

Table 10: Token pricing for models used in τ^2-Bench cost comparison (USD per 1M tokens).

Model	Input ($/1M)	Output ($/1M)	Source
Qwen3-8B	0.18	0.70	https://artificialanalysis.ai/models/qwen3-8b-instruct/
GPT-5.4	2.50	15.00	https://developers.openai.com/api/docs/pricing
GPT-4.1	2.00	8.00	https://developers.openai.com/api/docs/pricing
GPT-4.1-mini	0.40	1.60	https://developers.openai.com/api/docs/pricing
GPT-o4-mini	0.15	0.60	https://developers.openai.com/api/docs/pricing
Claude-4.6-Sonnet	3.00	15.00	https://claude.com/pricing#api
Gemini-3.1-Pro	2.00	12.00	https://ai.google.dev/gemini-api/docs/pricing#gemini-3.1-pro-preview
Llama-3.1-8B	0.02	0.05	https://novita.ai/models/model-detail/meta-llama-llama-3.1-8b-instruct

F.2 Additional ACEBench Comparisons

We additionally compare HEART against open-source models and SFT-based tool-calling baselines on ACEBench in Table 11. HEART's Overall score of 86.9% surpasses the best open-source baseline Qwen2.5-Coder-32B-Instruct (79.6%) by $+7.3\%$. The gap is most pronounced on Multi-Turn (83.5% vs. 71.0%, $+12.5\%$) and Agent (72.3% vs. 60.8%, $+11.5\%$). SFT-based tool-calling models struggle severely on Special and Agent categories: Watt-Tool-8B reaches only 6.0% on Special and 2.8% on Agent, and xLAM-7B-r scores 0.0 on Preference, confirming that fine-tuning on narrow tool-use trajectories does not generalize to robustness-critical or agentic scenarios.

F.3 Ablation Studies on Backbone LLM Scale

Since HEART uses Qwen3-8B for all four roles (decomposition, value resolution, validation, and Tool Primitives), we examine performance sensitivity to backbone capacity by evaluating four configurations on τ^2-Bench: HEART (LLaMA-3.1-8B), which replaces Qwen3-8B with LLaMA-3.1-8B across all roles; HEART (Qwen3-8B), the default configuration; HEART (GPT-5.4), which uses GPT-5.4 across all roles; and HEART (GPT-5.4 + Qwen3-8B), which assigns GPT-5.4 to decomposition, value resolution, and validation while retaining Qwen3-8B for Tool Primitives.

Table 11: Performance comparison (%) of different methods across domains on the ACEBench.

Model	Normal						Special	Agent	Overall
	Atom	Single-Turn	Multi-Turn	Similar API	Preference	Summary			
Qwen2.5-Coder-32B-Instruct	90.2	81.0	71.0	83.0	81.0	84.1	80.7	60.8	79.6
DeepSeek-V3	91.5	84.0	77.0	83.0	83.0	86.5	73.0	34.5	74.8
Qwen2.5-72B-Instruct	86.8	80.3	69.5	83.0	81.0	82.1	75.7	45.0	74.7
Llama-3.1-70B-Instruct	82.5	68.3	63.5	79.0	68.0	75.5	38.3	42.3	60.4
Qwen2.5-7B-Instruct	76.0	60.3	58.5	72.0	67.0	69.4	47.0	13.8	54.8
DeepSeek-Coder-V2-Lite-Instruct	75.2	57.8	46.5	72.0	65.0	66.4	40.3	2.0	49.5
Qwen2.5-Coder-7B-Instruct	76.0	63.8	57.5	74.0	68.0	70.1	22.3	15.5	48.9
Watt-Tool-8B	85.7	69.3	55.5	79.0	64.0	75.6	6.0	2.8	45.7
Hammer2.1-7B	73.7	57.5	40.0	62.0	55.0	62.8	14.7	16.8	42.9
Llama-3.1-8B-Instruct	51.9	39.8	28.0	66.0	46.0	46.6	21.0	5.3	33.4
Phi-3-Mini-128k-Instruct	57.2	39.3	23.0	58.0	32.0	46.5	18.7	0.8	32.0
xLAM-7B-r	43.5	22.0	19.0	61.0	0.0	33.7	2.7	8.8	21.6
Llama-3.2-3B-Instruct	38.7	15.3	9.0	42.0	32.0	29.6	9.4	0.0	19.6
Hammer2.1-3B	22.4	11.5	3.5	40.0	20.0	18.7	1.0	1.5	11.3
HEART	**94.2**	**87.5**	**83.5**	**89.0**	**86.0**	**89.8**	**95.0**	**72.3**	**86.9**

Table 12: Ablation study on backbone LLM scale evaluated on τ^2-Bench across three domains.

Variant	Domain	Pass[1]	Pass[2]	Pass[3]	Pass[4]	Tokens / Cost ($)	Latency (s)
HEART (LLaMA-3.1-8B)	Retail	0.58	0.54	0.52	0.51	23,247 / 0.00116	19.52
	Airline	0.53	0.48	0.45	0.43	25,812 / 0.00129	21.14
	Telecom	0.46	0.41	0.38	0.36	22,439 / 0.00112	26.43
HEART (GPT-5.4)	Retail	0.86	0.82	0.80	0.79	20,456 / 0.30684	26.34
	Airline	0.78	0.74	0.72	0.71	22,891 / 0.34337	29.58
	Telecom	0.68	0.64	0.62	0.61	19,573 / 0.29359	38.78
HEART (GPT-5.4 + Qwen3-8B)	Retail	0.84	0.79	0.76	0.74	18,362 / 0.19666	23.46
	Airline	0.75	0.70	0.67	0.65	19,147 / 0.20506	26.18
	Telecom	0.67	0.63	0.61	0.60	17,824 / 0.19089	42.84
HEART (Qwen3-8B)	Retail	0.82	0.78	0.76	0.75	21,684 / 0.0152	18.72
	Airline	0.73	0.68	0.65	0.63	24,531 / 0.0172	21.34
	Telecom	0.63	0.57	0.53	0.50	20,887 / 0.0146	37.95

Table 12 reports results. HEART (GPT-5.4) achieves the highest Pass^k scores across all domains, confirming that stronger backbone capacity benefits planning and routing. However, HEART (Qwen3-8B) achieves competitive performance at a fraction of the cost: it matches HEART (GPT-5.4 + Qwen3-8B) on most metrics while reducing cost by over $12\times$ relative to HEART (GPT-5.4). The HEART (GPT-5.4 + Qwen3-8B) configuration further demonstrates that Tool Primitives are modular and composable: by retaining Qwen3-8B solely for Tool Primitives while delegating planning, routing, and validation to GPT-5.4, the hybrid variant achieves performance close to the full GPT-5.4 configuration, confirming that Tool Primitives can be integrated with arbitrary backbone LLMs without architectural modification. This plug-and-play compatibility suggests that practitioners can independently scale the orchestration layer and the execution layer according to their cost and performance requirements. HEART (LLaMA-3.1-8B) lags behind HEART (Qwen3-8B) by a substantial margin across all domains and all k, indicating that model quality rather than parameter count is the primary driver of performance within the same size class. Taken together, these results confirm that Qwen3-8B strikes the best balance between task completion and inference cost, justifying its selection as the default backbone.

F.4 Retrieval Quality and Repository Scalability

ToolFace serves tools by retrieval rather than enumeration, so retriever accuracy bounds end-task performance. We measure Recall@k of the semantic retriever described in Sec. 4.1 over the full 25,519-tool repository, taking the gold tools of each task as ground truth.

Recall@5 reaches 92.4 on ToolBench and 94.7 on NESTFUL (Table 13). Retrieval misses account for 15–20% of HEART's residual end-task failures; validation-triggered revision loop recovers a portion of first-pass misses by reformulating the tool query.

Table 13: Recall@k (%) of ToolFace semantic retrieval over the full 25,519-tool repository.

Benchmark	Recall@1	Recall@3	Recall@5	Recall@10
ToolBench	76.5	88.2	92.4	95.8
NESTFUL	81.3	91.6	94.7	97.2

Because tools are retrieved rather than enumerated, context pressure is governed by the number of retrieved candidates per step rather than by repository size, so repository growth affects only retrieval difficulty. To measure this directly, we subsample ToolFace and re-evaluate retrieval on the same queries.

Table 14: ToolBench Recall@5 (%) as the ToolFace repository is subsampled.

Repository size	5k	10k	25,519
ToolBench Recall@5	94.6	93.7	92.4

Recall@5 declines by roughly 2 points as the repository grows $5\times$ (Table 14), indicating that the only component whose difficulty scales with repository size remains stable at this scale. For reference, the documented degradation regime for in-context tool catalogues begins at roughly 8K–120K tokens of schemas, i.e., tens to hundreds of tools in context; $25{,}519$ tools is two orders of magnitude beyond what fits in any context window.

F.5 Validation Reliability

HEART's recovery loop is conditioned on validation, which uses the same Qwen3-8B backbone as the other roles. We therefore measure its decision quality directly: we sample $N = 200$ validation decisions from ToolBench and NESTFUL trajectories and obtain ground-truth pass/fail labels by human annotation.

Table 15: validation decision quality against human-annotated pass/fail labels ($N = 200$ sampled decisions on ToolBench and NESTFUL).

Metric	Precision	Recall	False-pass rate	False-fail rate
Value	0.97	0.96	8%	4%

Table 15 reports the results. Among false passes, the majority occur on the task-completion criterion for partially satisfied objectives; fewer than 3% of all trajectories are terminated by a false pass. False fails trigger at least one redundant revision round in 10–15% of trajectories, which costs additional tokens without changing outcomes. This is consistent with the saturation observed at $B = 3$ in Table 7: frequent false fails would make a larger budget trigger further revision and perturb results, whereas the observed plateau indicates that the loop terminates because validation converges.

F.6 Matched-Backbone Comparisons

HEART pairs a harness with a small backbone, so its gains could in principle come from the scaffold, the backbone, or the tool catalogue. To isolate the harness, we hold the backbone fixed at Qwen3-8B and the catalogue fixed at ToolFace, and compare against four scaffolds: ReAct, DFSDT, retrieval-only tool selection, and a standard planner–executor pipeline.

HEART leads the strongest matched-backbone variant (DFSDT) by $+19.3$ Pass on ToolBench and $+0.31$ Full Acc. on NESTFUL (Table 16). Two results hold the comparison in the other direction, with the backbone fixed at a frontier model: HEART (GPT-5.4) improves over bare GPT-5.4 by $+16/+14/+12$ Pass[1] on Retail/Airline/Telecom (Table 12 vs. Table 3), and on BFCLv4 the same Qwen3-8B backbone moves from 12.0 to 86.0 Web Search Overall once placed inside HEART (Table 5). Regarding inference budget, HEART's per-task token count is roughly $3\times$ that of single-

Table 16: Matched-backbone comparison on ToolBench and NESTFUL. All variants use the same Qwen3-8B backbone and the same tool catalogue.

Method	ToolBench		NESTFUL	
	Avg. Pass	Win Rate	Full Acc.	Win Rate
Qwen3-8B (ReAct)	33.5	46.2	0.08	0.10
Qwen3-8B (retrieval-only selection)	41.0	49.5	0.10	0.12
Qwen3-8B (planner–executor)	52.6	55.8	0.19	0.28
Qwen3-8B (DFSDT)	55.8	57.4	0.13	0.16
HEART (Qwen3-8B)	**75.1**	**75.7**	**0.44**	**0.75**

model baselines (Table 3), yet its API cost remains up to $7.4\times$ lower, and performance saturates at $B = 3$ (Table 7), so the gains are not explained by spending more calls.

F.7 Component Ablation Variants

We specify the four ablation variants reported in Table 6. **w/o Decomposition** removes intent analysis and information sufficiency checking, passing the raw user query directly to value resolution without structured decomposition or clarification. **w/o Resolution** removes parameter mapping and tool dispatch, having decomposition invoke Tool Primitives directly from the invocation plan without intermediate argument resolution. **w/o ToolFace & Tool Primitives** removes the centralized schema–function registry and the natural language invocation abstraction, exposing raw API schemas directly to value resolution, which resolves arguments and dispatches calls itself. **w/o Validation** removes the validation loop, so execution results are returned to the user without evaluation or feedback-driven revision.

F.8 Budget-Matched Ablation of Tool Primitives

Removing Tool Primitives also removes one LLM call per invocation step, so the drop reported in Table 6 conflates the natural language interface with the additional reasoning capacity that carries it. We therefore construct a budget-matched variant: Tool Primitives are removed, and their call budget is reallocated to value resolution, which is allowed the same number of additional LLM calls per step to self-refine its schema-compliant arguments. Backbone, total call count, and token budget match full HEART.

Table 17: Budget-matched ablation of Tool Primitives on ToolBench and NESTFUL.

Variant	ToolBench Avg. Pass	NESTFUL Full Acc.
Equal-budget value resolution self-refine (no Tool Primitives)	34.2	0.15
w/o Tool Primitives, no budget reallocation (Table 6)	16.1	0.06
HEART (full)	**75.1**	**0.44**

Reallocating the same budget to schema-based self-refinement recovers only part of the gap ($16.1 \rightarrow 34.2$ ToolBench Pass), leaving full HEART far ahead at 75.1 (Table 17). The majority of the gain is therefore attributable to the natural language interface rather than to additional reasoning capacity. A second observation points the same way: under schema-based invocation, DeepSeek-V3 (685B, roughly $85\times$ our backbone) reaches 0.09 Full Acc. on NESTFUL against 0.44 for HEART at 8B (Table 2).

F.9 Real-World Tasks under Matched Tool Access

The 50-task comparison in Table 8 evaluates each commercial model through its product interface with its own built-in tool ecosystem (web browsing, search, code execution), and scores end-to-end outcomes, i.e., whether the system actually completes the booking or reservation on the user's

behalf. That protocol compares complete systems as deployed, but leaves tool access unmatched. We therefore repeat the evaluation with the frontier models called through their APIs and the ToolFace tools exposed via native function calling on the same 50 tasks.

Table 18: Task Completion Rate (%) on the 50 real-world tasks when the frontier models are given the ToolFace tools through native function calling.

Domain	**GPT-5.4**	**Claude-4.6-Sonnet**	**Gemini-3.1-Pro**	**HEART**
Travel Planning	30	30	20	**80**
Healthcare	20	30	20	**70**
E-commerce	20	40	20	**80**
Finance	40	40	30	**100**
Local Services	20	20	20	**90**
Overall	26	32	22	**84**

Equalizing tool access raises overall completion from 20–24% to 22–32%, at most +8 points (Claude-4.6-Sonnet). The gains concentrate in booking-heavy domains where APIs help most: all three models gain +10 on Travel Planning, and Claude-4.6-Sonnet gains +30 on E-commerce. Tool availability therefore explains only a small fraction of the remaining 52–62 point gap to HEART (84%). This is consistent with our failure analysis: the baselines retrieve options reliably but fail at multi-step grounding, state tracking, and recovery once the task requires acting on behalf of the user.

G LIMITATION

While HEART demonstrates strong performance across diverse benchmarks, we acknowledge several limitations. First, the multi-stage pipeline introduces higher token consumption than single-model baselines, though the use of a lightweight 8B backbone keeps overall API cost low; practitioners with stricter latency budgets may need to consolidate roles. Second, ToolFace currently relies on manually authored schemas to ensure interface quality, and extending the registry to new domains requires additional curation effort; automating schema generation is a natural direction for future work. Third, our robustness evaluation focuses on prompt injection at the tool-selection stage following ToolHijacker; broader threat models, such as compromised tool outputs, remain to be studied. We view these as opportunities for further refinement rather than fundamental obstacles to the framework.

H CODE AND RESULT

We publish ToolFace, and the comprehensive results of Tool Primitives on the web. For detailed information, please visit the following link: `https://anonymous.4open.science/r/32D8`.