Harness Engineering in LLM Tool Use via Agent-Native Reusable Tool Primitives

Anonymous Author(s)
Affiliation
Address
email

Abstract

Large language models (LLMs) augmented with external tools have demonstrated remarkable capability in solving complex real-world tasks. However, existing approaches suffer from two key challenges: brittle multi-step and multi-turn reasoning caused by incompatible tool output types and API schemas, and performance degradation under large tool catalogues. To address these, we introduce **Tool Primitives**, a design that replaces rigid API schema-based invocation with natural language as the interface for tool calling, where each tool is wrapped with an LLM interface that handles schema resolution and execution internally, enabling natural inter-tool communication for nested and multi-turn tool calling. Building on Tool Primitives, we host **ToolFace**, a centralized repository of 25,519 functions from which LLMs dynamically retrieve only the relevant tools at inference time, eliminating the need to enumerate raw API schemas in context. To orchestrate Tool Primitives and ToolFace reliably in complex settings, we further propose **HEART**, a **H**arness **E**ngineering framework via **A**gent-native, **R**eusable **T**ool Primitives, comprising a Planner, Router, and Verifier that jointly support dynamic tool invocation planning, multi-step execution, and feedback-driven recovery.

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 [1, 2]. 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 [3, 4], and fine-tuning-based methods that train models on curated tool-use trajectories via supervised fine-tuning (SFT) and preference optimization [5, 6, 7]. The latter have driven substantial improvements in structured function calling, with models increasingly evaluated on standardized benchmarks such as BFCLv4 [8].

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 [9], even the strongest models achieve only 28% full-sequence match accuracy, while on multi-turn benchmarks

Submitted to 40th Conference on Neural Information Processing Systems (NeurIPS 2026). Do not distribute.

such as τ^2-Bench [10], 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 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 [11, 12], and this problem is further amplified in real-world settings such as ToolBench [2], 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 [13] and Agent Primitives [14], 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. When calling tools, LLMs dynamically search and retrieve only the relevant tools from ToolFace at inference time, directly avoiding the performance degradation caused by large tool catalogues.

To make Tool Primitives and ToolFace practical for real-world deployment, we introduce **HEART**, a **H**arness **E**ngineering via **A**gent-native, **R**eusable **T**ool Primitive. We design a multi-agent system (MAS) comprising a **Planner** that decomposes user queries into an invocation plan (or requests clarification when context is insufficient) and a **Router** that performs parameter mapping to bind function arguments and hyperparameters according to user queries before dispatching calls to the corresponding Tool Primitives. Tool Primitives further execute the tools and return results. We also employ a **Verifier** that evaluates each execution result against four criteria: task completion, argument consistency, execution validity, and constraint satisfaction. When verification fails, feedback is returned to the Planner for re-planning, enabling iterative recovery instead of one-shot execution.

Extensive experiments on five benchmarks spanning large-scale real-world API invocation across single- and multi-tool scenarios (ToolBench [2]), nested API call sequences (NESTFUL [9]), dual-control conversational agent evaluation (τ^2-Bench [10]), robustness and fine-grained tool-use evaluation (ACEBench [15]) and agentic web search and memory function calling (BFCLv4 [8]), 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 for LLM tool use realized through a multi-agent system comprising a five-stage pipeline—Planning, Routing, Tools, Execution, and Verification—to jointly address key limitations of existing LLM tool-use approaches.
- 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 commercial models (22%).

2 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 [5] 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 [16] 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 [6] and the xLAM series [17] 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: they lack explicit, structured mechanisms for planning, routing, and verifying complex tool execution paths, leaving robust multi-step composition an open challenge.

Benchmarks for Tool Use Evaluation. Evaluating LLM tool use spans several dimensions. BFCLv4 [8] evaluates structured function calling across diverse APIs, while StableToolBench [18] improves ToolBench [2] by replacing unstable live APIs with deterministic virtual servers. Nexus-Raven [19] focuses on schema-compliant JSON generation, API-Bank [20] studies tool-augmented dialogue scenarios, and API-BLEND [21] evaluates compositional planning across heterogeneous APIs. NESTFUL [9] 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 [22] and τ^2-Bench [10] evaluate agents under rule-constrained conversational environments, while HammerBench [23], ACEBench [15], and ToolSandbox [24] 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, HEART formulates tool use as a multi-agent process with planning, routing, execution, and verification stages, enabling iterative reasoning, multi-turn interaction, and recovery from failures.

3 Methodology

3.1 Overview

The core of HEART lies in two key innovations: **ToolFace**, a centralized tool repository that decouples tool storage from the model's input context, and **Tool Primitives**, LLM-wrapped interfaces that enable natural language invocation and inter-tool communication without requiring knowledge of raw API schemas. To make these two components work reliably in complex settings, we further introduce other supporting functions. An overview of HEART is shown in Fig. 1.

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.

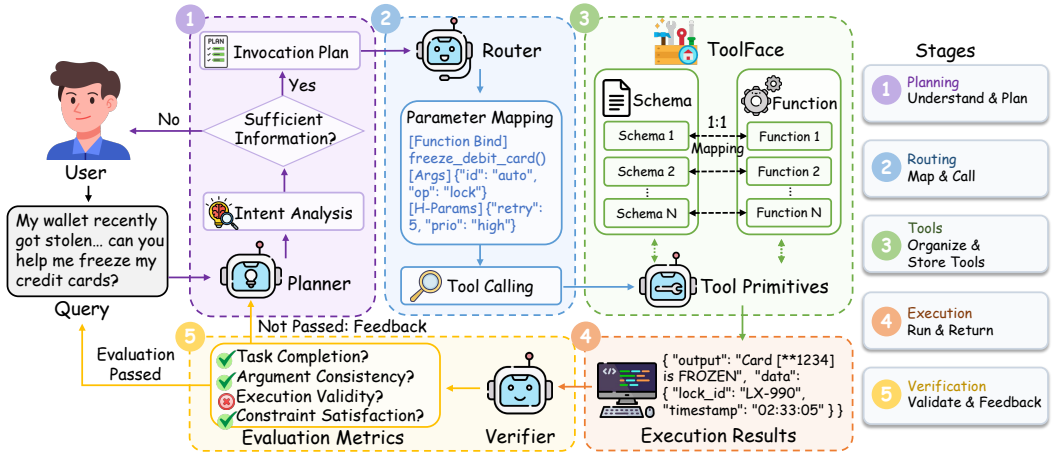

Figure 1: Overview of HEART. Planner performs intent analysis and checks for information sufficiency, soliciting missing details from the user if needed. Once sufficient context is gathered, the Router maps parameters and dispatches invocations to the corresponding Tool Primitives stored in ToolFace. Each Tool Primitive executes the underlying function and returns results to the Verifier, which evaluates task completion, argument consistency, execution validity, and constraint satisfaction. If verification fails, structured feedback is returned to the Planner for re-planning.

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 [2]; 40 mathematical reasoning tools and 4,348 coding tools from NESTFUL [9]; 68 tools across retail, airline, and telecom domains from τ^2-Bench [10]; 4,538 bilingual APIs spanning 8 major domains and 68 sub-categories (including technology, finance, entertainment, and healthcare) from ACEBench [15]; 2 web search and 5 memory tools from BFCLv4 [8]; 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

To make ToolFace and Tool Primitives work reliably in complex settings, we adopt harness engineering, realized through a multi-agent system with collaborative agents covering the full invocation lifecycle: a **Planner** for intent analysis and information sufficiency checking, a **Router** for parameter mapping and tool dispatch, and a **Verifier** for execution evaluation and feedback-driven recovery. We summarize their formal interfaces below; complete role specifications are provided in Appendix A, and prompts for each agent are listed in Appendix D.

Planner. The Planner decomposes the user query q into a structured intent and assesses whether the current context $\mathcal{C}_t$ is sufficient to construct an invocation plan:

$$\delta_t = \textsc{Planner}(q, \mathcal{C}_t) \in \{\texttt{sufficient}, \texttt{insufficient}\}. \tag{2}$$

If $\delta_t = \texttt{insufficient}$, a targeted clarification request is returned to the user; otherwise, the Planner emits an invocation plan $\Pi = (\pi_1, \ldots, \pi_K)$, an ordered sequence of K tool invocation steps. This iterative information acquisition mechanism directly addresses the multi-turn degradation challenge by resolving ambiguity before committing to execution, reducing downstream parameter mapping failures and invalid tool calls.

Router. For each step $\pi_k \in \Pi$, the Router resolves required arguments from $\mathcal{C}_t$ and constructs a natural language invocation request together with execution-level configuration:

$$\beta_k = \textsc{Router}(\pi_k,\, \mathcal{C}_t) = \big(x_k,\, \texttt{H-Params}_k\big), \tag{3}$$

where x_k encodes the target tool, intended operation, and resolved parameter values, and $\texttt{H-Params}_k$ specifies execution hyperparameters such as retry count and priority level. The request x_k is then dispatched to the corresponding Tool Primitive $\mathcal{P}_k$, which handles internal schema validation and

argument binding. This indirection sidesteps two persistent failure modes of dispatching to raw tool implementations—fragile parameter formatting and degraded selection accuracy under large tool catalogues—directly motivating the Tool Primitive abstraction introduced earlier.

Verifier. The Verifier evaluates each execution result r_k along four criteria: **task completion**, **argument consistency**, **execution validity**, and **constraint satisfaction**, producing a verification:

$$v_k = \textsc{Verifier}(r_k,\ \pi_k,\ s_k) \in \{\texttt{pass},\ \texttt{fail}\}. \tag{4}$$

If $v_k = \texttt{pass}$, execution proceeds to π_{k+1}, or the final result is returned to the user when $k = K$. If $v_k = \texttt{fail}$, the Verifier generates structured feedback ϕ_k diagnosing the failure mode and returns it to the Planner to enrich the context and trigger targeted re-planning:

$$\mathcal{C}_{t+1} = \mathcal{C}_t \cup \{\phi_k\}, \quad \Pi' = \textsc{Planner}(q,\ \mathcal{C}_{t+1}). \tag{5}$$

This feedback-driven recovery loop directly addresses the troubleshooting challenge: rather than reverting to a brittle fallback upon tool failure, HEART systematically diagnoses the failure mode and re-plans with enriched context, enabling structured error recovery across the full invocation lifecycle. Detailed criterion definitions are provided in Appendix B. We also provide the end-to-end execution flow of HEART and the prompts for each agent in Appendix C and Appendix D, respectively.

4 Experiments

4.1 Experimental Setup

Benchmarks. We evaluate HEART on five benchmarks: ToolBench [2] covers large-scale real-world API invocation across single- and multi-tool scenarios. NESTFUL [9] evaluates nested API call sequences where outputs of one call serve as inputs to the next. τ^2-Bench [10] assesses agents under dual-control conversational settings where both agent and user invoke tools in a shared environment. ACEBench [15] targets robustness and fine-grained tool-use evaluation across complex multi-tool workflows. BFCLv4 [8] focuses on agentic web search and memory function calling.

Baselines. Beyond benchmark-specific baselines, we include frontier models including GPT-5.4 [25], Claude-4.6-Sonnet [26], and Gemini-3.1-Pro [27] as shared baselines across all benchmarks.

Implementation Details. We use Qwen3-8B as the backbone LLM for all roles in HEART: Planner, Router, Verifier, and Tool Primitives, by default. All experiments are conducted with a maximum re-planning budget of $B = 3$. Tool retrieval from ToolFace is performed via semantic search over tool schema descriptors. We report average results over three runs.

4.2 Main Results

On ToolBench. We compare HEART against ToolBench [2]'s baselines including Vicuna [28], Alpaca [29], ToolLLaMA [2], GPT-3.5-Turbo [30], Claude-2 [31], Text-Davinci-003 [32], and GPT-4 [33]. 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 Planner-Router-Verifier pipeline generalizes effectively to large-scale real-world API invocation without relying on explicit search strategies such as DFSDT.

On NESTFUL. Following NESTFUL [9], we compare HEART against a broad set of baselines, including the xLAM series [34, 17], Hammer [5], ToolACE [6], Llama-3.1 [35], Mixtral [36], Granite-20B [37], DeepSeek-V3 [38], and GPT-4o [39]. 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 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**

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 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 absence of structured planning: DeepSeek-V3 (685B) reaches only 0.09 Full Acc., far below HEART's 0.44.

On τ^2-Bench. Following the τ^2-Bench [10], we include GPT-4.1 [40], GPT-o4-mini [41], GPT-4.1-mini [40], Claude-3.7-Sonnet [42] and report Pass^k ($k = 1, 2, 3, 4$), which measures the fraction of tasks completed successfully within k interaction rounds, alongside average per task token consumption, API cost (USD) (Detailed in Appendix E.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 re-planning 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 its multi-agent architecture, HEART reduces API cost by up to $7.4\times$ relative to GPT-5.4.

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-

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	0.53	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	0.69	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	0.31	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	0.51	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	0.79	0.73	0.65	0.60	7,153 / 0.1073	14.12
	Airline	0.71	0.64	0.60	0.51	7,412 / 0.1112	16.52
	Telecom	0.58	0.49	0.37	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	0.28	7,258 / 0.0871	10.82
	Telecom	0.55	0.43	0.36	0.31	6,492 / 0.0779	28.46
HEART	Retail	**0.82**	**0.79**	**0.77**	**0.73**	21,684 / 0.0152	18.72
	Airline	**0.72**	**0.69**	**0.65**	**0.63**	24,531 / 0.0172	21.34
	Telecom	**0.62**	**0.59**	**0.55**	**0.50**	20,887 / 0.0146	37.95

source baselines including Qwen2.5 series [43], Llama-3.1 [35], DeepSeek-V3 [38], Phi-3 [44], and tool-calling models such as Hammer [5], xLAM [17], and Watt-Tool-8B [45] are in Appendix E.2.

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 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 re-planning.

On BFCLv4. Following BFCLv4 [8], we compare HEART against a diverse set of baselines, including the xLAM series [17], Hammer [5], LLaMA models [35], Qwen3 [46], GPT-4.1 [40], GPT-4.1-Mini [40], Gemini-1.5-Pro [47], and Claude-3.5-Sonnet [48]. 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 (84.0%) and Memory Overall (69.38%), surpassing Claude-Sonnet-4.6 by $+1.0\%$ 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.

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	14.62	5.16	7.10	31.61
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	82.00	59.35	62.58	63.23	61.72
HEART	**87.00**	**85.00**	**86.00**	**68.47**	**69.10**	**70.58**	**69.38**

4.3 Ablation Study

Component Contribution. We evaluate four HEART ablations on ToolBench and NESTFUL, each removing one core component: (1) w/o Planner, which removes intent analysis and information sufficiency checking, passing the raw user query directly to the Router without structured decomposition or clarification;(2) w/o Router, which removes parameter mapping and tool dispatch, instead having the Planner invoke Tool Primitives directly from the invocation plan without intermediate argument resolution; (3) w/o ToolFace & Tool Primitives, which removes the centralized schema–function registry and the natural language invocation abstraction, exposing raw API schemas directly to the Router and dispatching calls; and (4) w/o Verifier, which removes the verification loop, so execution results are returned to the user without evaluation or feedback-driven re-planning. Table 6 reports results.

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 Planner	57.2	64.1	0.26	0.51
w/o Router	62.4	67.3	0.28	0.52
w/o ToolFace	16.1	27.9	0.06	0.06
w/o Verifier	47.6	49.6	0.15	0.33
HEART	**75.1**	**75.7**	**0.44**	**0.75**

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 Verifier produces the second-largest drop (Pass Rate 47.6, Full Acc. 0.15), demonstrating that feedback-driven re-planning is essential for recovering from execution failures. The w/o Planner and w/o Router variants show moderate but consistent degradation, confirming that intent decomposition and parameter mapping both contribute necessary intermediary layers to the pipeline.

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

Table 7: Effect of re-planning 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 re-planning 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: 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**

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.

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 attacks. Following ToolHijacker [49], we consider an attacker who injects a malicious tool document into the retrieved candidate set to manipulate the agent's tool selection, and report ACC under no attack and ASR under Gradient-Free and Gradient-Based strategies on ToolBench to evaluate Robustness under such attacks.

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 validate the three design properties of Tool Primitives. **(1) Natural language invocation.** Each Primitive resolves the schema internally, freeing the Router from raw parameter formatting. Removing this drops ToolBench Pass Rate from 75.1 to 16.1 (Table 6), and yields 0.00% ASR under prompt injection (Table 9). **(2) Inter-tool communication.** Primitive results are passed as context to the next, enabling LLM-to-LLM dialogue across dependent steps. On NESTFUL (Table 2), SFT models collapse to 0.00 Full Acc. and DeepSeek-V3 reaches only 0.09, while HEART attains 0.44. **(3) Execution isolation.** Each Primitive surfaces failures as signals to the Verifier, localizing errors. HEART sustains Pass^4 on τ^2-Bench (Table 3) where baselines degrade, and most failures resolve within $B = 3$ rounds (Table 7). We also discuss the limitations of HEART 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 coordinated multi-agent design with planning, routing, and verification, 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 the LLM tool use.

References

[1] 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, pages 4363–4400, 2024.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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, 2024.

[7] 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.

[8] 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.

[9] 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, pages 33526–33535, 2025.

[10] 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.

[11] 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.

[12] 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, 2024.

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

[14] 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.

[15] 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.

[16] 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.

[17] 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, 2024.

[18] 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, pages 11143–11156, 2024.

[19] 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.

[20] 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, pages 3102–3116, 2023.

[21] 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), pages 12859–12870, 2024.

[22] 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.

[23] 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, pages 3350–3376, 2025.

[24] 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, pages 1160–1183, 2025.

[25] OpenAI. Introducing gpt-5.4. `https://openai.com/index/introducing-gpt-5-4/`, 2025.

[26] Anthropic. Introducing claude sonnet 4.6. `https://www.anthropic.com/news/claude-sonnet-4-6`, 2025.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

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

[32] 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.

[33] 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.

[34] 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, pages 302–325, 2024.

[35] 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.

[36] 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.

[37] 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, pages 1131–1139, 2024.

[38] 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, 2024.

[39] 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.

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

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

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

[43] 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.

[44] 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.

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

[46] 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.

[47] 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.

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

[49] 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.

A Detailed Agent Role Specifications

We expand on the role specifications of the three collaborative agents introduced in the main text.

A.1 Planner

The Planner serves as HEART's entry point, responsible for two sequential functions: **intent analysis** and **information sufficiency checking**. Upon receiving a user query q, the Planner 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.

If the sufficiency check returns `insufficient`, the Planner 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 the Planner 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.

A.2 Router

Given an invocation plan Π from the Planner, the Router 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, the Router 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.

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

Directly dispatching from the Router to raw tool implementations faces two persistent limitations. First, precise parameter formatting is fragile: even when the Router 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, the Router 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 Router 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.

A.3 Verifier

The Verifier evaluates each execution result r_k returned by a Tool Primitive against four structured criteria before the result is accepted or escalated for re-planning. Detailed criterion definitions are provided in Appendix B.

If verification passes, execution proceeds to the next step π_{k+1} or, if $k = K$, the final result is returned to the user. If verification fails, the Verifier generates structured feedback ϕ_k that diagnoses which criterion failed and why, and returns ϕ_k to the Planner to trigger targeted re-planning with the enriched context.

B Verifier Evaluation Criteria

The Verifier 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 the Router 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 the Verifier to produce actionable, criterion-specific diagnostics rather than opaque pass/fail signals.

C End-to-End Execution Flow

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

Algorithm 1 Pseudo-code of HEART

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

$\mathcal{C} \leftarrow \{q\}, b \leftarrow 0$
while $\textsc{Planner}(q, \mathcal{C}) = \texttt{insufficient}$ **do**
  $c \leftarrow \textsc{Planner.Clarify}(q, \mathcal{C})$
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{\textsc{User}(c)\}$
end while
$\Pi \leftarrow \textsc{Planner.Plan}(q, \mathcal{C})$
while $b \leq B$ **do**
  for each step $\pi_k \in \Pi$ **do**
    $\beta_k \leftarrow \textsc{Router}(\pi_k, \mathcal{C})$
    $r_k \leftarrow \mathcal{P}_k(\beta_k \mid r_{k-1})$
    if $\textsc{Verifier}(r_k, \pi_k, s_k) = \texttt{fail}$ **then**
      $\phi_k \leftarrow \textsc{Verifier.Feedback}(r_k, \pi_k, s_k)$
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{\phi_k\}$
      $\Pi \leftarrow \textsc{Planner.Plan}(q, \mathcal{C})$
      $b \leftarrow b + 1$; **break**
    end if
  end for
  return r_K
end while
return $\textsc{FailureReport}(\mathcal{C})$

D Prompt Templates

D.1 Planner Prompt

Planner System Prompt

Role. You are the Planner 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>"
}
```

Re-planning. When provided with Verifier feedback, revise the plan to address the diagnosed failure. Output a revised plan in the same format, annotated with `"replanned": true`.

D.2 Router Prompt

Router System Prompt

Role. You are the Router 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 the Planner.

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.

D.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 the Router 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 the Router or Planner 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 the Verifier:

```
{
  "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 Verifier will diagnose the failure and trigger re-planning if needed.

D.4 Verifier Prompt

Verifier System Prompt

Role. Role. You are the Verifier in the tool-calling framework. Your responsibilities are (1) evaluation of the execution result against four criteria and (2) feedback generation for the Planner 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 the Planner:

```
{
  "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 the Planner needs to re-plan
               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.

E Additional Experiments

E.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

E.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.

E.3 Ablation Studies on Backbone LLM Scale

Since HEART uses Qwen3-8B for all four roles (Planner, Router, Verifier, 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 the Planner, Router, and Verifier while retaining Qwen3-8B for Tool Primitives.

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-

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	0.50	23,247 / 0.00116	19.52
	Airline	0.52	0.48	0.45	0.42	25,812 / 0.00129	21.14
	Telecom	0.45	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	0.70	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	0.83	0.79	0.77	0.74	18,362 / 0.19666	23.46
	Airline	0.74	0.70	0.68	0.65	19,147 / 0.20506	26.18
	Telecom	0.63	0.61	0.66	0.63	17,824 / 0.19089	42.84
HEART (Qwen3-8B)	Retail	0.82	0.79	0.77	0.73	21,684 / 0.0152	18.72
	Airline	0.72	0.69	0.65	0.63	24,531 / 0.0172	21.34
	Telecom	0.62	0.59	0.55	0.50	20,887 / 0.0146	37.95

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 verification 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 Use of AI Assistants

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.

G Limitation

While HEART demonstrates strong performance across diverse benchmarks, we acknowledge several limitations. First, the multi-agent design 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 will 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`.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly state the contributions made in the paper.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: See Section 4.5.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: All formulas and proofs in the Sec 3 are numbered and cross-referenced.

   Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: See the implementation details in Section 4 and Appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: See the Appendix H.

   Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: See the Section 4.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We repeated experiments and reported the average result.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: See the Section 4.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We make sure to remain anonymous.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: See the Section 5.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: Our paper poses no such risks.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We use the CC-BY 4.0 license, and provide appropriate citations.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: We provide our code and our dataset in Appendix H.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: Our paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: Our paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: The LLMs are the testing object of our paper.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.