AGENTORCH: HOW CLOSE CAN GLOBAL ORCHESTRATION COME TO STEP-BY-STEP INTERACTION?

Anonymous authors
Paper under double-blind review

ABSTRACT

Enterprise workflow agents typically rely on step-by-step interaction (STEP-BY-STEP): the agent executes an action, observes its result, and invokes a large language model (LLM) again to decide the next step. Global orchestration (GLOBAL) instead generates a complete executable workflow in advance and executes it without further LLM calls, substantially reducing inference overhead, but it remains less accurate than STEP-BY-STEP. We study how close GLOBAL can come to STEP-BY-STEP and what narrows the remaining gap. We introduce AGENTORCH, a benchmark that directly compares global orchestration with step-by-step interaction on 650 enterprise SOPs and 3,220 runtime variants under matched tools, environments, and evaluation. STEP-BY-STEP remains more accurate for every evaluated model, but the gap has shrunk by about 79% from early models to the strongest recent model. We identify three routes to further narrowing this gap. Incremental workflow construction improves GLOBAL accuracy by an average of 9.98 percentage points (pp) without runtime execution feedback. Revising complete workflows with execution feedback improves all evaluated models, including 6.13 pp on held-out runtime variants. Targeted training on complete workflows improves accuracy by an average of 12.18 pp across all five trained models. Although a gap remains, global orchestration offers a promising path toward enterprise agents that deliver higher accuracy with fewer LLM calls.

1 INTRODUCTION

Enterprise workflows often span multiple software tools and services, with later operations depending on the outputs of earlier ones (Li et al., 2026; Wang et al., 2026; Lu et al., 2025; Trivedi et al., 2024; Xie et al., 2024; Boisvert et al., 2024; Drouin et al., 2024). Completing these workflows requires an LLM agent to select tools (Qin et al., 2024; Song et al., 2023; Shen et al., 2023; Schick et al., 2023) and coordinate their execution order, data dependencies, and control flow (Liu et al., 2024; Nath et al., 2025; Li et al., 2026). We refer to this capability as *workflow orchestration*.

Existing work commonly realizes workflow orchestration through step-by-step interaction (STEP-BY-STEP): the model selects an action, observes its execution result, and then determines the next action (Shi et al., 2024; Yao et al., 2023; Zhou et al., 2024; Liu et al., 2024). This interaction allows subsequent decisions to incorporate execution feedback (Liu et al., 2024; Shinn et al., 2023), but requires repeated LLM inference throughout execution. Yet dependence on intermediate results does not inherently require the model itself to make a new decision after every action. Data dependencies, branching, loops, and other control flow can instead be specified before execution, allowing intermediate results to be consumed by a preconstructed workflow (Liang et al., 2023; Singh et al., 2022; Surís et al., 2023). We refer to this paradigm as *global orchestration* (GLOBAL): the LLM constructs a complete executable workflow before observing concrete runtime inputs, after which the workflow executes without further LLM calls. As shown in Figure 1, across the three models in the cost panels, STEP-BY-STEP makes 23.2 times as many LLM calls and generates 3.64 times as many completion tokens as GLOBAL, on average. As LLMs become more capable at planning and tool execution (Kim et al., 2023; Wang et al., 2023; Zhou et al., 2023), it becomes increasingly important to understand how much orchestration can be completed before execution. Prior work has explored generating executable workflows or plans before execution (Fan et al., 2025; Qiao et al., 2025; Zhang et al., 2025), but typically evaluates them in isolation rather than against step-by-step interaction on the same workflows, tools, execution environment, and criteria. We therefore ask:

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

Question: *How close can Global Orchestration come to Step-by-Step Interaction?*

We first make an initial comparison between GLOBAL and STEP-BY-STEP across model generations, shown in Figure 1. STEP-BY-STEP achieves higher Orchestration Accuracy for every evaluated model, but the magnitude of this advantage changes substantially across generations. From the earlier baselines through the 2024 and 2025 models, both settings improve without a consistent reduction in the gap. The 2026 models show clearer convergence, with gaps of 15.12 percentage points (pp) for GPT-5.6 Sol Thinking and 8.53 pp for Claude Opus 5 Thinking, compared with 40.67 pp for GPT-3.5 Turbo, a reduction of about 79%. Thus, global orchestration has become substantially stronger and the gap narrows in the latest model generation, but STEP-BY-STEP continues to retain an accuracy advantage across all evaluated models.

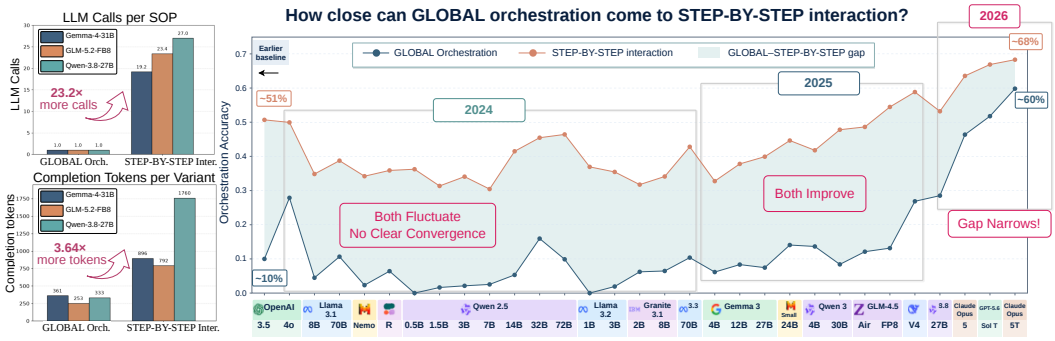


Figure 1: **Global orchestration reduces inference overhead but remains less accurate than step-by-step interaction. Left:** STEP-BY-STEP uses on average 23.2× more LLM calls per SOP and 3.64× more completion tokens per runtime variant than GLOBAL. **Right:** STEP-BY-STEP achieves higher Orchestration Accuracy across all evaluated models, while the gap narrows to approximately 8 percentage points for Claude Opus 5T.

Then to systematically study this gap, we introduce AGENTORCH, a benchmark grounded in 650 enterprise standard operating procedures (SOPs) and 3,220 runtime variants. Each SOP is associated with an executable reference workflow and multiple runtime variants, across which the same generated workflow is evaluated. Importantly, AGENTORCH evaluates GLOBAL and STEP-BY-STEP on the same SOPs, tool interfaces, execution environment, and evaluation criteria, enabling a direct and controlled comparison between the two orchestration protocols.

Using AGENTORCH, we first examine what determines how close GLOBAL can come to STEP-BY-STEP. Increasing model scale does not consistently narrow the gap, while newer model generations generally improve global orchestration and reasoning produces mixed effects. We then identify three routes to narrowing the gap, each drawing on progressively more signal. Incremental workflow construction improves GLOBAL Orchestration Accuracy by an average of 9.98 pp even without access to runtime execution feedback. We next revise the GLOBAL workflow using feedback from its own executions, without adding LLM calls at execution time. This improves Orchestration Accuracy for all eight evaluated models by an average of 6.18 pp, and by 6.13 pp on held-out runtime variants never used for feedback. Finally, we study whether stronger global orchestration can be internalized into model weights. We construct a separate SOP-grounded training set from 3,000 source recipes, with each sample pairing a task specification with a complete reference workflow. Targeted training improves Orchestration Accuracy by an average of 12.18 pp (6.84–19.88 pp) across all five trained models. Together, these results show that global orchestration can be substantially improved through incremental workflow construction, execution-feedback revision, and targeted training, while STEP-BY-STEP continues to retain an accuracy advantage across the evaluated models. Our contributions are summarized as follows:

- **Orchestration Benchmark.** We introduce AGENTORCH, an execution-grounded benchmark comparing GLOBAL and STEP-BY-STEP on 650 enterprise SOPs and 3,220 runtime variants under shared tools, environments, and evaluation criteria.
- **Test-time Orchestration Improvement.** We evaluate incremental workflow construction and execution-feedback revision as routes to stronger GLOBAL orchestration. Construction improves

several models without runtime execution feedback; revision improves all eight evaluated models, including on held-out variants.

- **Training for Global Orchestration.** We train models on 3,000 separate source recipes to generate complete workflows. Taking the better of SFT and RL for each base model, Orchestration Accuracy improves by an average of 12.18 pp across all five trained models.

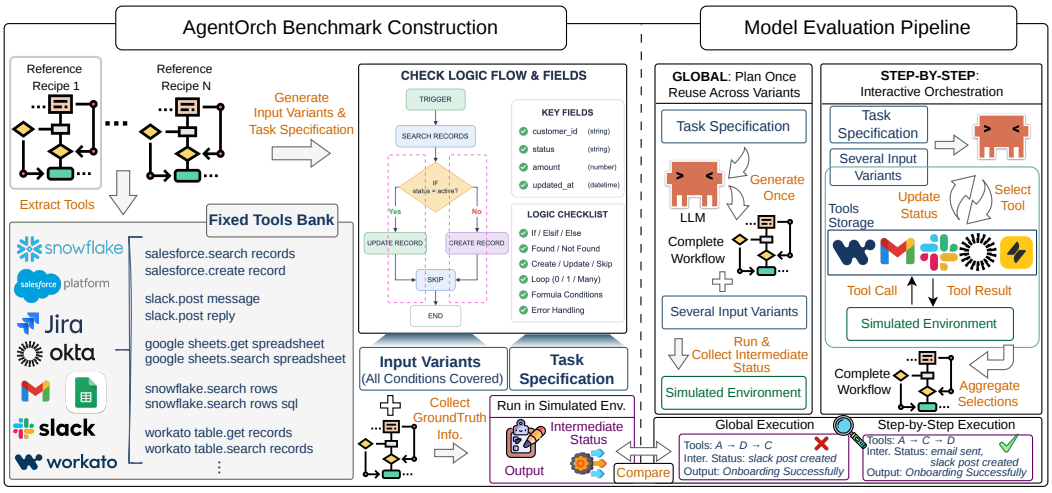


Figure 2: **Overview of the AGENTORCH benchmark construction and model evaluation pipeline. Left:** Reference recipes define the benchmark SOPs. We extract their tool operations, map them to a fixed tool bank, construct task specifications and runtime variants, and validate the resulting workflow logic and fields. Reference executions provide the behavior used for evaluation. **Right:** Under GLOBAL, the model generates one complete workflow before execution and reuses it across runtime variants. Under STEP-BY-STEP, the model selects tool calls sequentially using intermediate execution results. Both protocols are executed in the same simulated environment and evaluated against the corresponding reference behavior.

2 RELATED WORK

Interactive software-use benchmarks. A broad line of work evaluates language agents through step-by-step interaction with software environments. WebShop (Yao et al., 2022), Mind2Web (Deng et al., 2023), WebArena (Zhou et al., 2024), and VisualWebArena (Koh et al., 2024) study realistic websites, while WorkArena (Drouin et al., 2024) and WorkArena++ (Boisvert et al., 2024) target enterprise knowledge work. OSWorld (Xie et al., 2024), AndroidWorld (Rawles et al., 2025), AgentBench (Liu et al., 2024), SWE-bench (Jimenez et al., 2024), and TheAgentCompany (Xu et al., 2025) extend evaluation to desktop, mobile, coding, and simulated workplace settings. These benchmarks generally allow agents to repeatedly act on and adapt to execution feedback. Our work instead asks how close global orchestration can come to step-by-step interaction by directly evaluating both paradigms on the same enterprise workflows.

Tool-use and API-agent benchmarks. API-Bank (Li et al., 2023), ToolLLM (Qin et al., 2024), Gorilla (Patil et al., 2024), ToolAlpaca (Tang et al., 2023), RestGPT (Song et al., 2023), StableToolBench (Guo et al., 2024), TaskBench (Shen et al., 2024), UltraTool (Huang et al., 2024), τ-bench (Yao et al., 2024), and WorkBench (Styles et al., 2024) evaluate tool retrieval, action selection, argument prediction, planning, multi-step execution, and user-agent interaction. While these capabilities are central to enterprise orchestration, they are typically evaluated under a fixed execution protocol. In contrast, AGENTORCH evaluates GLOBAL and STEP-BY-STEP under matched tasks, tools, execution environments, and evaluation criteria, enabling a direct comparison of the two orchestration paradigms.

Summary. Table 1 summarizes these differences. Existing benchmarks typically evaluate either step-by-step interaction or complete workflow generation, whereas AGENTORCH supports both under the same executable setting with reference workflows, state grading, and side-effect evaluation. This enables direct comparison between GLOBAL and STEP-BY-STEP. Additional related work is discussed in Appendix C.

Table 1: Comparison of evaluation settings across agent and workflow benchmarks. **E2E Exec.** indicates whether generated actions or workflows are executed end-to-end in an environment. **STEP-BY-STEP** indicates whether the model can make new orchestration decisions using intermediate execution observations. **GLOBAL** indicates whether the model can generate a complete workflow before execution. **Reference Workflow** indicates whether executable reference workflow structures are available for evaluation. **State Grading** evaluates the resulting environment state. **Side-Effect Check** evaluates external effects produced during execution. ✓ = yes, △ = partial, and ✗ = no.

Benchmark	Domain	Interface	Execution		Global Orchestration		Outcome Evaluation	
			E2E Exec.	STEP-BY-STEP	GLOBAL	Reference Workflow	State Grading	Side-Effect Check
WebArena (Zhou et al., 2024)	Web / GUI	Browser	✓	✓	✗	✗	✓	△
WorkArena++ (Boisvert et al., 2024)	Enterprise / GUI	Browser / GUI	✓	✓	✗	✗	✓	△
OSWorld (Xie et al., 2024)	OS / GUI	Desktop GUI	✓	✓	✗	✗	△	△
AppWorld (Trivedi et al., 2024)	API / Tool	APIs	✓	✓	✗	✗	✓	✓
τ-bench (Yao et al., 2024)	API / Tool	Tools + User	✓	✓	✗	✗	✓	✓
ToolBench (Qin et al., 2024)	API / Tool	APIs	△	✓	✗	✗	△	✗
AutomationBench (Shepard & Salimans, 2026)	Workflow Agent	SaaS APIs	✓	✓	△	△	✓	✓
WorFBench (Qiao et al., 2025)	Workflow Generation	Workflow Graph	✗	✗	✓	✓	✗	✗
WorkflowBench (Fan et al., 2025)	Workflow Generation	APIs + Workflow	△	△	✓	✓	△	✗
T-Eval (Chen et al., 2024)	General Planning	Text / Tools	△	△	✓	△	✗	✗
AGENTORCH	**Enterprise Workflow**	**iPaaS Simulator**	✓	✓	✓	✓	✓	✓

3 AGENTORCH: BENCHMARKING ENTERPRISE WORKFLOW ORCHESTRATION

AGENTORCH is designed to quantify the gap between global and step-by-step orchestration and to study the factors that determine whether this gap can be narrowed. As illustrated in Figure 2, we construct the benchmark from executable enterprise workflows and evaluate orchestration methods under matched task specifications, tool interfaces, runtime variants, execution environments, and evaluation criteria. Each workflow is evaluated under multiple runtime conditions, allowing us to compare how the same task is handled when orchestration is completed before execution versus through step-by-step interaction.

Benchmark Overview. AGENTORCH contains 650 executable standard operating procedures (SOPs) and 3,220 runtime variants. Each source recipe defines one SOP and serves as its reference workflow. For each SOP, we construct a natural-language task description (Appendix A.3) and multiple runtime variants. The task description and reference workflow remain fixed, while variants instantiate different inputs and execution conditions. This design allows the same workflow task to be evaluated under both GLOBAL and STEP-BY-STEP orchestration.

Task Specification. Each SOP has a fixed *task specification* containing the task description, public input/output schemas, execution context, workspace resources, communication templates when applicable, and 64 available tool interfaces. Requirements are presented in non-causal order, requiring the model to infer tool calls, dependencies, control flow, and argument bindings. The specification excludes the reference workflow, reference traces, hidden environment state, fault settings, and concrete runtime inputs.

3.1 EXECUTION ENVIRONMENT AND RUNTIME VARIANTS

Environment Construction. We map recipe operations to a fixed catalog of 64 tool interfaces spanning services such as Slack, Jira, Salesforce, Google Sheets, Google Drive, Snowflake, and Workato. We implement real and simulated backends behind the same interfaces: real connectors are used to collect reference behavior, while model-generated workflows are evaluated in simulation. The simulated backend reproduces the benchmark-relevant connector behavior while enabling isolated and reproducible evaluation without external API rate limits, quotas, or mutable service state.

Runtime Variants. Each SOP is associated with multiple runtime variants that instantiate concrete inputs and execution conditions. These variants exercise different control-flow paths, loop cardinalities, and error conditions while preserving the same task description and reference workflow. Executions are isolated across variants. Construction details are provided in Appendix A.4.

Reference Workflow and Ground Truth. For each SOP, the source recipe serves as the reference workflow. We execute it on each runtime variant using the real connector backend and record the execution trace, intermediate tool results, and terminal output. These executions define the reference behavior used for evaluation.

3.2 ORCHESTRATION PROTOCOLS

We compare GLOBAL and STEP-BY-STEP as the two primary orchestration protocols. We additionally study LOCAL for incremental workflow construction and REPLAN for execution-feedback improvement.

GLOBAL. The model generates one complete workflow before execution and before observing concrete runtime inputs. The workflow is then fixed and reused across all variants of the SOP without further LLM interaction.

STEP-BY-STEP. The protocol uses an action–observation loop related to ReAct (Yao et al., 2023). Each variant is an independent episode. The model receives the task specification and concrete runtime input, produces one tool call, observes its result, and selects the next action until termination or the execution limit.

GLOBAL and STEP-BY-STEP differ in the information available during execution: STEP-BY-STEP receives the concrete runtime input and intermediate tool results, whereas GLOBAL constructs its workflow before observing either. We therefore use their performance difference as the gap between the two orchestration settings, rather than attributing it solely to execution feedback.

LOCAL. LOCAL receives the same task specification as GLOBAL and observes neither concrete runtime inputs nor execution results, but constructs the workflow incrementally through an append-only editor. After each proposed edit, a validator either accepts the node and returns the updated workflow draft or rejects it with an error. Thus, LOCAL tests whether incremental construction with validator feedback improves planning without runtime execution feedback.

REPLAN. REPLAN studies feedback-driven self-improvement without model-weight updates, a principle also used by Reflexion (Shinn et al., 2023). Starting from the GLOBAL workflow, we execute it on runtime variants and provide the model with its workflow and execution feedback, including tool calls, arguments, results or errors, control-flow outcomes, and terminal returns. The model then generates a revised complete workflow. Reference workflows, reference executions, and evaluation scores are never provided.

3.3 EVALUATION METRICS

Orchestration Accuracy. Our primary metric is the average of Execution Trace Accuracy and Side-Effect Accuracy for each runtime variant. *Execution Trace Accuracy.* Execution Trace Accuracy measures agreement with the reference execution trace and averages five binary components: tool selection, tool order, tool execution, required arguments, and control path. *Side-Effect Accuracy.* Side-Effect Accuracy measures whether execution produces the required state changes without missing, duplicated, or unintended effects.

Final Accuracy. Final Accuracy measures whether both the required side effects and terminal output are correct. It is counted as correct only when both Side-Effect Accuracy and Terminal Output Accuracy are correct. Terminal Output Accuracy checks whether the terminal response matches the expected output when the SOP specifies one.

Error Rate. Error Rate measures the fraction of runtime variants that do not produce a scoreable execution because of planning, model, infrastructure, or evaluation failures. These variants remain part of the evaluation and receive zero for the accuracy metrics rather than being excluded.

3.4 TRAINING FOR GLOBAL ORCHESTRATION

We next study whether global orchestration can be improved through targeted training. To align training with the GLOBAL setting, each training example requires the model to generate one complete workflow for an SOP without using any concrete runtime variant. We construct these examples from a separate set of 3,000 source recipes. The 3,000 training recipes are disjoint from the AGENTORCH benchmark by source recipe ID.

Training Data Construction For each source recipe, we construct a task specification following the same procedure used for AGENTORCH. The source recipe serves as the reference workflow. This produces one training pair per recipe: the task specification is the model input, and the complete workflow is the target output. The target preserves the full workflow structure of the source recipe, including conditional branches, loops, exception handling, tool calls, and dependencies between tool results and later operations. Runtime variants are not used to specialize the target. The resulting

training set contains 3,000 training pairs. Across the training set, the reference workflows use 1,033 tool operations spanning 305 namespaces.

3.5 Training Methods

Supervised Fine-Tuning. We first apply supervised fine-tuning to train the model to generate the reference workflow from the task specification. The training objective is applied to the generated workflow, while the task specification is provided as input context. We use the same training formulation across all evaluated base models.

Reinforcement Learning. We further apply reinforcement learning to optimize generated workflows against the reference workflow. For each sampled workflow, the reward measures whether the output is valid and how closely it matches the reference workflow in tool calls, tool arguments, and workflow structure. The reward does not execute the generated workflow or use the simulated connector backend. We optimize the model using Group Relative Policy Optimization (GRPO) (Shao et al., 2024). Training configurations for both methods are provided in Appendix A.7.

4 Experiments

We first characterize what does and does not explain the Global–Step-by-Step gap. We then study three routes for narrowing it, each drawing on a stronger signal than the last: incremental workflow construction without runtime execution feedback, workflow revision using execution feedback, and targeted training on reference workflows. Additional evaluation details are provided in Appendix A.5.

> **RQ1**: *Do improvements in model capability translate into stronger global orchestration and a smaller gap to step-by-step interaction?*

Improvements in model capability may raise both Global and Step-by-Step performance without reducing the difference between them. We therefore distinguish two outcomes: absolute improvement in Global performance and reduction in the Global–Step-by-Step gap. We examine three model-side factors: scale, model generation, and reasoning. Full results are provided in Appendix B.1.

Model Scale Does Not Consistently Improve Global Orchestration or Narrow the Gap. Figure 3 compares Global and Step-by-Step across model sizes within Qwen2.5 and Gemma-3. Neither Global Orchestration Accuracy nor the gap changes monotonically with scale. For Qwen2.5, Global Orchestration Accuracy drops from 16.18% at 32B to 9.91% at 72B, and the gap decreases from 36.33 pp at 0.5B to 27.87 pp at 7B before rising to 36.66 pp at 72B; for Gemma-3, the gap grows from 26.60 pp at 4B to 32.52 pp at 27B. Step-by-Step also retains higher Final Accuracy and lower Error Rate at every evaluated size.

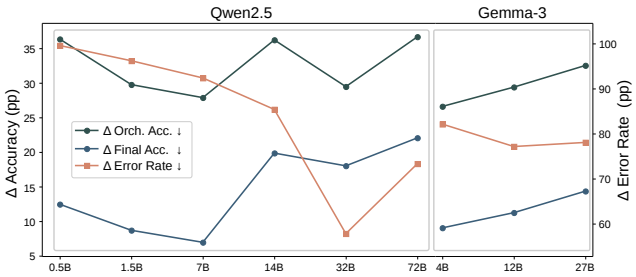


Figure 3: **Scaling does not consistently narrow the GLOBAL–STEP-BY-STEP gap.** Across Qwen2.5 and Gemma-3, the differences in Orchestration Accuracy, Final Accuracy, and Error Rate do not decrease monotonically with model size.

Newer Generations Improve Global Orchestration in Most Comparisons, but Do Not Consistently Narrow the Gap. Across the generation pairs (Table 2), newer generations improve Global Orchestration Accuracy in four pairs, by 5.80–17.98 pp, with only Llama decreasing slightly (-0.36 pp). However, the gap narrows only for Qwen (3.88 pp) and GPT (18.64 pp), and widens for Gemma, Llama, and GLM (1.39–6.18 pp). The Final Accuracy gap narrows only for GPT, while Global Error Rate decreases in all five pairs. Thus, newer generations generally strengthen Global, but stronger absolute performance does not consistently translate into a smaller gap.

Reasoning Does Not Consistently Improve Global Orchestration or Narrow the Gap. Enabling reasoning increases Global Orchestration Accuracy for four of eight models and decreases it for the other four (Table 3, Table 11), with changes ranging from +13.44 pp for Claude Opus 5 to -9.87

Table 2: **Newer model generations generally improve global orchestration, but do not consistently narrow the gap to step-by-step interaction.** Scores are reported in percent; Gap and Δ are reported in percentage points (pp). For accuracy metrics, Gap is STEP-BY-STEP minus GLOBAL; for Error Rate, Gap is GLOBAL minus STEP-BY-STEP. Δ denotes newer generation minus earlier generation.

		Qwen			Gemma			Llama			GPT			GLM		
		Qwen2.5	Qwen3		Gemma-3	Gemma-4		Llama-3.1	Llama-3.3					GLM-4.5	GLM-5.2	
Metric	**Setting**	3B	4B	Δ	12B	12B	Δ	70B	70B	Δ	GPT-3.5	GPT-4o	Δ	FP8	FP8	Δ
Orch. ↑	GLOBAL	2.19	13.75	+11.56	8.46	14.26	+5.80	10.81	10.45	-0.36	10.11	28.09	+17.98	12.93	19.23	+6.30
	STEP-BY-STEP	34.16	41.84	+7.68	37.85	49.83	+11.98	38.92	42.98	+4.06	50.78	50.12	-0.66	54.47	62.16	+7.69
	Gap	31.97	28.09	**-3.88**	29.39	35.57	**+6.18**	28.11	32.53	**+4.42**	40.67	22.03	**-18.64**	41.54	42.93	**+1.39**
Final ↑	GLOBAL	0.59	3.91	+3.32	1.83	4.07	+2.24	3.35	4.29	+0.94	4.32	10.34	+6.02	5.19	8.57	+3.38
	STEP-BY-STEP	9.50	17.30	+7.80	13.07	27.76	+14.69	14.41	20.31	+5.90	30.06	29.13	-0.93	38.29	48.20	+9.91
	Gap	8.91	13.39	**+4.48**	11.24	23.69	**+12.45**	11.06	16.02	**+4.96**	25.74	18.79	**-6.95**	33.10	39.63	**+6.53**
Error ↓	GLOBAL	94.78	61.96	-32.82	77.17	65.78	-11.39	72.24	71.46	-0.78	75.09	35.40	-39.69	75.68	61.34	-14.34
	STEP-BY-STEP	0.50	0.06	-0.44	0.03	0.16	+0.13	0.03	0.00	-0.03	0.19	0.16	-0.03	0.09	0.12	+0.03
	Gap	94.28	61.90	**-32.38**	77.14	65.62	**-11.52**	72.21	71.46	**-0.75**	74.90	35.24	**-39.66**	75.59	61.22	**-14.37**

Table 3: **Reasoning does not consistently improve global orchestration.** We report global-mode performance for base and thinking variants across models. For each base–thinking pair, the better-performing score is highlighted and bolded (higher is better for accuracy metrics; lower is better for error rate).

	Qwen3-4B		Qwen3-30B		Qwen3.8-27B		DeepSeek-V4		GLM-4.5-Air		GLM-4.5-FP8		GLM-5.2-FP8		Claude-Opus-5	
Metric	Base	Think	Base	Think	Base	Think	Base	Think	Base	Think	Base	Think	Base	Think	Base	Think
Orch.	**13.75%**	10.25%	8.35%	**14.75%**	28.57%	**33.02%**	**27.01%**	17.14%	**12.28%**	10.72%	12.93%	**20.09%**	**19.23%**	17.75%	46.44%	**59.88%**
Final	**3.91%**	2.70%	2.95%	**5.19%**	12.67%	**17.05%**	**11.43%**	9.66%	**4.78%**	4.25%	5.19%	**8.48%**	8.57%	**9.29%**	23.54%	**32.33%**
Side	**16.40%**	13.04%	9.44%	**16.83%**	31.80%	**35.71%**	**29.94%**	17.52%	**14.22%**	11.61%	13.88%	**21.46%**	**21.15%**	18.66%	50.16%	**63.98%**
Trace	**11.11%**	7.45%	7.26%	**12.68%**	25.33%	**30.33%**	**24.07%**	16.76%	**10.34%**	9.83%	11.99%	**18.72%**	**17.31%**	16.84%	42.72%	**55.78%**
Term.	**16.30%**	8.42%	13.73%	**20.12%**	28.98%	**35.16%**	**28.98%**	20.06%	11.71%	**13.45%**	10.81%	**19.88%**	**18.94%**	14.32%	42.24%	**48.29%**
Mean Acc. ↑	**12.29%**	8.37%	8.35%	**13.91%**	25.47%	**30.25%**	**24.29%**	16.23%	**10.67%**	9.97%	10.96%	**17.73%**	**17.04%**	15.37%	41.02%	**52.05%**
Error ↓	**61.96%**	77.76%	76.43%	**60.81%**	43.88%	**39.47%**	**43.51%**	67.67%	**73.76%**	74.63%	75.68%	**58.35%**	**61.34%**	73.01%	19.16%	**6.21%**

pp for DeepSeek-V4-Flash. The corresponding gap narrows for only three models and widens for five. Thus, reasoning provides neither a consistent absolute benefit to GLOBAL nor a consistent reduction in the gap.

Ablation: Access to Runtime Inputs Alone Does Not Explain the Gap We next test whether access to concrete runtime inputs accounts for the protocol gap. INPUT-CONDITIONED GLOBAL receives the runtime input but generates the complete workflow before execution, without intermediate execution results. Across six models, this changes Orchestration Accuracy by -0.18 pp on average; the mean gap to STEP-BY-STEP changes from 30.92 pp to 31.09 pp (Table 4). Providing the runtime input alone therefore leaves the average gap largely unchanged. The comparison still jointly varies sequential decision making and intermediate execution information, so it does not isolate their contributions.

Table 4: **Conditioning global orchestration on concrete runtime inputs does not narrow the gap.** Orchestration Accuracy (%); Δ is INPUT-CONDITIONED GLOBAL minus GLOBAL in percentage points.

Model	**GLOBAL**	**INPUT-COND. GLOBAL**	**STEP-BY-STEP**	**Δ**
Qwen3-4B	13.75	13.04	41.84	-0.71
Gemma-4-12B	14.26	12.73	49.83	-1.53
Gemma-4-26B	4.05	3.66	52.10	-0.39
Claude Opus 5	46.44	48.79	63.52	+2.35
DeepSeek-V4	27.01	25.62	58.99	-1.39
Qwen3.8-27B	28.57	29.19	53.31	+0.62
Average	22.35	22.18	53.27	**-0.18**

RQ2: *Can incremental workflow construction improve global orchestration without runtime execution feedback?*

Incremental Workflow Construction Improves Global Orchestration for Most Evaluated Models. Since stronger models alone do not reliably narrow the gap, we next change how the workflow is constructed while keeping the available information fixed. LOCAL and GLOBAL receive the same task specification and observe neither concrete runtime inputs nor execution results; they differ only in whether the workflow is generated in one shot or constructed incrementally with validation after each edit (Section 3.2). LOCAL improves Orchestration Accu-

Table 5: **Incremental plan construction can improve global orchestration without execution feedback.** Orchestration Accuracy (%); Δ is LOCAL minus GLOBAL.

Model	**GLOBAL**	**LOCAL**	**Δ**
Llama-3.3-70B	10.4	34.9	**+24.5**
Qwen2.5-3B	2.2	22.8	**+20.6**
Mistral-Small-3.1-24B	14.1	27.3	**+13.1**
Qwen3.8-27B-Thinking	33.0	41.5	**+8.5**
Qwen3-30B-A3B	8.4	8.5	**+0.1**
Qwen3-4B	13.8	6.9	-6.9

racy by 9.98 pp on average (Table 5), with the largest gains for Llama-3.3-70B (+24.5 pp) and Qwen2.5-3B (+20.6 pp). Qwen3-4B is the only model that declines (-6.9 pp).

RQ3: *Can execution feedback improve global workflows, and where do the gains come from?*

Execution-Feedback Revision Improves Global Orchestration. RQ2 improves GLOBAL without observing any execution. We next allow the model to observe how its workflow executes. REPLAN executes the GLOBAL workflow, returns the execution feedback to the model, and asks it to revise the complete workflow. The revised workflow still executes without further LLM calls. Over three rounds, REPLAN improves Orchestration Accuracy for all eight evaluated models, by 6.18 pp on average, and reduces Error Rate for every model (Figure 4).

The Gains Transfer to Unseen Runtime Variants. We next examine whether the improvement is limited to the runtime variants used to construct the feedback. We provide feedback from only two variants of each SOP and evaluate on the remaining held-out variants. The average held-out gain is 6.13 pp, nearly identical to the 6.18 pp obtained with feedback from all variants (Table 7), and far more individual held-out variants improve than regress (Appendix B.2). REPLAN therefore improves the workflow itself, not only its behavior on the feedback executions.

Most Gains Come from Fixing How Tools Are Called. Across the three models in the decomposition, revisions that turn invalid Round-0 workflows into valid ones account for 92.6% of the aggregate REPLAN gain (Table 6). In the eight-model analysis, most repaired workflows initially parse successfully but fail schema, type, or reference checks governing tool arguments and references between steps (Appendix B.2). GLOBAL must specify these arguments and references before execution, while STEP-BY-STEP can use observed results to construct subsequent calls. All initial failures in the repaired workflows are detectable before execution. Revising the same Round-0 workflows using only pre-execution validation errors recovers 75.5% of the aggregate REPLAN gain across the three models.

Table 6: **Decomposition of REPLAN gains.** Repair and Valid: gains from initially invalid and initially valid workflows. Validation-only: revision using pre-execution validation errors without execution. Recovery $= \Delta/(\text{Repair} + \text{Valid})$.

	REPLAN			Validation-only		
Model	**Repair**	**Valid**	**Share**	**Δ**	**Recovery**	**R–S**
Mistral-3.1-24B	+7.09	+0.52	93.2%	+6.10	80.2%	+1.51
Qwen3-30B-A3B	+2.75	+0.66	80.6%	+1.70	49.9%	+1.71
Qwen3-4B	+4.88	0.00	100.0%	+4.20	86.1%	+0.68
Across models	**+4.91**	**+0.39**	**92.6%**	**+4.00**	**75.5%**	**+1.30**

Together, these results show that global orchestration requires not only selecting the right tools, but also specifying before execution how each tool is called and how data flows between calls. Getting these specifications right is a major bottleneck of global orchestration.

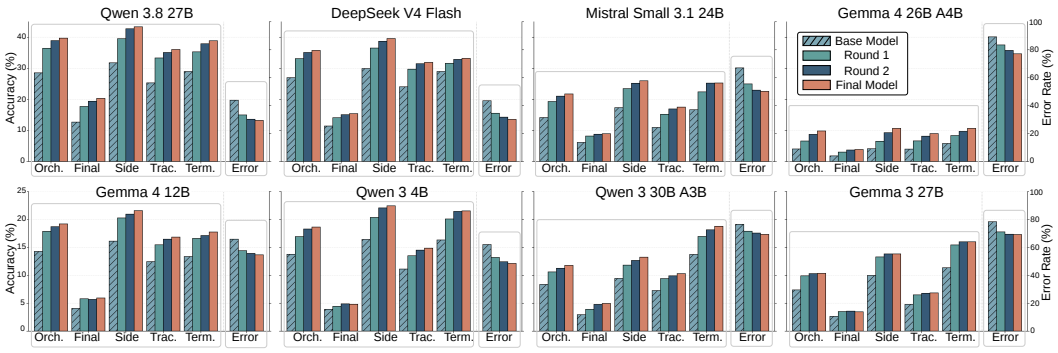


Figure 4: **REPLAN consistently improves global orchestration across models.** Performance over Rounds 0–3 for eight models when execution feedback is collected from all runtime variants of each SOP. Accuracy metrics generally increase across revision rounds, while Error Rate decreases.

RQ4: *Can stronger global orchestration be learned through targeted training?*

Table 7: **REPLAN improvements generalize to unseen runtime variants.** We report performance before REPLAN (**Base**, Round 0) and after three REPLAN rounds (**Final**, Round 3). At each round, the model receives execution feedback from only the two random runtime variants of each SOP, while evaluation is performed on the remaining held-out variants that are never used for feedback. For Mean Accuracy and Error Rate, we additionally report the percentage-point change (Δ). Mean Accuracy averages Orchestration, Final, Side-effect, Execution-trace, and Terminal-output Accuracy. Higher accuracy and lower Error Rate are better. No model weights are updated.

Model	Orch. ↑		Final ↑		Side ↑		Trace ↑		Term. ↑		Mean Acc. ↑			Error ↓		
	Base	Final	Base	Final	Base	Final	Base	Final	Base	Final	Base	Final	Δ	Base	Final	Δ
Qwen3.8-27B	29.11	40.07	14.37	22.22	33.02	44.34	25.19	35.79	28.73	38.81	26.08	36.25	**+10.16**	45.22	31.58	**-13.64**
DeepSeek-V4-Flash	27.21	35.46	12.35	16.69	30.90	40.62	23.51	30.29	28.48	33.18	24.49	31.25	**+6.76**	44.86	31.27	**-13.59**
Mistral-Small-3.1-24B	14.52	21.89	6.93	9.82	18.29	26.72	10.74	17.06	16.38	24.55	13.37	20.01	**+6.64**	67.60	51.21	**-16.39**
Gemma-4-26B-A4B	3.82	9.62	1.91	4.29	4.03	10.65	3.61	8.59	4.96	9.92	3.67	8.61	**+4.95**	90.65	79.59	**-11.06**
Gemma-4-12B	14.47	20.24	4.34	6.67	16.90	23.98	12.04	16.51	12.87	17.57	12.12	16.99	**+4.87**	66.93	54.68	**-12.25**
Qwen3-4B	14.24	19.13	4.81	6.61	17.31	23.62	11.17	14.65	15.71	21.55	12.65	17.11	**+4.46**	63.62	50.49	**-13.13**
Qwen3-30B-A3B	8.50	11.14	3.31	5.27	10.03	12.66	6.98	9.61	12.92	17.98	8.35	11.33	**+2.98**	77.26	71.11	**-6.15**
Gemma-3-27B	7.80	11.13	3.15	4.39	10.90	15.30	4.70	6.96	10.96	15.50	7.50	10.66	**+3.15**	78.55	69.15	**-9.40**

Table 8: **Targeted training improves global orchestration.** Results under GLOBAL; Δ compares the better of SFT and RL with Base (pp).

	Orch. Acc. (%) ↑				Final Acc. (%) ↑				Trace Acc. (%) ↑				Error Rate (%) ↓			
Model	Base	SFT	RL	Δ	Base	SFT	RL	Δ	Base	SFT	RL	Δ	Base	SFT	RL	Δ
Qwen3.8-27B	28.57	**41.96**	41.14	**+13.39**	12.67	**18.48**	18.26	**+5.81**	25.33	**35.87**	35.24	**+10.54**	43.88	**17.08**	19.38	−26.80
Qwen3-4B	13.75	25.46	**27.16**	**+13.41**	3.91	**10.78**	10.71	**+6.87**	11.11	19.80	**21.43**	**+10.32**	61.96	40.99	**36.12**	−25.84
Gemma-4-26B-A4B	4.05	19.89	**23.93**	**+19.88**	1.80	8.45	**9.35**	**+7.55**	3.99	15.40	**18.66**	**+14.67**	89.22	56.61	**48.23**	−40.99
Gemma-4-31B	12.71	**19.55**	19.10	**+6.84**	4.78	**9.78**	9.22	**+5.00**	12.07	**16.03**	15.93	**+3.96**	74.94	57.17	**56.55**	−18.39
Gemma-4-12B	14.26	17.92	**21.66**	**+7.40**	4.07	8.51	**8.54**	**+4.47**	12.44	14.29	**17.33**	**+4.89**	65.78	63.94	**52.76**	−13.02

Training on Complete Workflows Improves Global Orchestration for All Models. RQ2 and RQ3 improve GLOBAL at test time, and RQ3 shows that its main obstacle is producing a workflow with correct arguments and references. We next ask whether training can teach models to produce such workflows directly. We train five models on 3,000 source recipes disjoint from AGENTORCH, each pairing a task specification with a complete reference workflow (Section 3.4). Supervised fine-tuning (SFT) improves Orchestration Accuracy and Final Accuracy for all five models, and taking the better of SFT and RL for each model yields an average Orchestration Accuracy gain of 12.18 pp (Table 8). Error Rate drops even more sharply, by 13.02–40.99 pp; Qwen3.8-27B falls from 43.88% to 17.08%, below the 19.16% of Claude Opus 5 under GLOBAL.

Reinforcement Learning Adds Gains for Some Models. We next examine whether reinforcement learning (RL) further improves the SFT models. As shown in Table 8, RL further increases Orchestration Accuracy for three of five models: Qwen3-4B by 1.70 pp, Gemma-4-26B-A4B by 4.04 pp, and Gemma-4-12B by 3.74 pp. Error Rate also decreases for four models, including reductions of 8.38 pp for Gemma-4-26B-A4B and 11.18 pp for Gemma-4-12B. However, RL does not improve Orchestration Accuracy for Qwen3.8-27B or Gemma-4-31B, and Final Accuracy improves for only two models. Overall, RL can provide additional gains after SFT, but the benefit is not consistent across models or metrics.

5 CONCLUSION

We introduced AGENTORCH to study how close global orchestration can come to step-by-step interaction on the same enterprise workflows. STEP-BY-STEP remains more accurate for every evaluated model. The gap has shrunk by about 79% from early models to the strongest recent model, but this narrowing is concentrated in the latest generation: neither model scale, newer generations, nor reasoning alone consistently narrows it. We identified three routes that narrow the gap further, each drawing on a stronger signal than the last: incremental workflow construction without execution feedback, revision with execution feedback that transfers to unseen runtime variants, and targeted training on reference workflows that improves every trained model. Our analysis further shows that global orchestration requires not only selecting the right tools, but also specifying before execution how each tool is called and how data flows between calls; getting these specifications right is a key lever for future work. The gap has not closed, but it is closing: global orchestration offers a promising path toward enterprise agents that approach step-by-step accuracy with fewer LLM calls.

AI USE STATEMENT

We used generative AI tools for language editing, rewriting, and improving the organization and clarity of the manuscript. Generative AI was also used as part of the research methodology for model-based components of benchmark construction, including task-description generation and runtime-variant construction, as described in the paper and appendix. We did not rely on generative AI to determine the reported experimental results or final scientific claims. All AI-assisted text, code, data-processing outputs, and experimental analyses were reviewed and verified by the authors. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work does not involve human-subject experiments. AGENTORCH is constructed from sanitized enterprise workflow recipes, and model evaluation is performed in isolated simulated connector environments rather than on users' live software accounts. The benchmark studies workflows that may produce external side effects, such as modifying records or sending messages; accordingly, our evaluation explicitly checks both execution behavior and resulting side effects. The benchmark is intended for evaluating and improving workflow orchestration systems, and deployment of such systems in real enterprise environments should include appropriate access control, validation, and monitoring to prevent unintended actions.

REPRODUCIBILITY STATEMENT

We provide the details needed to reproduce AGENTORCH and the reported experiments throughout the paper and appendix. Section 3 describes the benchmark construction, task specification, execution environment, orchestration protocols, and evaluation metrics. Appendix A.3 details task-description generation, and Appendix A.4 describes runtime-variant construction. Training-data construction and train–benchmark separation are documented in Appendix A.6, while optimization settings and training procedures are provided in Appendix A.7. Additional evaluation details and robustness analyses are provided in Appendix A.5 and Appendix B.2, respectively.

REFERENCES

Léo Boisvert, Megh Thakkar, Maxime Gasse, Massimo Caccia, Thibault L De Chezelles, Quentin Cappart, Nicolas Chapados, Alexandre Lacoste, and Alexandre Drouin. Workarena++: Towards compositional planning and reasoning-based common knowledge work tasks. *Advances in Neural Information Processing Systems*, 37:5996–6051, 2024.

Kiran Busch and Henrik Leopold. Towards a benchmark for large language models for business process management tasks. *arXiv preprint arXiv:2410.03255*, 2024.

Shobhit Chaurasia and Raymond Mooney. Dialog for language to code. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pp. 175–180, 2017.

Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, Kai Chen, and Feng Zhao. T-eval: Evaluating the tool utilization capability of large language models step by step. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 9510–9529. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.ACL-LONG.515. URL https://doi.org/10.18653/v1/2024.acl-long.515.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36:28091–28114, 2023.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, et al. Workarena:

How capable are web agents at solving common knowledge work tasks? *arXiv preprint arXiv:2403.07718*, 2024.

Shengda Fan, Xin Cong, Yuepeng Fu, Zhong Zhang, Shuyan Zhang, Yuanwei Liu, Yesai Wu, Yankai Lin, Zhiyuan Liu, and Maosong Sun. Workflowllm: Enhancing workflow orchestration capability of large language models. In *International Conference on Learning Representations*, volume 2025, pp. 24498–24525, 2025.

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

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

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. An llm compiler for parallel function calling. *arXiv preprint arXiv:2312.04511*, 2023.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 881–905, 2024.

Humam Kourani, Alessandro Berti, Daniel Schuster, and Wil MP van der Aalst. Process modeling with large language models. In *International Conference on Business Process Modeling, Development and Support*, pp. 229–244. Springer, 2024.

Humam Kourani, Alessandro Berti, Daniel Schuster, and Wil M. P. van der Aalst. Evaluating large language models on business process modeling: framework, benchmark, and self-improvement analysis. *Softw. Syst. Model.*, 25(4):1151–1186, 2026. doi: 10.1007/S10270-025-01318-W. URL https://doi.org/10.1007/s10270-025-01318-w.

Junlong Li, Wenshuo Zhao, Jian Zhao, Weihao Zeng, Haoze Wu, Xiaochen Wang, Rui Ge, Yuxuan Cao, Yuzhen Huang, Wei Liu, et al. The tool decathlon: Benchmarking language agents for diverse, realistic, and long-horizon task execution. In *International Conference on Learning Representations*, volume 2026, pp. 53359–53394, 2026.

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

Xiaoxuan Li, Lin Ni, Xin Wang, Tang Yitong, Ruoxuan Li, Jiamou Liu, and Zhongsheng Wang. Llm-based business process models generation from textual descriptions. In Kentaro Inui, Sakriani Sakti, Haofen Wang, Derek F. Wong, Pushpak Bhattacharyya, Biplab Banerjee, Asif Ekbal, Tanmoy Chakraborty, and Dhirendra Pratap Singh (eds.), *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics, IJCNLP-AACL 2025, Mumbai, India, December 20-24, 2025*, pp. 523–533. The Asian Federation of Natural Language Processing and The Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.FINDINGS-IJCNLP.31. URL https://doi.org/10.18653/v1/2025.findings-ijcnlp.31.

Zelong Li, Shuyuan Xu, Kai Mei, Wenyue Hua, Balaji Rama, Om Raheja, Hao Wang, He Zhu, and Yongfeng Zhang. Autoflow: Automated workflow generation for large language model agents. *arXiv preprint arXiv:2407.12821*, 2024.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International conference on robotics and automation (ICRA)*, pp. 9493–9500. IEEE, 2023.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. In *International Conference on Learning Representations*, volume 2024, pp. 52989–53046, 2024.

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

Vaskar Nath, Pranav Raja, Claire Yoon, and Sean Hendryx. Toolcomp: A multi-tool reasoning & process supervision benchmark. *arXiv preprint arXiv:2501.01290*, 2025.

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

Shuofei Qiao, Runnan Fang, Zhisong Qiu, Xiaobin Wang, Ningyu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Benchmarking agentic workflow generation. In *International Conference on Learning Representations*, volume 2025, pp. 69679–69703, 2025.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *International Conference on Learning Representations*, volume 2024, pp. 9695–9717, 2024.

Chris Quirk, Raymond Mooney, and Michel Galley. Language to code: Learning semantic parsers for if-this-then-that recipes. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 878–888, 2015.

Chris Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, et al. Androidworld: A dynamic benchmarking environment for autonomous agents. In *International Conference on Learning Representations*, volume 2025, pp. 406–441, 2025.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

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.

Yongliang Shen, Kaitao Song, Xu Tan, Wenqi Zhang, Kan Ren, Siyu Yuan, Weiming Lu, Dongsheng Li, and Yueting Zhuang. Taskbench: Benchmarking large language models for task automation. *Advances in Neural Information Processing Systems*, 37:4540–4574, 2024.

Daniel Shepard and Robin Salimans. Automationbench. *arXiv preprint arXiv:2604.18934*, 2026.

Zhengliang Shi, Shen Gao, Xiuyi Chen, Yue Feng, Lingyong Yan, Haibo Shi, Dawei Yin, Pengjie Ren, Suzan Verberne, and Zhaochun Ren. Learning to use tools via cooperative and interactive agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 10642–10657, 2024.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. *arXiv preprint arXiv:2209.11302*, 2022.

Yifan Song, Weimin Xiong, Dawei Zhu, Wenhao Wu, Han Qian, Mingbo Song, Hailiang Huang, Cheng Li, Ke Wang, Rong Yao, et al. Restgpt: Connecting large language models with real-world restful apis. *arXiv preprint arXiv:2306.06624*, 2023.

Olly Styles, Sam Miller, Patricio Cerda-Mardini, Tanaya Guha, Victor Sanchez, and Bertie Vidgen. Workbench: a benchmark dataset for agents in a realistic workplace setting. *arXiv preprint arXiv:2405.00823*, 2024.

Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 11854–11864. IEEE, 2023.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*, 2023.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, 2024.

Blase Ur, Melwyn Pak Yong Ho, Stephen Brawner, Jiyun Lee, Sarah Mennicken, Noah Picard, Diane Schulze, and Michael L Littman. Trigger-action programming in the wild: An analysis of 200,000 ifttt recipes. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pp. 3227–3231, 2016.

Zhenting Wang, Qi Chang, Hemani Patel, Shashank Biju, Cheng-En Wu, Quan Liu, Aolin Ding, Alireza Rezazadeh, Ankit Parag Shah, Yujia Bao, et al. Mcp-bench: Benchmarking tool-using llm agents with complex real-world tasks via mcp servers. In *International Conference on Learning Representations*, volume 2026, pp. 97377–97425, 2026.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Shawn Ma, and Yitao Liang. Describe, explain, plan and select: interactive planning with llms enables open-world multi-task agents. *Advances in Neural Information Processing Systems*, 36:34153–34189, 2023.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

Frank Fangzheng Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Zhiruo Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, Mingyang Yang, Hao Yang Lu, Amaad Martin, Zhe Su, Leander Maben, Raj Mehta, Wayne Chi, Lawrence Jang, Yiqing Xie, Shuyan Zhou, and Graham Neubig. Theagentcompany: Benchmarking LLM agents on consequential real world tasks. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025,*

NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/0d744742f6fac4d1134c019b7cef3c8a-Abstract-Datasets_and_Benchmarks_Track.html.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022.

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, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

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.

Zhen Zeng, William Watson, Nicole Cho, Saba Rahimi, Shayleen Reynolds, Tucker Balch, and Manuela Veloso. Flowmind: automatic workflow generation with llms. In *Proceedings of the Fourth ACM International Conference on AI in Finance*, pp. 73–81, 2023.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. In *International Conference on Learning Representations*, volume 2025, pp. 34040–34077, 2025.

Yi Zhong, Buqiang Xu, Yijun Wang, Zifei Shan, Shuofei Qiao, Guozhou Zheng, and Ningyu Zhang. Chat2workflow: A benchmark for generating executable visual workflows with natural language. *arXiv preprint arXiv:2604.19667*, 2026.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*, 2023.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024.

A Benchmark and Data Construction

A.1 Source Recipes and Filtering

We start from a sanitized collection of 803 executable Workato workspace recipes. We exclude 153 recipes with fewer than two reachable enabled actions, resulting in the 650 SOPs used in AgentOrch. Each retained source recipe defines one SOP and serves as its reference workflow.

The benchmark split is defined at the source-recipe level. Each recipe appears at most once in the benchmark, while multiple runtime variants may be associated with the same recipe. The resulting benchmark contains 650 SOPs and 3,220 runtime variants.

A.2 Execution Environment Construction

We construct the execution environment from the software operations used by the source recipes. These operations are mapped to the fixed set of 64 tool interfaces exposed by AgentOrch. The same tool names and argument interfaces are used across the benchmark.

Real connector backend. The real connector backend invokes the corresponding external software services where supported. During benchmark construction, we execute each reference workflow using the real connector backend under its runtime variants. We record the active execution trace,

intermediate tool results, and terminal output produced by these executions. These records are used to construct the corresponding reference execution data.

Simulated connector backend. For scalable model evaluation, we implement a local simulated backend for the connector operations exercised by AGENTORCH. The simulated backend exposes the same tool names, argument schemas, and return interfaces as the corresponding real connectors, and reproduces the benchmark-relevant execution semantics required by the reference workflows, including state transitions, observable side effects, and supported error behavior.

The simulated backend is used as an evaluation substitute for the real connectors rather than as a different task environment. Evaluating every model-generated workflow directly against live services would be impractical at benchmark scale because external connectors impose API rate limits and quotas, incur service-dependent latency, and operate over mutable external state. The simulated backend instead provides deterministic, isolated, and reproducible executions while preserving the connector behavior exercised by the benchmark.

Simulation is also necessary for safety: executing model-generated workflows against live enterprise services would send real messages, modify real records, and produce side effects that cannot be undone.

Each runtime variant initializes a separate simulated connector state. State changes produced by one tool call are visible to subsequent calls within the same execution, and the environment is reset between executions so that different models, SOPs, and runtime variants do not interfere with one another.

All model evaluations reported in this paper execute model-generated workflows using the simulated connector backend. The real connector backend is used during benchmark construction to collect reference execution results rather than to execute the evaluated model.

A.3 Task Description Generation

Following WorkflowLLM Fan et al. (2025), we use a bottom-up workflow-to-description procedure to construct the natural-language task description for each SOP. Let A_r denote the structured representation of reference workflow r, and let D_r denote the associated connector documentation and tool schemas. The construction follows

$$A_r + D_r \longrightarrow C_r \longrightarrow P_r \longrightarrow \{Q_r^{(k)}\}_{k=1}^{K_Q} \longrightarrow Q_r,$$

where C_r contains action-level semantic annotations, P_r contains the workflow requirements across its control-flow paths, $\{Q_r^{(k)}\}$ contains candidate task descriptions, and Q_r is the selected task description.

1. **Workflow representation.** We deterministically transcribe reference workflow r into a structured representation A_r. The representation retains the executable actions and their arguments, literal values, formula expressions, branch conditions, loops, try/catch blocks, and terminal behavior. Actions without a documented tool interface are identified during this stage.
2. **Action-level annotation.** A language model produces an annotation for each workflow node using A_r and D_r. The annotation describes the corresponding operation, its arguments, dependencies on preceding results, branch conditions, and error-handling behavior. Each annotation is required to be grounded in information contained in A_r or D_r.
3. **Task-plan construction.** The action-level annotations are combined into a task plan P_r covering the requirements represented by the workflow. The plan includes the public inputs, branch conditions, loop behavior, fallback behavior, and exception handling required by the reference workflow. Runtime-variant values and hidden connector state are not included.
4. **Candidate task-description generation.** Using A_r, C_r, and P_r, the model generates $K_Q = 3$ candidate task descriptions. Each candidate describes the workflow objective, required inputs, conditions, and expected behavior without exposing recipe node identifiers or the reference execution trace.

   The requirements are not presented in the execution order of the reference workflow, so their textual presentation does not reveal the solution sequence. However, ordering requirements prescribed by the source SOP, including dependencies and temporal constraints between operations,

are retained in the task description. Thus, the model must infer the prescribed execution order from the stated workflow requirements rather than from their presentation order.

5. **Consistency checking and selection.** We extract a semantic ledger L_r from A_r containing the public inputs, operational requirements, and terminal behavior represented by the workflow. A model-based critic compares each candidate task description with L_r for two forms of consistency: whether required information from the workflow is represented in the description, and whether the description introduces requirements not supported by the workflow. The selected candidate becomes Q_r.
6. **Variant invariance and hidden-information filtering.** Each SOP uses one task description Q_r across all of its runtime variants. The description does not contain concrete variant inputs, hidden connector state, fault-injection settings, reference execution traces, or expected tool-call sequences.

The selected task description Q_r is included in the task specification described in Section 3. The remaining components of the task specification, including schemas, workspace resources, communication templates, and tool interfaces, are supplied separately. Runtime information is provided according to the orchestration protocol described in Section 3.2; in particular, GLOBAL does not receive the concrete runtime input when generating its workflow.

A.4 RUNTIME VARIANT GENERATION

Each SOP is associated with multiple runtime variants that instantiate different runtime inputs and execution conditions while keeping the task specification and reference workflow fixed. A runtime variant contains a public runtime input together with the environment configuration required to execute that input. The environment configuration may include hidden connector state and controlled tool failures.

Most benchmark variants were generated through the model-based variant-construction pipeline. Nine additional variants were constructed through coverage synthesis. The accepted variants are stored as part of the released benchmark.

Control-flow coverage. Variant construction tracks coverage obligations derived from the reference workflow. For each conditional, the construction targets both branch outcomes. For each try/catch block, it targets execution both without an exception and through the catch path. For loops, it targets zero, one, and multiple iterations when these behaviors are represented by the workflow and variant construction.

Fault injection. Variants may include a controlled failure for an operation inside an error-handling region. A fault is matched to an executed operation using the tool name together with a literal argument selector when available. If this does not uniquely identify the target call, matching falls back to the ordinal occurrence of that tool. The same fault registry and matching procedure are used when executing different orchestration protocols.

Execution isolation. Before each execution, the corresponding connector state is initialized from the runtime variant. Each execution uses an independent state, and the environment is reset before the next execution.

A.5 EVALUATION DETAILS

All orchestration protocols are scored from their realized executions using the same evaluator. For each runtime variant, the evaluator receives the executed tool trajectory, execution failures, resulting simulated connector state, externally recorded effects where applicable, and terminal output.

Reference workflows represent the prescribed enterprise SOPs rather than arbitrary implementations of underspecified tasks. Accordingly, evaluation measures fidelity to the prescribed workflow behavior, including its required operation order, as well as the resulting state and outputs.

Why evaluate fidelity to the prescribed SOP. Enterprise SOPs prescribe the procedure as well as the outcome. For example, an approval may need to precede a payment, or a duplicate check may need to precede record creation. An execution that reaches the correct final state through a different procedure can still violate compliance or audit requirements, so we treat the prescribed procedure

as part of the task. The required order is not hidden from the model: task descriptions retain the ordering constraints of the source SOP (Appendix A.3). Execution Trace Accuracy contributes only half of Orchestration Accuracy; the other half, Side-Effect Accuracy, evaluates the resulting state. Final Accuracy evaluates only side effects and terminal output, independent of the execution trajectory. The evaluator further normalizes runtime-generated identifiers and compares supported queries by result semantics rather than exact form.

Execution Trace Accuracy. Execution Trace Accuracy is the mean of five binary components: tool selection, tool order, tool execution, required arguments, and control path.

Tool selection requires the executed tool identities and call counts to match the reference SOP. Tool order measures whether the realized execution follows the prescribed operation order of the reference SOP and requires equality between the executed tool-name sequence and the reference tool-name sequence. Tool execution requires tool selection and tool order to pass and additionally requires that no unexpected execution failure occur. Required arguments evaluates the required arguments associated with the reference tool calls. Control path checks whether the realized execution follows the prescribed branch, error-handling, and termination behavior.

These components are evaluated jointly from the realized execution. For example, an additional tool call may affect tool selection, tool order, tool execution, and control-path agreement.

Required arguments. Argument scoring is performed on the arguments of executed tool calls after runtime values have been resolved. Runtime-generated identifiers covered by the benchmark normalization procedure are normalized before comparison. For standard tool arguments, the evaluator checks the required argument fields against the corresponding reference values.

The evaluator includes operation-specific handling for supported Salesforce and Snowflake query arguments, where query behavior can be compared using result or state semantics.

Side-Effect Accuracy. Side-Effect Accuracy is binary and is evaluated over the state produced in the simulated connector backend together with externally recorded effects used by the corresponding verifier. The verifier checks that required changes are present and that prohibited or unintended changes covered by the verifier are absent.

Side-Effect Accuracy evaluates the resulting state rather than the complete sequence of intermediate connector states. This complements Execution Trace Accuracy, which evaluates the realized tool trajectory.

Termination and partial executions. Executions that terminate because of a tool failure, formatting limit, or execution budget are scored using the trajectory and simulated connector state accumulated before termination.

Aggregation. Each of the five Execution Trace Accuracy components is binary, and their mean forms the per-variant Execution Trace Accuracy. Side-Effect Accuracy is binary. Orchestration Accuracy is the average of Execution Trace Accuracy and Side-Effect Accuracy for the same runtime variant. We then average the per-variant score across the benchmark. Terminal-output accuracy is reported separately.

A.6 Training Data Construction

We construct the training data from a separate set of 3,000 source recipes. Recipes are separated from the AgentOrch benchmark by source recipe ID.

For each source recipe, we construct a task specification following the same task-description procedure used for the benchmark. The source recipe serves as the reference workflow. We deterministically convert the complete reference workflow into the serialized workflow target used for training. Each recipe produces one training example; concrete runtime variants are not included in the training input and do not specialize the target.

Complete workflow targets. The training target retains the full workflow structure represented by the source recipe. This includes both outcomes of conditional branches, loop nodes and their loop bodies, try and catch regions, tool calls, dependencies on runtime inputs and preceding tool results, and terminal control behavior. Loops are represented as workflow structure rather than being expanded into the number of iterations observed in one runtime execution.

Target validation. Each target is parsed and normalized before inclusion. The validation checks the workflow syntax and structure, references between workflow nodes, availability of the tools used by the target, and the argument schemas of those tools. Compilation, schema, planner-protocol, and tool-surface checks are fail-closed.

These checks do not execute the serialized training target on runtime variants. Replay of the original source recipe is not an inclusion criterion for the active 3,000-example training set. We therefore use structural and interface validation, rather than execution validation, to describe the validation of the training targets.

Tool exposure. Across the training corpus, the tool-card pool contains 1,036 distinct tool cards spanning 306 namespaces. Of these, 1,033 tool cards spanning 305 namespaces occur in the reference workflow targets. Each individual training example exposes 64 tool cards. The 64-card set contains the tools required by that recipe together with sampled distractor tools and therefore varies across training examples.

Train–benchmark separation. Training and benchmark sets are disjoint at the source-recipe level: no source recipe used in the AGENTORCH benchmark is used for training. Both sets are drawn from the same enterprise automation environment and therefore intentionally share the same tool ecosystem and may contain recurring workflow motifs. The split is designed to evaluate generalization to unseen source recipes and task instances, rather than transfer to unseen tools or a different automation domain. Thus, all reported training results are evaluated on source recipes that are not observed during training.

A.7 Training Details

Supervised Fine-Tuning. We apply supervised fine-tuning to five base models using the training pairs described in Appendix A.6. The model receives the task specification and is trained to generate the complete reference workflow. The loss is applied only to the workflow response; tokens in the input prompt are masked from the training loss. We do not supervise chain-of-thought explanations or intermediate reasoning. All reported SFT runs use LoRA adaptation.

Reinforcement Learning. We further train models using Group Relative Policy Optimization (GRPO). The reward is computed by comparing a sampled workflow directly with the serialized reference workflow. Reward computation does not execute the sampled workflow and does not invoke either connector backend.

A sampled output must first be parsed as a workflow. If parsing fails, the total reward is zero. For valid outputs, the reward combines five components:

- **Validity** (0.15): whether the sampled response can be parsed as a valid workflow.
- **Tool calls** (0.25): we recursively flatten tool-call nodes from the generated and reference workflows and compute the longest-common-subsequence similarity between their tool-name sequences.
- **Arguments** (0.35): tool calls are aligned using the tool-name sequence alignment, after which their arguments are compared after canonical JSON serialization. The component is normalized by the number of reference tool calls.
- **Workflow structure** (0.15): we flatten the workflow-node types, including call, conditional, loop, try/catch, transformation, and termination nodes, and compute longest-common-subsequence similarity between the generated and reference node-type sequences.
- **Exact match** (0.10): the complete generated workflow receives this component when its canonical JSON serialization exactly matches the reference target.

The reward contains no execution-based component. In particular, it does not use connector observations, simulated state, Side-Effect Accuracy, or the benchmark evaluator. It also contains no separate response-length reward. Outputs that cannot be parsed, including outputs truncated before a valid workflow is completed, receive zero reward through the validity check.

For GRPO, we sample eight responses per prompt and normalize advantages within each group. The RL dataset uses 1,000 training prompts selected by prompt length to fit the configured gen-

Table 9: Training hyperparameters.

Setting	SFT	Qwen RL	Gemma RL
Method	LoRA (r=16, α=32)	GRPO	GRPO + LoRA (r=16, α=32)
Learning rate	1×10^{-4}	3×10^{-6} (27B)	1×10^{-5}
Batch / rollouts	16 seq.	16 / 8 rollouts	8 prompts $\times$ 8 gen.
Max. length	81,920	65,536 / 6,144	4,096 completion
KL coefficient	–	0.001	0.001

Table 10: **Scaling does not consistently narrow the GLOBAL–STEP-BY-STEP gap.** We report performance under the GLOBAL and STEP-BY-STEP (`react`) settings. Δ denotes STEP-BY-STEP minus GLOBAL in percentage points (pp).

		Orchestration Accuracy			Final Accuracy			Error Rate		
Family	Scale	Global	Step-by-Step	Δ	Global	Step-by-Step	Δ	Global	Step-by-Step	Δ
Qwen2.5	0.5B	0.13%	36.46%	+36.33	0.00%	12.45%	+12.45	99.78%	0.22%	−99.56
	1.5B	1.66%	31.41%	+29.75	0.22%	8.91%	+8.69	96.43%	0.25%	−96.18
	7B	2.63%	30.50%	+27.87	0.47%	7.42%	+6.95	93.45%	1.09%	−92.36
	14B	5.33%	41.54%	+36.21	1.61%	21.46%	+19.85	85.40%	0.03%	−85.37
	32B	16.18%	45.64%	+29.46	6.40%	24.41%	+18.01	58.14%	0.34%	−57.80
	72B	9.91%	46.57%	+36.66	3.91%	25.96%	+22.05	73.57%	0.25%	−73.32
Gemma-3	4B	6.15%	32.75%	+26.60	1.61%	10.65%	+9.04	82.39%	0.28%	−82.11
	12B	8.46%	37.85%	+29.39	1.83%	13.07%	+11.24	77.17%	0.03%	−77.14
	27B	7.41%	39.93%	+32.52	2.67%	17.02%	+14.35	78.14%	0.09%	−78.05

eration limits. Model-specific LoRA ranks, learning rates, KL coefficients, generation limits, and optimization settings are provided in Table 9.

B MORE EXPERIMENTAL ANALYSIS

B.1 FULL RESULTS FOR RQ1

We provide full results of our RQ1 below:

B.2 ADDITIONAL ANALYSES FOR RQ3

We examine how feedback scope affects REPLAN gains, how these gains extend to held-out runtime variants, and which failures are resolved by workflow revision. Mean Accuracy averages Orchestration, Final, Side-Effect, Execution Trace, and Terminal Output Accuracy.

Two-Variant Feedback Produces Similar Average Gains to All-Variant Feedback. We compare feedback from two variants of each SOP with feedback from all variants, evaluating both settings on all variants. Two-variant feedback improves Mean Accuracy by 5.49 pp on average, 0.05 pp more than all-variant feedback. It reduces Error Rate by 12.07 pp, 0.17 pp less than all-variant feedback (Table 12).

Held-Out Variants Improve as Much as Feedback Variants. Under two-variant feedback, we compare the improvement on the variants that provided feedback with that on held-out variants. The average held-out minus seen difference is $+0.03$ pp for Mean Accuracy and -0.31 pp for Error Rate reduction (Table 13). The revised workflow thus improves as much on variants it never observed as on those used for feedback.

Gains Also Hold on Two Held-Out Variants per SOP. Evaluating on a subset of two held-out variants per SOP, Mean Accuracy improves for all eight models by 5.34 pp on average, and Error Rate decreases by 12.31 pp (Table 14). The transfer result therefore does not depend on aggregating over all held-out variants.

Table 11: **Reasoning does not consistently improve orchestration performance.** We compare base and thinking variants under both GLOBAL and STEP-BY-STEP (`react`) execution. Within each base–thinking pair and execution setting, the better-performing score is highlighted and bolded (higher is better for accuracy metrics; lower is better for error rate). Mean accuracy is the average of orchestration, final, side-effect, execution-trace, and terminal-output accuracy.

(a) GLOBAL

	Qwen3-4B		Qwen3-30B		Qwen3.8-27B		DeepSeek-V4		GLM-4.5-Air		GLM-4.5-FP8		GLM-5.2-FP8		Claude-Opus-5	
Metric	Base	Think	Base	Think	Base	Think	Base	Think	Base	Think	Base	Think	Base	Think	Base	Think
Orch.	**13.75%**	10.25%	8.35%	**14.75%**	28.57%	**33.02%**	**27.01%**	17.14%	**12.28%**	10.72%	12.93%	**20.09%**	**19.23%**	17.75%	46.44%	**59.88%**
Final	**3.91%**	2.70%	2.95%	**5.19%**	12.67%	**17.05%**	**11.43%**	9.66%	**4.78%**	4.25%	5.19%	**8.48%**	8.57%	**9.29%**	23.54%	**32.33%**
Side	**16.40%**	13.04%	9.44%	**16.83%**	31.80%	**35.71%**	**29.94%**	17.52%	**14.22%**	11.61%	13.88%	**21.46%**	**21.15%**	18.66%	50.16%	**63.98%**
Trace	**11.11%**	7.45%	7.26%	**12.68%**	25.33%	**30.33%**	**24.07%**	16.76%	**10.34%**	9.83%	11.99%	**18.72%**	**17.31%**	16.84%	42.72%	**55.78%**
Term.	**16.30%**	8.42%	13.73%	**20.12%**	28.98%	**35.16%**	**28.98%**	20.06%	11.71%	**13.45%**	10.81%	**19.88%**	**18.94%**	14.32%	42.24%	**48.29%**
Mean Acc. ↑	**12.29%**	8.37%	8.35%	**13.91%**	25.47%	**30.25%**	**24.29%**	16.23%	**10.67%**	9.97%	10.96%	**17.73%**	**17.04%**	15.37%	41.02%	**52.05%**
Error ↓	**61.96%**	77.76%	76.43%	**60.81%**	43.88%	**39.47%**	**43.51%**	67.67%	**73.76%**	74.63%	75.68%	**58.35%**	**61.34%**	73.01%	19.16%	**6.21%**

(b) STEP-BY-STEP (`react`)

	Qwen3-4B		Qwen3-30B		Qwen3.8-27B		DeepSeek-V4		GLM-4.5-Air		GLM-4.5-FP8		GLM-5.2-FP8		Claude-Opus-5	
Metric	Base	Think	Base	Think	Base	Think	Base	Think	Base	Think	Base	Think	Base	Think	Base	Think
Orch.	41.84%	**47.34%**	47.81%	**52.09%**	53.31%	**63.08%**	**58.99%**	58.85%	48.59%	**49.49%**	54.47%	**54.68%**	62.16%	**62.66%**	63.52%	**68.41%**
Final	17.30%	**22.17%**	26.68%	**28.17%**	31.77%	**49.41%**	44.57%	**45.43%**	28.88%	**30.06%**	**38.29%**	37.92%	48.20%	**48.91%**	47.14%	**55.31%**
Side	51.37%	**57.05%**	58.57%	**63.35%**	64.91%	**72.39%**	**68.26%**	67.67%	59.69%	**59.88%**	**64.13%**	64.10%	71.58%	**72.08%**	70.09%	**74.57%**
Trace	32.32%	**37.62%**	37.04%	**40.82%**	41.71%	**53.76%**	49.73%	**50.03%**	37.48%	**39.11%**	44.81%	**45.26%**	52.73%	**53.24%**	56.95%	**62.25%**
Term.	52.17%	**53.70%**	**57.24%**	56.55%	60.06%	**72.42%**	71.21%	**71.43%**	**59.88%**	58.14%	**67.89%**	67.08%	**72.98%**	72.48%	72.36%	**77.55%**
Mean Acc. ↑	39.00%	**43.58%**	45.47%	**48.20%**	50.35%	**62.21%**	58.55%	**58.68%**	46.90%	**47.34%**	**53.92%**	53.81%	61.53%	**61.87%**	62.01%	**67.62%**
Error ↓	**0.06%**	0.16%	0.19%	**0.06%**	**0.16%**	0.71%	**0.19%**	1.68%	**0.12%**	4.04%	**0.09%**	0.19%	**0.12%**	0.53%	**0.00%**	**0.00%**

Table 12: **Effect of limiting REPLAN feedback to two runtime variants per SOP.** All columns evaluate on all runtime variants. Gaps compare two-variant feedback with all-variant feedback. Error reduction is $\text{Error}_{R0} - \text{Error}_{R3}$. All values are in pp.

Model	Two-Var. ΔMean Acc.	Gap vs. All-Var.	Two-Var. Error Red.	Gap vs. All-Var.
Qwen3.8-27B	+10.42	+0.21	14.00	-0.53
DeepSeek-V4-Flash	+6.88	-0.02	13.32	-0.07
Mistral-Small-3.1-24B	+7.02	+0.16	17.08	+0.28
Gemma-4-26B-A4B	+5.12	+0.27	11.55	-0.62
Gemma-4-12B	+4.70	+0.50	12.52	+1.31
Qwen3-4B	+4.13	-0.02	13.14	-0.34
Qwen3-30B-A3B	+2.72	-0.73	5.81	-1.37
Gemma-3-27B	+2.90	+0.03	9.16	+0.00
Average	**+5.49**	**+0.05**	**12.07**	**-0.17**

Improvements Are Widespread Across Held-Out Variants. To check whether the average gain is driven by a few variants, we count held-out variants whose score changes from Round 0 to Round 3. For every model, improvements outnumber regressions on both Orchestration and Final Accuracy. In total across models, Orchestration Accuracy improves on 2,214 held-out variants and decreases on 165, while Final Accuracy improves on 555 and decreases on 75 (Table 15).

Most Repaired Workflows Initially Fail Schema, Type, or Reference Checks. Across the eight models, 641 workflows are invalid at Round 0 and become valid by Round 3. Of these, 538 (83.9%) initially parse successfully but fail schema, type, or reference validation. These checks cover constraints on tool arguments and references between workflow steps. The remaining 103 workflows (16.1%) fail response-parsing or plan-structure checks (Table 16). The share of parsing or plan-structure failures is highest for Gemma-4-26B-A4B, at 39.8%. Together, these checks detect the initial failures in all 641 repaired workflows before execution.

Summary. REPLAN gains require feedback from only two variants per SOP, extend equally to variants never used for feedback, and are spread across most held-out variants rather than concentrated in a few. The invalid workflows it repairs mostly parse correctly but specify tool arguments or references between steps incorrectly, and all of these defects are detectable before execution. Consistent with the main text, these results identify workflow validity, particularly correct arguments and references, as a major bottleneck in global orchestration.

Table 13: **Transfer from feedback variants to held-out variants.** REPLAN feedback uses two variants of each SOP. The transfer gap is the held-out value minus the seen value; values near zero indicate similar improvement on both subsets.

Model	Seen ΔMean Acc.	Held-out–Seen Gap	Seen Error Red.	Held-out–Seen Gap
Qwen3.8-27B	+10.81	-0.65	14.55	-0.91
DeepSeek-V4-Flash	+7.07	-0.31	12.92	+0.67
Mistral-Small-3.1-24B	+7.59	-0.96	18.13	-1.74
Gemma-4-26B-A4B	+5.38	-0.43	12.29	-1.23
Gemma-4-12B	+4.45	+0.42	12.92	-0.67
Qwen3-4B	+3.63	+0.83	13.16	-0.03
Qwen3-30B-A3B	+2.32	+0.66	5.30	+0.85
Gemma-3-27B	+2.52	+0.64	8.79	+0.61
Average	**+5.47**	**+0.03**	**12.26**	**-0.31**

Table 14: **REPLAN performance on a subset of two held-out variants per SOP.** Feedback uses two variants of each SOP; evaluation uses two other variants never used for feedback. Base is Round 0 and Final is Round 3.

Model	Mean Accuracy ↑			Error Rate ↓		
	Base	Final	Δ	Base	Final	Δ
Qwen3.8-27B	26.27	36.13	+9.86	42.96	28.43	-14.53
DeepSeek-V4-Flash	24.72	31.05	+6.32	42.87	29.69	-13.18
Mistral-Small-3.1-24B	13.31	20.40	+7.09	67.35	48.79	-18.56
Gemma-4-26B-A4B	4.12	9.20	+5.09	88.88	77.31	-11.57
Gemma-4-12B	12.17	16.89	+4.72	65.65	53.09	-12.56
Qwen3-4B	12.80	16.92	+4.12	60.81	47.26	-13.55
Qwen3-30B-A3B	8.32	10.93	+2.61	75.96	70.67	-5.29
Gemma-3-27B	6.93	9.83	+2.91	79.10	69.87	-9.23
Average	**13.58**	**18.92**	**+5.34**	**65.45**	**53.14**	**-12.31**

C ADDITIONAL RELATED WORK

Workflow generation and process automation. The closest work studies workflow generation and automation. WorfBench (Qiao et al., 2025), WorkflowLLM (Fan et al., 2025), AutoFlow (Li et al., 2024), AFlow (Zhang et al., 2025), FlowMind (Zeng et al., 2023), Chat2Workflow (Zhong et al., 2026), and AutomationBench (Shepard & Salimans, 2026) consider agentic pipelines, visual workflows, RPA-style procedures, or cross-application automation. Earlier work studies IFTTT-style trigger-action programming (Quirk et al., 2015; Chaurasia & Mooney, 2017; Ur et al., 2016), while recent business-process research explores language-based process modeling and generation (Kourani et al., 2024; Busch & Leopold, 2024; Kourani et al., 2026; Li et al., 2025). Our work differs by treating complete workflow generation as the GLOBAL orchestration paradigm and directly comparing it with STEP-BY-STEP interaction on the same executable enterprise workflows, including connector contracts, parameter bindings, dependencies, transformations, branching, looping, and observable side effects. We further study how global orchestration can be improved through execution feedback and SOP-grounded supervision.

D SIMULATOR FIDELITY VIA REFERENCE-ACTION REPLAY

Model-generated workflows in AGENTORCH are evaluated using the simulated connector backend, while reference executions are collected from the corresponding real connectors where supported. We therefore validate whether the simulator preserves the benchmark-relevant behavior observed in the real-backend reference executions.

Specifically, we replay the recorded reference tool calls from each real-backend execution one action at a time in the simulated backend, without any model planning. The resulting executions are scored

Table 15: **Variant-level improvements and regressions on held-out variants.** Counts of held-out variants whose score increases (Up) or decreases (Down) from Round 0 to Round 3 under two-variant feedback. Unchanged variants are omitted.

Model	Orch. Up	Orch. Down	Final Up	Final Down
Qwen3.8-27B	428	31	161	9
DeepSeek-V4-Flash	335	51	110	26
Mistral-Small-3.1-24B	357	31	68	12
Gemma-4-26B-A4B	240	12	56	10
Gemma-4-12B	264	6	49	4
Qwen3-4B	249	12	41	6
Qwen3-30B-A3B	155	12	43	5
Gemma-3-27B	186	10	27	3
Total	**2214**	**165**	**555**	**75**

Table 16: **Failure types of invalid workflows repaired by REPLAN.** Repaired workflows are invalid at Round 0 and valid by Round 3. Parse/Shape: failures in response parsing or plan-structure checks. Validator: workflows that parse successfully but fail schema, type, or reference validation. Together, these checks detect all 641 repaired workflows before execution.

Model	Repaired	Parse / Shape	Validator	Validator (%)
Qwen3.8-27B	97	16	81	83.5
DeepSeek-V4-Flash	84	12	72	85.7
Mistral-Small-3.1-24B	117	13	104	88.9
Gemma-4-12B	79	9	70	88.6
Qwen3-4B	85	13	72	84.7
Qwen3-30B-A3B	40	5	35	87.5
Gemma-3-27B	56	2	54	96.4
Gemma-4-26B-A4B	83	33	50	60.2
All models	**641**	**103**	**538**	**83.9**

using the same evaluation pipeline as model-generated runs, including the same normalization of dynamic identifiers, timestamps, argument values, and query semantics.

Table 17 reports the agreement between these simulated replays and the stored real-backend reference executions. The simulator achieves 97.20% agreement on tool selection, tool order, tool execution, and control path, 99.97% on required arguments, 97.75% Execution Trace Accuracy, and 99.32% Side-Effect Accuracy. The resulting Orchestration Accuracy is 98.54%.

These results show that the simulated backend reproduces the benchmark-relevant execution behavior of the recorded real-backend references with high fidelity. This makes broad simulator–reference mismatch an unlikely explanation for the observed performance gap between GLOBAL and STEP-BY-STEP orchestration.

E PERFORMANCE ACROSS WORKFLOW CHARACTERISTICS

We next examine whether the gap between GLOBAL and STEP-BY-STEP is concentrated in particular types of workflows. We characterize each SOP using its reference workflow. Workflow size is measured by the number of tool-call nodes, with short, medium, and long workflows defined as 1–2, 3–5, and at least 6 calls, respectively, based on the quartiles of the benchmark distribution. We additionally record whether the workflow contains conditional branches, loops, or try/catch regions, and measure its maximum nesting depth. To characterize cross-application workflows, we count the number of distinct application families used by tool-call nodes in the reference workflow, excluding runtime-helper tools. Fault injection is defined at the runtime-variant level.

We report results over the 31 models for which both GLOBAL and STEP-BY-STEP evaluations are complete. Fine-tuned checkpoints are excluded from this analysis. Failed executions remain in the

Table 17: Simulator fidelity under reference-action replay. Recorded tool calls from the real-backend reference executions are replayed one action at a time in the simulated backend and evaluated using the same benchmark evaluator as model-generated runs.

Metric	Reference-Action Replay
Tool Selection	97.20%
Tool Order	97.20%
Tool Execution	97.20%
Required Arguments	99.97%
Control Path	97.20%
Execution Trace Accuracy	97.75%
Side-Effect Accuracy	99.32%
Orchestration Accuracy	98.54%

Table 18: Performance across workflow characteristics. Orchestration Accuracy and Error Rate are reported in percent. The gap is STEP-BY-STEP minus GLOBAL Orchestration Accuracy in percentage points. Confidence intervals are 95% bootstrap intervals clustered by SOP. Workflow characteristics are derived from the reference workflow, except fault injection, which is defined per runtime variant.

Subset	# SOP	# Var.	GLOBAL	STEP-BY-STEP	Gap	95% CI	GLOBAL Err.	STEP Err.
All workflows								
All	650	3220	13.8	46.4	+32.6	[31.2, 34.2]	70.7	0.5
Workflow size								
Short (1–2 calls)	234	1035	18.8	45.2	+26.3	[24.5, 28.4]	60.9	0.5
Medium (3–5 calls)	251	1149	14.2	47.3	+33.1	[30.9, 35.1]	69.3	0.5
Long ($\geq$6 calls)	165	1036	8.3	46.6	+38.3	[35.1, 41.3]	82.0	0.7
Number of application families								
1	230	1167	18.8	51.2	+32.4	[30.3, 34.7]	65.4	0.4
2	360	1676	11.7	44.2	+32.5	[30.4, 34.8]	72.1	0.6
3+	55	352	5.9	39.4	+33.4	[28.5, 38.3]	82.4	0.7
Conditional branches								
No conditional	303	1239	14.2	40.7	+26.5	[24.6, 28.3]	66.5	0.6
Has conditional	347	1981	13.6	50.0	+36.4	[34.3, 38.4]	73.3	0.5
Loops								
No loop	511	2413	15.6	47.1	+31.5	[29.9, 33.1]	67.9	0.4
Has loop	139	807	8.5	44.3	+35.7	[32.5, 39.2]	78.9	0.8
Try/catch regions								
No try/catch	413	1929	13.7	43.2	+29.5	[27.7, 31.4]	67.9	0.6
Has try/catch	237	1291	14.0	51.2	+37.2	[34.7, 39.6]	74.8	0.5
Control-flow composition								
Linear	186	781	14.4	39.7	+25.3	[22.7, 28.0]	64.2	0.5
Branch only	132	688	14.9	48.6	+33.6	[30.2, 36.9]	69.4	0.5
Loop only	36	148	8.9	30.9	+22.0	[18.6, 26.2]	70.4	0.7
Error handling only	78	295	16.6	48.8	+32.2	[28.6, 36.1]	69.6	0.5
Multiple control-flow types	218	1308	12.8	50.4	+37.7	[35.2, 40.4]	75.5	0.6
Maximum nesting depth								
1	186	781	14.4	39.7	+25.3	[22.9, 28.0]	64.2	0.5
2	235	1030	15.0	47.8	+32.8	[30.4, 35.3]	69.3	0.4
3+	229	1409	12.6	49.0	+36.5	[34.2, 38.8]	75.3	0.6
Fault injection								
No injected fault	650	2956	13.6	45.8	+32.2	[30.7, 33.8]	70.4	0.5
Injected fault	224	264	16.4	52.8	+36.5	[33.8, 39.3]	74.2	0.6

evaluation and receive zero accuracy, following the main benchmark evaluation. The gap is defined as STEP-BY-STEP Orchestration Accuracy minus GLOBAL Orchestration Accuracy. Confidence intervals are computed using a bootstrap clustered by SOP.

The gap persists across workflow characteristics. As shown in Table 18, STEP-BY-STEP achieves higher Orchestration Accuracy than GLOBAL in every reported subgroup. This pattern also holds at the model level: all 45 evaluated models have a positive STEP-BY-STEP–GLOBAL gap in each reported subgroup. Thus, the aggregate gap is not attributable to a single workflow-size, application-count, or control-flow subgroup.

The gap increases with workflow size. The Orchestration Accuracy gap increases from 26.3 percentage points (pp) for short workflows to 33.1 pp for medium workflows and 38.3 pp for long

workflows. This increase is driven primarily by lower GLOBAL performance as workflow size increases: GLOBAL decreases from 18.8% on short workflows to 8.3% on long workflows, while STEP-BY-STEP remains between 45.2% and 47.3% across the three size groups. The GLOBAL Error Rate also increases from 60.9% to 82.0% across these groups, whereas the STEP-BY-STEP Error Rate remains below 1%.

Control-flow structure shows a similar pattern. Workflows containing conditional branches have a 36.4 pp gap, compared with 26.5 pp for workflows without conditionals. The corresponding gaps are 35.7 pp versus 31.5 pp for workflows with and without loops, and 37.2 pp versus 29.5 pp for workflows with and without try/catch regions. When workflows are grouped by control-flow composition, linear workflows have a 25.3 pp gap, while workflows containing multiple control-flow types have a 37.7 pp gap. The same pattern appears with nesting depth: the gap is 25.3 pp at depth 1, 32.8 pp at depth 2, and 36.5 pp at depth 3 or greater. Together, these stratifications show that the STEP-BY-STEP–GLOBAL gap is larger in workflows with greater structural complexity, as measured by workflow size, control-flow composition, and nesting depth.

Application count does not explain the gap. The gap is 32.4 pp for workflows using one application family, 32.5 pp for two application families, and 33.4 pp for three or more application families. Both protocols obtain lower Orchestration Accuracy as the number of application families increases, but their difference remains similar across these groups. Thus, the larger gaps observed for longer and more structurally nested workflows are not reproduced by application count alone.

The gap also persists under fault injection. For runtime variants without injected faults, the gap is 32.2 pp, compared with 36.5 pp for fault-injected variants. Both settings therefore preserve the main comparison between GLOBAL and STEP-BY-STEP. We do not interpret the higher absolute accuracy on fault-injected variants as an effect of fault injection because the two variant groups differ in the workflows and execution paths they contain.

Overall, these results show that the advantage of STEP-BY-STEP over GLOBAL is not concentrated in a particular workflow subgroup. The gap is present across workflow sizes, application counts, control-flow structures, nesting depths, and fault-injection settings, and is larger under several measures of workflow structural complexity.

F INFERENCE COST

GLOBAL constructs a complete workflow before execution, whereas STEP-BY-STEP repeatedly invokes the model as execution progresses. Table 19 compares the resulting inference usage across open-weight and closed-source models.

LLM calls. The reduction in repeated inference is consistent across model families. Across the models in Table 19, STEP-BY-STEP uses $15.4\times$–$35.4\times$ more LLM calls than GLOBAL. For example, Qwen3-30B-A3B increases from 650 calls under GLOBAL to 22,997 under STEP-BY-STEP, while Claude Opus 5 Thinking increases from 650 to 11,964 calls. On a per-variant basis, GLOBAL uses approximately 0.2 calls, compared with 3.1–7.1 calls for STEP-BY-STEP among the models shown.

Completion tokens. The larger number of calls also increases total generation. Across the models in Table 19, STEP-BY-STEP generates $1.42\times$–$5.29\times$ more completion tokens than GLOBAL. For Qwen3.8-27B, completion-token usage increases from 1.07M under GLOBAL to 5.67M under STEP-BY-STEP. Claude Opus 5 Thinking increases from 0.91M to 1.30M, while GPT-5.6 Sol Thinking increases from 0.59M to 1.40M.

Individual GLOBAL calls are substantially longer because each call generates a complete workflow. Qwen3.8-27B, for example, generates 1,652 completion tokens per call under GLOBAL, compared with 323 under STEP-BY-STEP. This pattern holds broadly across the models in the table: GLOBAL uses fewer, longer calls, whereas STEP-BY-STEP uses many more, shorter calls. The substantially larger number of STEP-BY-STEP calls nevertheless results in higher total completion-token usage.

Summary. GLOBAL substantially reduces inference usage: across the evaluated models, STEP-BY-STEP requires $15.4\times$–$35.4\times$ more LLM calls and $1.42\times$–$5.29\times$ more completion tokens.

	Total LLM Calls			Calls / Variant		Completion Tokens (M)			Tokens / Variant		Tokens / Call	
Model	GLOBAL	STEP	×	GLOBAL	STEP	GLOBAL	STEP	×	GLOBAL	STEP	GLOBAL	STEP
Open-weight models												
Gemma-3-12B	649	12,098	**18.6**×	0.20	3.76	0.54	1.92	**3.55**×	167.8	595.7	833	159
Gemma-3-27B	649	14,253	**22.0**×	0.20	4.43	0.95	2.49	**2.62**×	295.8	774.3	1,468	175
Gemma-4-12B	649	13,888	**21.4**×	0.20	4.31	1.18	3.29	**2.79**×	365.1	1,020.3	1,812	237
Gemma-4-26B-A4B	649	17,198	**26.5**×	0.20	5.34	0.95	4.10	**4.30**×	296.2	1,272.9	1,470	238
Gemma-4-31B	649	12,484	**19.2**×	0.20	3.88	1.16	2.89	**2.48**×	360.9	896.3	1,791	231
GLM-4.5-Air	650	22,338	**34.4**×	0.20	6.94	0.85	4.21	**4.98**×	262.6	1,307.1	1,301	188
GLM-4.5	650	15,488	**23.8**×	0.20	4.81	0.88	3.39	**3.87**×	272.0	1,053.7	1,348	219
GLM-5.2	650	15,228	**23.4**×	0.20	4.73	0.81	2.55	**3.13**×	252.8	791.8	1,252	167
Llama-3.3-70B	650	14,440	**22.2**×	0.20	4.48	0.72	1.59	**2.23**×	222.5	495.3	1,102	110
Qwen2.5-72B	650	17,290	**26.6**×	0.20	5.37	0.99	2.21	**2.24**×	307.1	686.9	1,521	128
Qwen3-30B-A3B	650	22,997	**35.4**×	0.20	7.14	1.13	3.71	**3.28**×	350.7	1,151.3	1,737	161
Qwen3.8-27B	649	17,539	**27.0**×	0.20	5.45	1.07	5.67	**5.29**×	332.9	1,759.6	1,652	323
Closed-source models												
GPT-3.5 Turbo	650	12,742	**19.6**×	0.20	3.96	0.45	1.93	**4.30**×	139.7	600.1	692	152
GPT-4o	650	10,042	**15.4**×	0.20	3.12	0.68	1.08	**1.60**×	210.5	336.1	1,043	108
Claude Opus 5 Thinking	650	11,964	**18.4**×	0.20	3.72	0.91	1.30	**1.42**×	284.0	402.9	1,407	108
GPT-5.6 Sol Thinking	647	11,312	**17.5**×	0.20	3.51	0.59	1.40	**2.37**×	184.3	436.2	917	124

Table 19: **Inference usage under global and step-by-step orchestration.** The × columns report STEP-BY-STEP/GLOBAL. For closed-source models, completion-token counts are estimated from retained output text using median characters-per-token ratios measured from the corresponding non-thinking runs; all other quantities are directly measured.

G HUMAN EVALUATION.

Source Recipes. AGENTORCH is constructed from sanitized executable workflows instantiated in a Workato automation environment prior to benchmark construction. Customer-identifying information is removed before benchmark artifacts are derived from these workflows.

Human Review of Task Descriptions. Four authors participated in an iterative manual validation process covering all benchmark task descriptions. In each round, task descriptions were reviewed against their corresponding source workflows for omitted workflow requirements, unsupported requirements, and disclosure of information excluded from the model's task specification. Reviews were cross-checked across authors, and descriptions with identified issues were regenerated and subsequently re-evaluated. In the first review round, approximately 57% of the task descriptions required regeneration. We repeated this review–regeneration–validation procedure for five rounds, until all task descriptions passed the review criteria.

Execution-Based and Human Validation of Runtime Variants. For every generated runtime variant, we executed the corresponding reference workflow in the simulated environment and manually inspected the resulting execution behavior. The execution-based check verified that the runtime input and environment configuration supported the intended workflow execution, while human review examined the resulting tool outputs, state changes, control-flow behavior, and terminal outputs for consistency with the source workflow and intended runtime condition. In the first validation round, approximately 38.5% of generated runtime variants passed these checks. Variants that failed either the execution-based or human validation criteria were regenerated and subsequently re-executed and re-reviewed. This generate–execute–review cycle was repeated until all retained runtime variants passed the predefined validation criteria. Because the benchmark infrastructure provides a local simulated execution environment with the same tool interfaces and benchmark-relevant connector semantics as the corresponding real connectors, we were able to inspect the execution outcome of every runtime variant rather than relying only on static input checks.

H LIMITATIONS

Scope. AGENTORCH targets structured enterprise workflows, where procedures are prescribed in advance and the same workflow is executed repeatedly on different inputs. This is the setting in which global orchestration is most practical. In GUI-based or open-ended environments, a complete executable workflow often cannot be specified before execution, and we leave the comparison in

such settings to future work. Our workflows are derived from sanitized, executable Workato recipes spanning 64 tool interfaces across services such as Slack, Jira, Salesforce, and Snowflake; real enterprise workflows of this kind are rarely publicly available. Training and benchmark sets are disjoint by source recipe but drawn from the same enterprise automation environment. This mirrors deployment, where an organization trains on its existing workflows and applies the model to new workflows over the same tool ecosystem. The training set is also broader than the benchmark, covering 1,033 tool operations across 305 namespaces with only partial overlap with the benchmark tools. Transfer to unrelated automation domains remains to be studied.

Protocol comparison. We compare GLOBAL and STEP-BY-STEP as deployed paradigms, so they differ in both when decisions are made and what information is available. Our additional protocols vary these factors separately: providing concrete runtime inputs alone leaves the gap largely unchanged, incremental construction adds step-wise decisions without execution information, and execution-feedback revision adds execution information without runtime decisions. A complete separation of the two factors is not possible, because using intermediate results at runtime without new decisions is precisely what a global workflow does through references between steps.

Evaluation. Our primary metric measures fidelity to the prescribed SOP, including its operation order, because enterprise procedures constrain how a task is carried out, not only its outcome. Final Accuracy, which evaluates only side effects and terminal output, complements this metric, and the gap and the gains from revision and training also hold under it. Model-generated workflows are executed in a simulated backend, since executing them against live enterprise services would produce irreversible side effects; reference-action replay shows high agreement between the simulator and the real connectors (Appendix D).

I FULL RESULTS FOR FIGURE 1

Table 20 reports the Orchestration Accuracy of all 31 models shown in Figure 1 under GLOBAL orchestration and STEP-BY-STEP interaction, from which the per-model gaps in the figure are computed.

Table 20: Orchestration accuracy under GLOBAL orchestration and STEP-BY-STEP (ReAct) interaction. Results are reported as percentages.

Model	**GLOBAL (%)**	**STEP-BY-STEP / ReAct (%)**
GPT-3.5 Turbo	10.11	50.78
GPT-4o	28.09	50.12
Llama 3.1 8B	4.45	34.82
Llama 3.1 70B	10.81	38.92
Mistral Nemo	2.26	34.27
Command R	6.64	35.97
Qwen 2.5 0.5B	0.13	36.46
Qwen 2.5 1.5B	1.66	31.41
Qwen 2.5 3B	2.19	34.16
Qwen 2.5 7B	2.63	30.50
Qwen 2.5 14B	5.33	41.54
Qwen 2.5 32B	16.18	45.64
Qwen 2.5 72B	9.91	46.57
Llama 3.2 1B	0.00	36.93
Llama 3.2 3B	2.07	35.50
Granite 3.1 2B	6.30	31.83
Granite 3.1 8B	6.52	34.19
Llama 3.3 70B	10.45	42.98
Gemma 3 4B	6.15	32.75
Gemma 3 12B	8.46	37.85
Gemma 3 27B	7.41	39.93
Mistral Small 3.1 24B	14.15	44.71
Qwen 3 4B	13.75	41.84
Qwen 3 30B	8.35	47.81
GLM-4.5 Air FP8	12.28	48.59
GLM-4.5 FP8	12.93	54.47
DeepSeek V4 Flash	27.01	58.99
Qwen 3.8 27B FP8	28.57	53.31
Claude Opus 5	46.44	63.52
GPT-5.6 Sol Thinking	51.76	66.88
Claude Opus 5 Thinking	59.88	68.41