

# 000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015 016 017 018 019 020 021 022 023 024 025 026 027 028 029 030 031 032 033 034 035 036 037 038 039 040 041 042 043 044 045 046 047 048 049 050 051 052 053 MEGAFLOW: LARGE-SCALE DISTRIBUTED ORCHESTRATION SYSTEM FOR THE AGENTIC ERA

Anonymous authors

Paper under double-blind review

## ABSTRACT

The rapid development of interactive and autonomous AI systems signals our entry into the agentic era. Training and evaluating agents on complex agentic tasks such as *software engineering* and *computer use* requires not only efficient model computation but also sophisticated infrastructure capable of coordinating vast agent-environment interactions. However, no open-source infrastructure can effectively support large-scale training and evaluation on such complex agentic tasks. To address this challenge, we present **MegaFlow**, a large-scale distributed orchestration system that enables efficient scheduling, resource allocation, and fine-grained task management for agent-environment workloads. MegaFlow abstracts agent training infrastructure into three independent services (*Model Service*, *Agent Service*, and *Environment Service*) that interact through unified interfaces, enabling independent scaling and flexible resource allocation across diverse agent-environment configurations. In our agent training deployments, MegaFlow successfully orchestrates tens of thousands of concurrent agent tasks while maintaining high system stability and achieving efficient resource utilization. By enabling such large-scale agent training, MegaFlow addresses a critical infrastructure gap in the emerging agentic AI landscape.



Figure 1: The proposed three-service architecture for agent training. **(left)** The *Environment Service* provides diverse interactive execution environments and returns feedback (observations, rewards, termination signals) in response to actions. **(middle)** The *Agent Service* orchestrates interaction, collects trajectories, and manages experiences. **(right)** The *Model Service* supports both inference (returning policies from context) and training (updating from experiences).

## 1 INTRODUCTION

The rapid development of interactive and autonomous AI systems signals our entry into the agentic era, where intelligent agents must be trained and evaluated on increasingly complex real-world tasks (Wang et al., 2024a; Xi et al., 2025). This transformation is driven by remarkable advances in large language models, reinforcement learning, and multi-agent coordination, enabling the development of agents capable of sophisticated reasoning and planning across diverse domains (Dorri et al., 2018). Training these agents on complex agentic tasks such as software engineering (Jimenez et al., 2023) and computer use (Xie et al., 2024) requires not only efficient model computation but also sophisticated infrastructure capable of orchestrating vast agent-environment interactions at unprecedented scale (Gao et al., 2024; Sun et al., 2025). The promise of large-scale agent training lies in its potential to develop more capable and versatile AI systems through massive parallel training across heterogeneous environments and tasks. Realizing this vision requires sophisticated infrastructure capable of supporting large-scale agent training and evaluation. However, no existing infrastructure

054 can effectively support the large-scale training and evaluation demands of such complex agentic  
 055 tasks.

056 Traditional approaches to agent training work well for simple tasks such as single-turn function  
 057 calling (Patil et al.) and basic question answering (Mialon et al., 2023). Nonetheless, they fail to  
 058 address the unique challenges of orchestrating massive numbers of concurrent agent-environment  
 059 interactions required for effective training on complex multi-step tasks at scale. The core challenge  
 060 lies not merely in computational power (modern distributed computing frameworks have adequately  
 061 addressed model training and inference scalability) but in the complex coordination of dynamic,  
 062 interdependent processes that characterize large-scale agentic training workloads. Our experience  
 063 training agents on complex tasks such as *software engineering* and *computer use automation* re-  
 064 veals three critical infrastructure bottlenecks that exemplify the broader scalability challenges facing  
 065 large-scale agent training: (1) *Security and Isolation Constraints*: Complex agent training requires  
 066 containerized environments to provide secure, isolated execution contexts for agent-environment  
 067 interactions. However, security policies in typical training clusters prohibit the execution of ar-  
 068bitrary containers, creating a fundamental incompatibility between large-scale agent training re-  
 069 quirements and existing computational infrastructure. (2) *Storage Scalability Limitations*: Each  
 070 complex agent task instance requires corresponding containerized environments containing spe-  
 071 cific software dependencies and execution contexts. Even relatively modest datasets such as SWE-  
 072 bench (Jimenez et al., 2023) and SWE-Gym (Pan et al., 2024) require over 25TB of storage for  
 073 their associated container images. Storage requirements grow dramatically as training scales to  
 074 larger and more diverse task sets, creating prohibitive infrastructure costs and management over-  
 075 head. (3) *Computational Throughput Bottlenecks*: The resource-intensive nature of containerized  
 076 agent-environment interactions severely limits concurrent training throughput, preventing the mas-  
 077 sive parallelism necessary for effective large-scale agent training.

078 To address these fundamental infrastructure challenges, we present **MegaFlow**, a large-scale dis-  
 079 tributed orchestration system that enables efficient scheduling, resource allocation, and fine-grained  
 080 task management for agent training workloads. MegaFlow abstracts agent training infrastructure  
 081 into three independent services (*Model Service*, *Agent Service*, and *Environment Service*) that inter-  
 082 act through unified interfaces, enabling independent scaling and flexible resource allocation across  
 083 diverse agent-environment configurations. The *Environment Service* provides diverse interactive ex-  
 084 ecution environments and returns feedback (observations, rewards, termination signals) in response  
 085 to actions. The *Agent Service* orchestrates interactions, collects experiences, and manages experi-  
 086 ence data throughout the training process. The *Model Service* supports both inference (returning  
 087 policies from context) and training (updating model parameters from collected experiences). While  
 088 existing approaches treat agent training as monolithic computational tasks, this modular architecture  
 089 enables independent optimization and scaling of each component according to its specific computa-  
 090 tional requirements. The key insight underlying MegaFlow is that the primary scalability bottleneck  
 091 in large-scale agent training lies not in model computation (which existing distributed frameworks  
 092 handle well) but in the efficient coordination of dynamic agent-environment interactions. By pro-  
 093 viding unified APIs for the orchestration of these three services, MegaFlow enables researchers and  
 094 practitioners to focus on algorithmic development rather than infrastructure complexity.

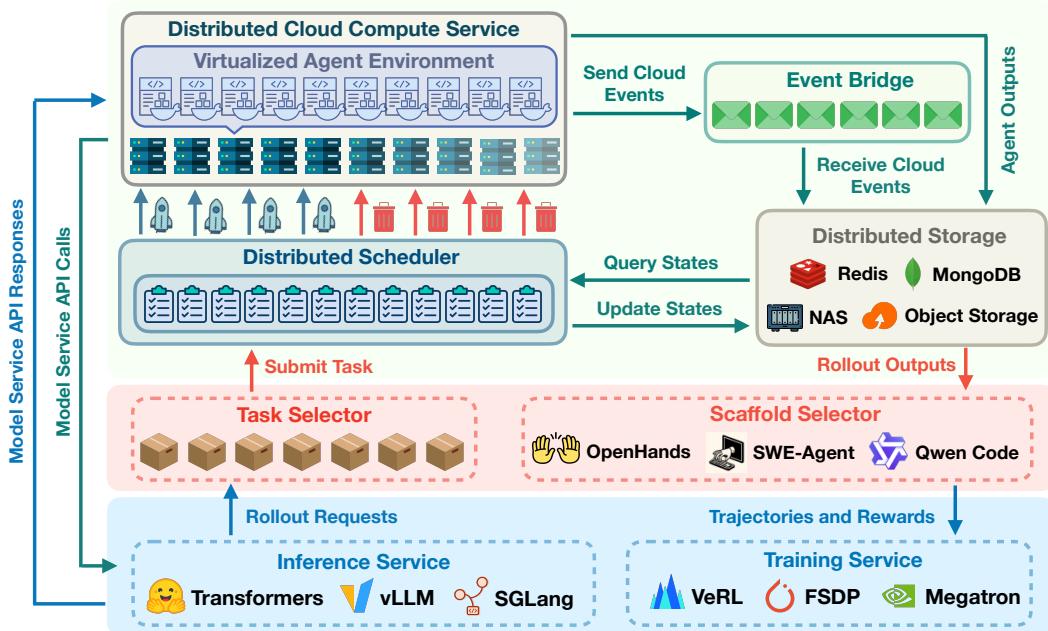
095 This work makes the following key contributions to large-scale agent training infrastructure:

- 096 • **Overcoming Security and Isolation Constraints:** We address the fundamental incompatibility  
 097 between agent training requirements and cluster security policies by migrating containerized  
 098 workloads to elastic cloud compute services. This enables secure, isolated  
 099 agent execution without requiring specialized cluster configurations or compromising ex-  
 100 isting security frameworks.
- 101 • **Solving Storage Scalability Limitations:** We implement on-demand container image pro-  
 102 visioning using cloud registry services with high-bandwidth internal network access, elim-  
 103 inating the need for massive local storage. This approach transforms storage requirements  
 104 from a fixed infrastructure cost to an elastic, usage-based model that scales efficiently with  
 105 training demands.
- 106 • **Breaking Computational Throughput Bottlenecks:** We introduce a distributed orches-  
 107 tration system that coordinates thousands of lightweight instances rather than relying on  
 108 high-specification machines. Our many-small-instances approach achieves superior re-

108 source utilization and eliminates the availability constraints that limit traditional centralized  
 109 methods to hundreds of concurrent tasks.  
 110

111 • **System Performance Validation:** We design and implement **MegaFlow**, a three-service  
 112 architecture that enables independent scaling of model serving, agent coordination, and  
 113 environment provisioning. Our evaluation demonstrates 32% cost reduction and consistent  
 114 scaling to tens of thousands of concurrent tasks, with production validation across over 2  
 115 million agent training executions.

## 116 2 MEGAFLOW



140 Figure 2: The architecture of **MegaFlow**. **(bottom)** The *Model Service* provides inference and  
 141 training capabilities through various engines and distributed frameworks. **(middle)** The *Agent Service*  
 142 coordinates execution strategies, integrates with agent frameworks, and manages experience  
 143 feedback loops. **(top)** The *Environment Service* provides containerized execution environments and  
 144 handles distributed task scheduling.

### 145 2.1 SYSTEM OVERVIEW

146 Figure 2 illustrates the overall architecture of MegaFlow’s three-service ecosystem. The system  
 147 operates through three independent services, each with specialized responsibilities:

148 **Model Service** This service handles the computational aspects of agent intelligence, providing  
 149 inference capabilities through various inference engines such as Transformers (Wolf et al., 2020),  
 150 vLLM (Kwon et al., 2023), and SGLang (Zheng et al., 2024), while supporting training operations  
 151 via distributed training frameworks including VeRL (Sheng et al., 2025), FSDP (Zhao et al., 2023),  
 152 and Megatron (Shoeybi et al., 2019). It focuses purely on model computation and parameter updates,  
 153 abstracting away the complexities of agent-environment interactions.

154 **Agent Service** This service acts as the intelligent coordinator that manages agent execution strate-  
 155 gies based on task requirements. It integrates with various agent frameworks such as Open-  
 156 Hands (Wang et al., 2024b), SWE-Agent (Yang et al., 2024a), and Qwen Code (Yang et al., 2025a)  
 157 for different task types (training, evaluation, or data synthesis) and coordinates rollout execution  
 158 across specified datasets. The *Agent Service* processes rollout outputs, aggregating evaluation met-  
 159 rics and feeding experience data back to the *Model Service* for training iterations.

162 **Environment Service** This service represents the most resource-intensive component, responsible  
 163 for the physical execution of agent tasks. It queues tasks in a distributed system and employs sophis-  
 164 ticated scheduling to monitor resource availability and dispatch tasks to cloud compute instances.  
 165 Each instance executes multiple concurrent agent tasks through containerized environments that  
 166 provide isolated execution contexts for agent-environment interactions.

167 **MegaFlow Orchestration** MegaFlow orchestrates the interaction between these three services  
 168 through unified APIs. It manages the complete lifecycle of agent training: from receiving requests  
 169 and provisioning environments, to monitoring progress through event-driven updates, and collecting  
 170 results for downstream processing. The system leverages cloud-native services for elastic compute,  
 171 real-time monitoring, and distributed storage. While our current implementation is built on *Alibaba*  
 172 *Cloud*, the abstracted APIs enable straightforward migration to other major cloud providers such as  
 173 *Amazon Web Services*, *Microsoft Azure*, and *Google Cloud Platform*.

174 This architecture enables: (1) *elastic scaling* through dynamic resource allocation, (2) *fault tolerance*  
 175 via event-driven monitoring, (3) *resource efficiency* through intelligent scheduling, and (4) *service*  
 176 *isolation* allowing independent optimization of each component.

## 178 2.2 KEY DESIGN PRINCIPLES

180 The design of MegaFlow is guided by four key principles that reflect our understanding of large-  
 181 scale agent training requirements and distinguish our approach from traditional distributed systems.

183 **Elastic Resource Strategy** MegaFlow adopts a many-small-instances approach with standardized  
 184 compute configurations, providing superior elasticity and cost optimization compared to few-large-  
 185 instances models. This design aligns with containerized agent workload characteristics and enables  
 186 rapid resource provisioning and deallocation.

187 **Hybrid Execution Model** The system implements dual execution modes: *ephemeral* execution  
 188 for perfect task isolation and *persistent* execution for resource efficiency. This hybrid approach  
 189 optimizes both reliability and resource utilization based on task characteristics.

191 **Event-Driven Coordination** Rather than complex consensus protocols, MegaFlow employs  
 192 event-driven coordination with distributed state management, eliminating polling overhead while  
 193 providing strong consistency guarantees for resource allocation and task scheduling.

194 **Specialized Component Delegation** MegaFlow strategically delegates domain-specific opera-  
 195 tions to specialized systems (agent frameworks for container orchestration, cloud services for storage  
 196 and monitoring), focusing on the unique challenges of agent-environment coordination rather than  
 197 reimplementing general-purpose solutions.

## 199 2.3 ARCHITECTURE DESIGN

201 Based on these design principles, the MegaFlow architecture implements five core components that  
 202 work in concert to provide scalable, fault-tolerant orchestration of agent training workloads.

203 **Task Scheduler** At the heart of MegaFlow lies a high-performance asynchronous scheduler that  
 204 enables massive concurrency for task processing. The system implements a FIFO scheduling policy,  
 205 which proves sufficient for our workloads while maintaining simplicity and predictability.

207 The scheduler intelligently handles two distinct task categories with optimized resource allocation  
 208 strategies. For *Ephemeral Tasks*, the system follows an ephemeral compute model: upon receiving  
 209 a task request, a dedicated compute instance is provisioned, executes the single task, and is imme-  
 210 diately deallocated, eliminating resource contention and providing perfect isolation. For *Persistent*  
 211 *Tasks*, which require sustained execution, the scheduler maintains a pool of persistent compute in-  
 212 stances and employs pool-based allocation, efficiently reusing resources while maintaining isolation  
 213 through containerization.

214 **Resource Manager** The resource management subsystem employs distributed coordination mech-  
 215 anisms to maintain real-time visibility into system state and resource availability. Rather than im-  
 plementing complex resource monitoring and allocation algorithms, our design adopts a uniform

216 resource allocation strategy with standardized compute instances. This standardization simplifies  
 217 scheduling decisions, improves resource predictability, and aligns with containerized workload char-  
 218 acteristics where each instance typically executes a single agent task.

219 The system implements sophisticated concurrency control through a three-tier limiting mechanism:  
 220 (1) User-specified parameters control the rate of *Model Service* API calls, preventing downstream  
 221 bottlenecks; (2) Distributed semaphores ensure that task execution never exceeds available compute  
 222 capacity; and (3) Administrative quotas provide control over resource usage, preventing system  
 223 abuse while enabling fair resource sharing.

224 **Environment Manager** Our environment management strategy demonstrates a key architectural  
 225 insight: by delegating container lifecycle operations to proven open-source agent frameworks,  
 226 MegaFlow focuses on what it does best (orchestration and coordination). The system pre-provisions  
 227 all required container images in cloud registry services, enabling rapid deployment through high-  
 228 bandwidth internal network access.

229 Environment isolation is achieved through a layered approach: each compute instance provides  
 230 resource isolation, while containerization within instances provides process and filesystem isolation.  
 231 This dual-layer isolation ensures that agent operations (including code editing, command execution,  
 232 and file system modifications) remain completely contained within their designated environments.

233 **Event-Driven Monitoring** MegaFlow employs cloud event services to implement reactive sys-  
 234 tem behavior through two critical event streams. Instance lifecycle events enable the system to  
 235 track compute instance state transitions, ensuring tasks are only dispatched to fully operational in-  
 236 stances. Task completion events provide real-time notification of task outcomes, enabling immediate  
 237 resource reclamation and result processing.

238 This event-driven architecture eliminates the need for expensive polling operations while providing  
 239 near-instantaneous response to state changes. The system supplements event notifications with direct  
 240 API calls for detailed task execution information, striking an optimal balance between real-time  
 241 responsiveness and comprehensive monitoring.

242 **Data Persistence** The system architecture separates concerns between operational data and result  
 243 artifacts through specialized storage systems. Operational metadata (including task specifications,  
 244 execution state, and compute instance information) is managed through document databases with  
 245 schema validation and type safety. Task queues are implemented using in-memory storage systems,  
 246 leveraging high-performance operations for rapid task dispatch.

247 Agent execution artifacts are persisted to cloud object storage, providing durable, scalable storage  
 248 for trajectory data, evaluation results, and training artifacts. This separation allows the Agent Service  
 249 to retrieve results asynchronously while maintaining system responsiveness during peak execution  
 250 periods.

251

### 252 3 EVALUATION

253

254 We evaluate MegaFlow’s performance on large-scale complex agent training tasks, focusing on  
 255 multi-step software engineering scenarios that require containerized environments and sustained  
 256 agent-environment interactions. These tasks present unique infrastructure challenges due to their  
 257 need for sophisticated orchestration at concurrent execution scale. Since no existing infrastructure  
 258 provides comparable functionality for such agent training orchestration, our evaluation compares  
 259 MegaFlow against traditional high-specification centralized approaches and analyzes system per-  
 260 formance characteristics.

261

#### 262 3.1 EXPERIMENTAL SETUP

263 **Task Definition and Datasets** We evaluate MegaFlow using software engineering agent training  
 264 tasks that require containerized environments and sustained agent-environment interactions. Our  
 265 evaluation leverages large-scale software engineering datasets (Jimenez et al., 2023; Pan et al., 2024;  
 266 Zhang et al., 2025a; Yang et al., 2025b; Zan et al., 2025; Zhang et al., 2025b), conducting experi-  
 267 ments with workloads scaling up to tens of thousands of concurrent tasks to demonstrate system  
 268 performance at scale.

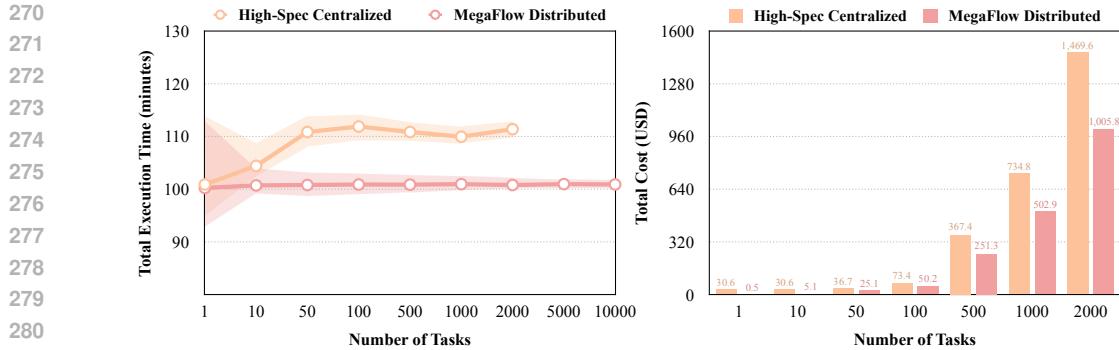


Figure 3: Throughput scaling and cost comparison between MegaFlow and centralized approaches. **(Left)** Total execution time showing MegaFlow’s consistent performance versus centralized degradation. **(Right)** Total cost comparison with 32% cost reduction at 2,000 concurrent tasks. Data represents bootstrap sampling from over 130,000 production records.

**Agent Framework Compatibility** MegaFlow supports major agent frameworks including SWE-Agent (Yang et al., 2024a), OpenHands (Wang et al., 2024b), Mini-SWE-Agent (Yang et al., 2024a), Qwen Code (Yang et al., 2025a), and Claude Code (Anthropic, 2025) across all evaluated benchmark suites, validating our architecture’s generalizability and broad compatibility with existing tools.

**Baseline Configurations** Since no comparable infrastructure exists for large-scale agent training orchestration, we establish baselines through systematic comparison of execution strategies:

- **High-Spec Centralized:** High-specification machines (208-core CPU, 3TB memory, 1 Gbps network bandwidth) with maximum sustainable parallelism of 50 concurrent tasks per instance.
- **MegaFlow Distributed:** Standardized 8-core, 16GB instances (100 Mbps network bandwidth each) with dynamic elastic scaling, where each instance handles 1 concurrent task.

**Data Collection and Analysis** Our evaluation is based on production deployment records comprising over 130,000 ephemeral execution tasks and over 2 million persistent execution tasks. Experiments utilized up to 40 high-specification instances for centralized approaches and up to 10,000 standardized instances for distributed approaches. Performance metrics are computed using bootstrap sampling (100 iterations per data point) with 95% confidence intervals. All experiments were conducted on *Alibaba Cloud*. Unless otherwise stated, we use `ecs.re6.52xlarge` instances for high-specification centralized approaches and `ecs.c8a.2xlarge`, `ecs.c8i.2xlarge` instances for distributed approaches.

### 3.2 THROUGHPUT AND SCALABILITY ANALYSIS

We evaluate MegaFlow’s scalability by measuring system performance across varying workload sizes, examining both throughput and latency characteristics under different concurrency levels.

**Performance and Scalability** Figure 3 demonstrates MegaFlow’s superior characteristics compared to traditional centralized approaches. MegaFlow maintains consistent execution times of approximately 100 minutes across 1 to 10,000 tasks, while high-specification centralized methods exhibit degradation from 100 to 110 minutes due to resource contention bottlenecks. Centralized approaches suffer from network bandwidth congestion during container image pulls and resource competition during initialization. MegaFlow’s distributed architecture eliminates these bottlenecks by providing dedicated resources per task.

The centralized approach faces fundamental scalability constraints, limited to 2,000 concurrent tasks due to instance availability (40 high-specification instances maximum). MegaFlow’s standardized instances enabled provisioning up to 10,000 instances, demonstrating superior elastic scaling capabilities.

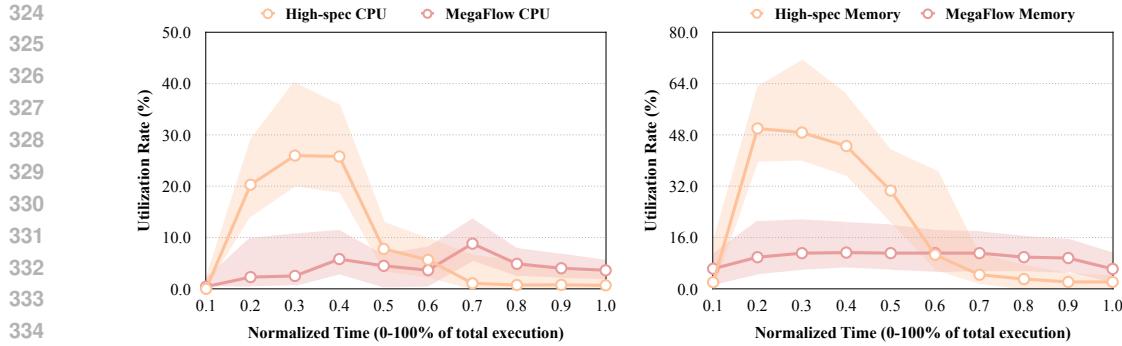


Figure 4: Resource utilization patterns across normalized execution time. **(Left)** CPU utilization: centralized peak at 25% versus MegaFlow’s stable 5-10%. **(Right)** Memory utilization: centralized peak at 50% versus MegaFlow’s consistent 12%. Shaded areas represent 95% confidence intervals.

**Cost Efficiency** At 2,000 tasks, MegaFlow achieves 32% cost reduction (1,005 vs 1,470 USD), with cost advantages increasing at larger scales. Beyond direct savings, MegaFlow eliminates resource availability constraints that prevent traditional methods from scaling to large workloads.

### 3.3 RESOURCE UTILIZATION ANALYSIS

We analyze resource utilization patterns throughout task execution lifecycles to evaluate the efficiency of different architectural approaches. Figure 4 presents CPU and memory utilization rates across normalized execution time for both approaches.

**Utilization Pattern Analysis** The resource utilization patterns reveal fundamental differences between centralized and distributed approaches. High-specification centralized instances exhibit pronounced resource usage spikes, with CPU utilization peaking at 25% during the initial 30% of execution time before declining to near-zero levels. Memory utilization follows a similar pattern, reaching 50% peak usage during mid-execution (20-40% of total time) then dropping sharply.

In contrast, MegaFlow’s distributed architecture maintains consistent resource utilization throughout execution cycles. CPU utilization remains stable at 5-10% across the entire execution period, while memory utilization maintains approximately 12% with minimal variation.

**Resource Efficiency Implications** The contrasting utilization patterns highlight significant efficiency differences. Centralized approaches demonstrate typical “bursty” resource consumption with substantial idle periods, leading to poor overall resource efficiency despite high-specification hardware. The large confidence intervals in centralized approaches indicate high variability in resource demand, making capacity planning challenging.

MegaFlow’s stable utilization patterns with narrow confidence intervals demonstrate predictable resource consumption, enabling more efficient capacity planning and resource allocation. While individual instances operate at lower peak utilization rates, the distributed model achieves better overall resource efficiency through consistent utilization across the execution lifecycle.

### 3.4 END-TO-END LATENCY ANALYSIS

We analyze the complete task execution pipeline to identify performance bottlenecks and validate our hybrid execution model design. Figure 5 presents latency breakdown across different execution phases and environment startup scaling characteristics.

**Latency Breakdown Analysis** The latency decomposition reveals significant differences in execution efficiency across approaches. MegaFlow’s persistent execution mode achieves the lowest total latency at approximately 75 minutes, with minimal infrastructure overhead. The ephemeral execution mode requires approximately 90 minutes total, with additional environment startup costs, while high-specification centralized approaches exhibit the highest latency at 110 minutes due to resource contention across all execution phases.

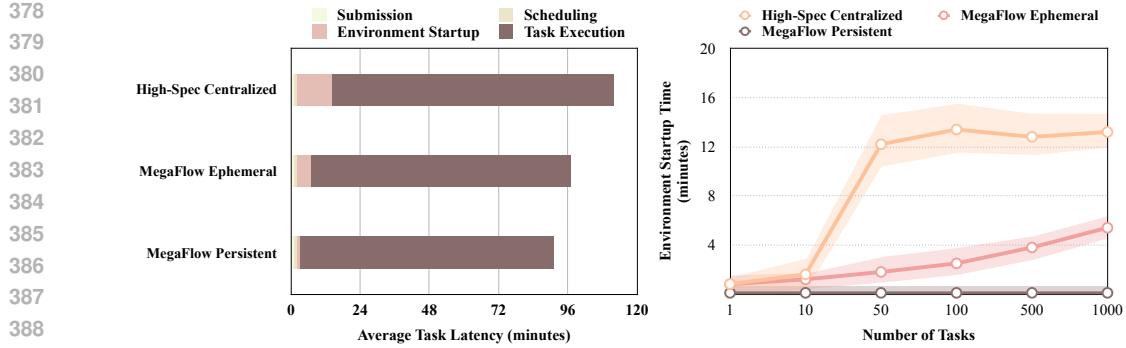


Figure 5: End-to-end latency breakdown and environment startup scaling comparison. **(Left)** Total execution times: MegaFlow Persistent (75 min), Ephemeral (90 min), and High-Spec Centralized (110 min). **(Right)** Environment startup time scaling showing centralized degradation (1-13 min) versus MegaFlow’s stable performance.

Task execution represents the dominant component of total latency across all approaches, but infrastructure overheads vary substantially. Centralized approaches suffer from extended submission, scheduling, and environment startup phases due to resource competition and coordination bottlenecks.

**Environment Startup Scaling** Environment startup times demonstrate critical scalability differences between execution strategies. High-specification centralized approaches exhibit severe startup time degradation, increasing from 1 minute for single tasks to 13 minutes at 1,000 concurrent tasks due to resource contention during container image pulls and initialization. MegaFlow’s ephemeral mode shows modest startup time growth from 1 to 6 minutes, while persistent execution maintains consistently low startup times below 1 minute across all concurrency levels through environment reuse.

The scaling patterns reveal multiple bottleneck sources. The modest increase in MegaFlow’s ephemeral startup times suggests that cloud container registry services experience some performance degradation under high concurrent pull requests, but remain relatively stable. However, the dramatic startup time increase in centralized approaches indicates that the primary bottleneck lies in local resource constraints (network bandwidth limitations and resource contention within high-specification instances) rather than cloud service limitations. This analysis reinforces the effectiveness of MegaFlow’s distributed approach in avoiding local resource bottlenecks through dedicated per-task resource allocation.

**Hybrid Execution Model Validation** These results validate our hybrid execution model design principle. Persistent execution provides optimal performance for sustained workloads through environment reuse, while ephemeral execution offers better isolation guarantees at moderate overhead. The ability to select execution modes based on task characteristics enables MegaFlow to optimize both performance and resource utilization according to specific workload requirements.

### 3.5 DISCUSSION

Our evaluation demonstrates that MegaFlow successfully addresses the scalability challenges of large-scale agent training through distributed orchestration and hybrid execution models. The results validate our core design principles and provide several key insights for agent training infrastructure.

The many-small-instances approach achieves superior cost efficiency (32% reduction) while maintaining consistent performance, contrasting with centralized methods that suffer from resource contention bottlenecks. Resource utilization analysis reveals that stable, predictable consumption patterns enable more efficient capacity planning than bursty high-peak usage, challenging conventional assumptions about resource optimization in agent training systems.

Our bottleneck analysis identifies coordination overhead rather than raw computational power as the primary scalability constraint, with the distinction between cloud service limitations and local

432 resource constraints providing important design guidance. MegaFlow’s broad compatibility across  
 433 agent frameworks validates the practical value of infrastructure-level solutions for the research com-  
 434 munity.

435 Future work should explore orchestration of multi-environment agent tasks with complex service  
 436 dependencies, potentially leveraging container orchestration paradigms like Kubernetes (Kubernetes  
 437 Project, 2014) for dependency management. Additional directions include dynamic execution mode  
 438 switching and multi-cloud deployment strategies to further enhance system flexibility.

## 4 RELATED WORK

440 Our work intersects several research areas, including distributed systems orchestration, container-  
 441 ization technologies, and infrastructure for AI workloads. We briefly review the most relevant prior  
 442 work in each area, with additional comprehensive coverage provided in Appendix B.

443 **Distributed Container Orchestration** Traditional container orchestration systems like Kuber-  
 444 netes (Kubernetes Project, 2014; Verma et al., 2015; Burns et al., 2016), Docker Swarm (Docker,  
 445 2025), and Apache Mesos (Mesos, 2025) focus on general-purpose workload management across  
 446 distributed clusters. While these systems provide powerful abstractions for resource allocation and  
 447 service discovery, they are not optimized for the unique characteristics of agent training workloads,  
 448 such as rapid environment provisioning, heterogeneous execution requirements, and tight integration  
 449 with model serving infrastructure.

450 **Cloud-Native AI Infrastructure** Systems like Kubeflow (Kubeflow Project, 2018), MLflow (Za-  
 451 haria et al., 2018; Chen et al., 2020), and Ray (Moritz et al., 2018) have advanced machine learning  
 452 infrastructure by providing distributed training, model serving, and workflow orchestration capabili-  
 453 ties. However, these systems primarily target traditional ML pipelines rather than interactive agent  
 454 training workloads that require dynamic environment creation, containerized execution contexts,  
 455 and complex agent-environment interaction patterns.

456 **Multi-Agent System Infrastructure** Prior work on multi-agent systems has largely focused on  
 457 coordination algorithms, communication protocols, and simulation environments (Sun et al., 2025).  
 458 While recent efforts have explored infrastructure for LLM-based agents (Tran et al., 2025), these  
 459 works primarily address single-agent scenarios or small-scale interactions. The infrastructure chal-  
 460 lenges of executing thousands of concurrent agent training tasks across distributed environments  
 461 remain largely unaddressed.

462 **Large-Scale AI Training Systems** Distributed training frameworks such as Horovod (Sergeev &  
 463 Del Balso, 2018), FairScale (Facebook Research, 2020), and Megatron-LM (Shoeybi et al., 2019)  
 464 have demonstrated the feasibility of coordinating AI workloads across large clusters. However, these  
 465 systems focus on model training rather than agent execution, and their synchronous, tightly-coupled  
 466 architectures are poorly suited to the asynchronous, loosely-coupled nature of agent-environment  
 467 interactions.

468 Unlike existing approaches, MegaFlow provides a specialized three-service architecture that decou-  
 469 ples model serving, agent coordination, and environment provisioning, enabling independent scaling  
 470 and optimization for large-scale agent training workloads.

## 5 CONCLUSION

471 In this paper, we presented **MegaFlow**, a large-scale distributed orchestration system that addresses  
 472 the fundamental scalability challenges facing agent training infrastructure through a three-service  
 473 architecture that decouples *Model Service*, *Agent Service*, and *Environment Service*. Through  
 474 comprehensive evaluation using over 130,000 production task records, we demonstrated that MegaFlow  
 475 overcomes critical infrastructure bottlenecks, achieving 32% cost reduction and consistent perfor-  
 476 mance scaling to 10,000 concurrent tasks compared to traditional centralized approaches. By estab-  
 477 lishing unified APIs and eliminating orchestration bottlenecks, MegaFlow provides a production-  
 478 ready foundation for large-scale agent training research and enables the development of sophisti-  
 479 cated AI agents at unprecedented scale.

486 REPRODUCIBILITY STATEMENT  
487488 MegaFlow is built using standard cloud-native technologies and evaluated on *Alibaba Cloud* in-  
489 frastructure. The architectural principles and implementation details provided in Section 2.3 are  
490 sufficient for independent implementation. We plan to release the system as open source to facilitate  
491 broader adoption and reproducibility.  
492493 THE USE OF LARGE LANGUAGE MODELS  
494495 Large language models (LLMs) were used solely for language polishing and expression refinement  
496 to improve the clarity and readability of this paper. No LLMs were involved in the research design,  
497 system implementation, data analysis, or generation of technical content and conclusions.  
498499 REFERENCES  
500501 Anthropic. Claude code. <https://claude.com/product/claude-code>, 2025.502 Ibragim Badertdinov, Alexander Golubev, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik,  
503 Andrei Andriushchenko, Maria Trofimova, Daria Litvintseva, and Boris Yangel. Swe-rebench:  
504 An automated pipeline for task collection and decontaminated evaluation of software engineering  
505 agents. *arXiv preprint arXiv:2505.20411*, 2025.506 Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. Borg, omega, and  
507 kubernetes. *Communications of the ACM*, 59(5):50–57, 2016.508 Andrew Chen, Andy Chow, Aaron Davidson, Arjun DCunha, Ali Ghodsi, Sue Ann Hong, Andy  
509 Konwinski, Clemens Mewald, Siddharth Murching, Tomas Nykodym, et al. Developments in  
510 mlflow: A system to accelerate the machine learning lifecycle. In *Proceedings of the fourth*  
511 *international workshop on data management for end-to-end machine learning*, pp. 1–4, 2020.512 Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su.  
513 Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing*  
514 *Systems*, 36:28091–28114, 2023.515 Docker. Docker swarm. <https://docs.docker.com/engine/swarm/>, 2025.516 Ali Dorri, Salil S Kanhere, and Raja Jurdak. Multi-agent systems: A survey. *Ieee Access*, 6:28573–  
517 28593, 2018.518 Facebook Research. Fairscale: A general purpose modular pytorch library for high performance and  
519 large scale training. <https://github.com/facebookresearch/fairscale>, 2020.  
520 Accessed: 2025-01-01.521 Chen Gao, Xiaochong Lan, Nian Li, Yuan Yuan, Jingtao Ding, Zhilun Zhou, Fengli Xu, and Yong  
522 Li. Large language models empowered agent-based modeling and simulation: A survey and  
523 perspectives. *Humanities and Social Sciences Communications*, 11(1):1–24, 2024.524 Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik  
525 Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint*  
526 *arXiv:2310.06770*, 2023.527 Kubeflow Project. Kubeflow: Machine learning toolkit for kubernetes. <https://www.kubeflow.org>, 2018.528 Kubernetes Project. Kubernetes, 2014. URL <http://kubernetes.io>.529 Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph  
530 Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model  
531 serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.532 Apache Mesos. Apache mesos. <https://mesos.apache.org>, 2025.

540 Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia:  
 541 a benchmark for general ai assistants. In *The Twelfth International Conference on Learning*  
 542 *Representations*, 2023.

543

544 Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang,  
 545 Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. Ray: A distributed frame-  
 546 work for emerging {AI} applications. In *13th USENIX symposium on operating systems design*  
 547 *and implementation (OSDI 18)*, pp. 561–577, 2018.

548

549 Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe  
 550 Zhang. Training software engineering agents and verifiers with swe-gym. *arXiv preprint*  
 551 *arXiv:2412.21139*, 2024.

552

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

557

558 Alibaba Qwen Team. Qwen-agent. <https://github.com/QwenLM/Qwen-Agent>, 2025.

559

560 Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensor-  
 561 flow. *arXiv preprint arXiv:1802.05799*, 2018.

562

563 Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng,  
 564 Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlf framework. In *Proceedings*  
 565 *of the Twentieth European Conference on Computer Systems*, pp. 1279–1297, 2025.

566

567 Mohammad Shoeybi, Mostafa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan  
 568 Catanzaro. Megatron-lm: Training multi-billion parameter language models using model par-  
 569 allelism. *arXiv preprint arXiv:1909.08053*, 2019.

570

571 Lijun Sun, Yijun Yang, Qiqi Duan, Yuhui Shi, Chao Lyu, Yu-Cheng Chang, Chin-Teng Lin, and  
 572 Yang Shen. Multi-agent coordination across diverse applications: A survey. *arXiv preprint*  
 573 *arXiv:2502.14743*, 2025.

574

575 The Terminal-Bench Team. Terminal-bench: A benchmark for ai agents in terminal environments,  
 576 Apr 2025. URL <https://github.com/laude-institute/terminal-bench>.

577

578 Khanh-Tung Tran, Dung Dao, Minh-Duong Nguyen, Quoc-Viet Pham, Barry O’Sullivan, and  
 579 Hoang D Nguyen. Multi-agent collaboration mechanisms: A survey of llms. *arXiv preprint*  
 580 *arXiv:2501.06322*, 2025.

581

582 Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John  
 583 Wilkes. Large-scale cluster management at google with borg. In *Proceedings of the tenth euro-  
 584 pean conference on computer systems*, pp. 1–17, 2015.

585

586 Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai  
 587 Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents.  
 588 *Frontiers of Computer Science*, 18(6):186345, 2024a.

589

590 Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan,  
 591 Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software  
 592 developers as generalist agents. *arXiv preprint arXiv:2407.16741*, 2024b.

593

594 Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi,  
 595 Pierrick Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art  
 596 natural language processing. In *Proceedings of the 2020 conference on empirical methods in*  
 597 *natural language processing: system demonstrations*, pp. 38–45, 2020.

598

599 Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe  
 600 Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents:  
 601 A survey. *Science China Information Sciences*, 68(2):121101, 2025.

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

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

602 John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan,  
 603 and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering.  
 604 *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024a.

605 John Yang, Carlos E Jimenez, Alex L Zhang, Kilian Lieret, Joyce Yang, Xindi Wu, Ori Press,  
 606 Niklas Muennighoff, Gabriel Synnaeve, Karthik R Narasimhan, et al. Swe-bench multimodal:  
 607 Do ai systems generalize to visual software domains? *arXiv preprint arXiv:2410.03859*, 2024b.

609 John Yang, Kilian Lieret, Carlos E Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang,  
 610 Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. Swe-smith: Scaling data for software  
 611 engineering agents. *arXiv preprint arXiv:2504.21798*, 2025b.

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

615 Matei Zaharia, Andrew Chen, Aaron Davidson, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Sid-  
 616 dharth Murching, Tomas Nykodym, Paul Oggilvie, Mani Parkhe, et al. Accelerating the machine  
 617 learning lifecycle with mlflow. *IEEE Data Eng. Bull.*, 41(4):39–45, 2018.

618 Daoguang Zan, Zhirong Huang, Wei Liu, Hanwu Chen, Linhao Zhang, Shulin Xin, Lu Chen, Qi Liu,  
 619 Xiaojian Zhong, Aoyan Li, et al. Multi-swe-bench: A multilingual benchmark for issue resolving.  
 620 *arXiv preprint arXiv:2504.02605*, 2025.

622 Lei Zhang, Jiaxi Yang, Min Yang, Jian Yang, Mouxiang Chen, Jiajun Zhang, Zeyu Cui, Binyuan  
 623 Hui, and Junyang Lin. Swe-flow: Synthesizing software engineering data in a test-driven manner.  
 624 *arXiv preprint arXiv:2506.09003*, 2025a.

625 Linghao Zhang, Shilin He, Chaoyun Zhang, Yu Kang, Bowen Li, Chengxing Xie, Junhao Wang,  
 626 Maoquan Wang, Yufan Huang, Shengyu Fu, et al. Swe-bench goes live! *arXiv preprint  
 627 arXiv:2505.23419*, 2025b.

628 Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright,  
 629 Hamid Shojanazeri, Myle Ott, Sam Shleifer, et al. Pytorch fsdp: experiences on scaling fully  
 630 sharded data parallel. *arXiv preprint arXiv:2304.11277*, 2023.

632 Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang,  
 633 Yuqiong Liu, Rui Men, An Yang, et al. Group sequence policy optimization. *arXiv preprint  
 634 arXiv:2507.18071*, 2025.

635 Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi  
 636 Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sqlang: Efficient execution of  
 637 structured language model programs. *Advances in neural information processing systems*, 37:  
 638 62557–62583, 2024.

639 Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng,  
 640 Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for build-  
 641 ing autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

643  
 644  
 645  
 646  
 647

648  
649  

## A DEFINITIONS

650  
651  
652  
This section provides formal definitions for the key concepts underlying MegaFlow’s architecture  
and the agent training tasks it supports.653  
**Definition A.1: Three-Service Architecture**654  
655  
An agentic system is decomposed into three modular services with well-defined interfaces:  
656  
**Model Service** ( $\mathcal{M}$ ): Provides policy inference and training capabilities.

657  
658  
Inference:  $\mathcal{M}_{\text{infer}} : \mathcal{S} \times \Theta \rightarrow \Pi(A)$  (1)

659  
660  
Training:  $\mathcal{M}_{\text{train}} : \mathcal{D} \times \Theta \rightarrow \Theta'$  (2)

661  
662  
where  $\mathcal{S}$  is the state space,  $\Theta$  represents model parameters,  $\Pi(A)$  is a policy distribution over  
663  
actions, and  $\mathcal{D}$  is training data.664  
**Agent Service** ( $\mathcal{A}$ ): Manages agent coordination and experience collection.

665  
666  
$$\mathcal{A} : \mathcal{T} \times \mathcal{M} \rightarrow \mathcal{D} \times \mathcal{R}$$
 (3)

667  
668  
where  $\mathcal{T}$  is an agent task specification, and the output includes collected experiences  $\mathcal{D}$  and  
669  
execution results  $\mathcal{R}$ .670  
**Environment Service** ( $\mathcal{E}$ ): Provides interactive environments and feedback signals.

671  
672  
$$\mathcal{E} : \mathcal{E}_{\text{spec}} \times \mathcal{A} \rightarrow \mathcal{S}' \times \mathcal{R}$$
 (4)

673  
674  
where  $\mathcal{E}_{\text{spec}}$  is the environment specification,  $\mathcal{A}$  is an action, and the output includes the next  
675  
state  $\mathcal{S}'$  and reward signal  $\mathcal{R}$ .676  
677  
To support large-scale execution of agent tasks across distributed infrastructure, MegaFlow decom-  
678  
poses the traditional monolithic agent system into three specialized services. This architectural sep-  
679  
aration enables independent scaling and optimization of each component while maintaining clear  
680  
interfaces for coordination.681  
**Definition A.2: Agent Task**682  
683  
An *Agent Task* is an interactive problem-solving environment defined as a six-tuple

684  
685  
$$\mathcal{T} = (\mathcal{E}, \mathcal{D}, \mathcal{G}, \mathcal{S}, \mathcal{A}, \mathcal{T}),$$

686  
687  
where:688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
•  $\mathcal{E}$  is the environment specification that defines the interactive context in which the agent  
operates, including containerized execution environments, software repositories, or sim-  
ulation parameters;  
•  $\mathcal{D}$  is the task description that provides natural language instructions, problem state-  
ments, or objectives that guide the agent’s behavior;  
•  $\mathcal{G}$  is the goal and evaluation criteria that specify success conditions, reward functions,  
or metrics used to assess task completion and performance;  
•  $\mathcal{S}$  is the state space representing all possible configurations of the environment and agent  
context during task execution;  
•  $\mathcal{A}$  is the action space defining the set of permissible operations the agent can perform  
within the environment;  
•  $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$  is the transition function that specifies how the task state evolves  
given an agent action  $a \in \mathcal{A}$ , capturing the deterministic or stochastic dynamics of the  
environment.702  
703  
704  
705  
706  
707  
708  
709  
710  
An agent task execution produces a trajectory  $\tau = \{(s_0, a_0), (s_1, a_1), \dots, (s_T, a_T)\}$  where  
711  
 $s_t \in \mathcal{S}$  and  $a_t \in \mathcal{A}$ . Task termination is controlled by the agent framework, either through  
712  
explicit completion decisions or upon reaching the maximum step limit. During execution, the  
713  
environment processes each action to generate state transitions, while the final reward signal  
714  
 $R = \mathcal{G}(\tau)$  is computed only upon task completion using the complete trajectory.

702

703

Table 1: Compatibility matrix of agent frameworks with software engineering datasets

704

705

| Dataset                | SWE-Agent | OpenHands | Mini-SWE-Agent | Qwen Code | Claude Code |
|------------------------|-----------|-----------|----------------|-----------|-------------|
| SWE-bench              | ✓         | ✓         | ✓              | ✓         | ✓           |
| SWE-Gym                | ✓         | ✓         | ✓              | ✓         | ✓           |
| SWE-rebench            | ✓         | ✓         | ✓              | ✓         | ✓           |
| SWE-smith              | ✓         | ✓         | ✓              | ✓         | ✓           |
| SWE-bench Multilingual | ✓         | ✓         | ✓              | ✓         | ✓           |
| Multi-SWE-Bench        | ✓         | ✓         | ✓              | ✓         | ✓           |
| Multi-SWE-RL           | ✓         | ✓         | ✓              | ✓         | ✓           |
| SWE-bench-live         | ✓         | ✓         | ✓              | ✓         | ✓           |
| SWE-Flow               | ✓         | ✓         | ✓              | ✓         | ✓           |
| Terminal-bench         | ✓         | ✓         | ✓              | ✓         | ✓           |

715

716

## B ADDITIONAL RELATED WORK

717

718

This section provides comprehensive coverage of related work in software engineering agent systems, computer use automation, and benchmark datasets that complement the core infrastructure focus of our main paper.

722

723

**Software Engineering Agent Systems** The field of software engineering automation has seen significant advancement with the development of specialized agent frameworks. SWE-Agent (Yang et al., 2024a) pioneered the application of language models to software engineering tasks through interactive code editing and testing. OpenHands (Wang et al., 2024b) provides a comprehensive platform for building generalist AI agents that can write code, interact with command lines, and browse the web, with support for multi-agent coordination and extensive evaluation benchmarks. More recent efforts include Qwen-Agent (Qwen Team, 2025), which focuses on Chinese language support, and specialized frameworks like Cursor and GitHub Copilot that target specific development workflows. While these frameworks demonstrate the potential of AI-assisted software development, they primarily operate at small scales and lack the infrastructure support for large-scale distributed training and evaluation.

733

734

**Software Engineering Benchmarks and Datasets** The development of standardized benchmarks has been crucial for advancing software engineering AI. SWE-bench (Jimenez et al., 2023) established the foundation with real-world GitHub issues and corresponding fixes. SWE-gym (Pan et al., 2024) extended this with interactive environments for iterative development. Recent efforts have expanded coverage with SWE-bench Multimodal (Yang et al., 2024b), SWE-bench Multilingual (Yang et al., 2025b), Multi-SWE-bench (Zan et al., 2025), and language-specific variants. Terminal-bench (Team, 2025) evaluates command-line interactions, while Tau-bench (Yao et al., 2024) focuses on tool use capabilities essential for software development workflows. These datasets provide the foundation for systematic evaluation but require sophisticated infrastructure for large-scale concurrent execution.

743

744

**Computer Use and Browser Automation** Beyond software engineering, agent systems have expanded to general computer use automation. WebArena (Zhou et al., 2023) introduced realistic web-based task environments for agent evaluation. Mind2Web (Deng et al., 2023) focuses on web navigation and interaction tasks. OSWorld (Xie et al., 2024) provides comprehensive operating system interaction benchmarks. Recent work on computer use agents includes Anthropic’s computer use capabilities and similar efforts from other organizations. Browser automation frameworks like Playwright and Selenium provide the underlying execution capabilities, but lack the orchestration infrastructure for large-scale agent training scenarios.

751

752

753

## C AGENT FRAMEWORK COMPATIBILITY

754

755

Table 1 demonstrates MegaFlow’s comprehensive compatibility across major agent frameworks and software engineering datasets. The system’s unified API abstraction enables seamless integration

756  
 757 Table 2: Overview of the RL training environments before and after filtering. Instances with pass  
 758 rate equal to 1 (very easy) or 0 (very difficult) are removed to stabilize large-scale rollouts.

| 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 10010 10011 10012 10013 10014 10015 10016 10017 10018 10019 10020 10021 10022 10023 10024 10025 10026 10027 10028 10029 10030 10031 10032 10033 10034 10035 10036 10037 10038 10039 10040 10041 10042 10043 10044 10045 10046 10047 10048 10049 10050 10051 10052 10053 10054 10055 10056 10057 10058 10059 10060 10061 10062 10063 10064 10065 10066 10067 10068 10069 10070 10071 10072 10073 10074 10075 10076 10077 10078 10079 10080 10081 10082 10083 10084 10085 10086 10087 10088 10089 10090 10091 10092 10093 10094 10095 10096 10097 10098 10099 100100 100101 100102 100103 100104 100105 100106 100107 100108 100109 100110 100111 100112 100113 100114 100115 100116 100117 100118 100119 100120 100121 100122 100123 100124 100125 100126 100127 100128 100129 100130 100131 100132 100133 100134 100135 100136 100137 100138 100139 100140 100141 100142 100143 100144 100145 100146 100147 100148 100149 100150 100151 100152 100153 100154 100155 100156 100157 100158 100159 100160 100161 100162 100163 100164 100165 100166 100167 100168 100169 100170 100171 100172 100173 100174 100175 100176 100177 100178 100179 100180 100181 100182 100183 100184 100185 100186 100187 100188 100189 100190 100191 100192 100193 100194 100195 100196 100197 100198 100199 100200 100201 100202 100203 100204 100205 100206 100207 100208 100209 100210 100211 100212 100213 100214 100215 100216 100217 100218 100219 100220 100221 100222 100223 100224 100225 100226 100227 100228 100229 100230 100231 100232 100233 100234 100235 100236 100237 100238 100239 100240 100241 100242 100243 100244 100245 100246 100247 100248 100249 100250 100251 100252 100253 100254 100255 100256 100257 100258 100259 100260 100261 100262 100263 100264 100265 100266 100267 100268 100269 100270 100271 100272 100273 100274 100275 100276 100277 100278 100279 100280 100281 100282 100283 100284 100285 100286 100287 100288 100289 100290 100291 100292 100293 100294 100295 100296 100297 100298 100299 100300 100301 100302 100303 100304 100305 100306 100307 100308 100309 100310 100311 100312 100313 100314 100315 100316 100317 100318 100319 100320 100321 100322 100323 100324 100325 100326 100327 100328 100329 100330 100331 100332 100333 100334 100335 100336 100337 100338 100339 100340 100341 100342 100343 100344 100345 100346 100347 100348 100349 100350 100351 100352 100353 100354 100355 100356 100357 100358 100359 100360 100361 100362 100363 100364 100365 100366 100367 100368 100369 100370 100371 100372 100373 100374 100375 100376 100377 100378 100379 100380 100381 100382 100383 100384 100385 100386 100387 100388 100389 100390 100391 100392 100393 100394 100395 100396 100397 100398 100399 100400 100401 100402 100403 100404 100405 100406 100407 100408 100409 100410 100411 100412 100413 100414 100415 100416 100417 100418 100419 100420 100421 100422 100423 100424 100425 100426 100427 100428 100429 100430 100431 100432 100433 100434 100435 100436 100437 100438 100439 100440 100441 100442 100443 100444 100445 100446 100447 100448 100449 100450 100451 100452 100453 100454 100455 100456 100457 100458 100459 100460 100461 100462 100463 100464 100465 100466 100467 100468 100469 100470 100471 100472 100473 100474 100475 100476 100477 100478 100479 100480 100481 100482 100483 100484 100485 100486 100487 100488 100489 100490 100491 100492 100493 100494 100495 100496 100497 100498 100499 100500 100501 100502 100503 100504 100505 100506 100507 100508 100509 100510 100511 100512 100513 100514 100515 100516 100517 100518 100519 100520 100521 100522 100523 100524 100525 100526 100527 100528 100529 100530 100531 100532 100533 100534 100535 100536 100537 100538 100539 100540 100541 100542 100543 100544 100545 100546 100547 100548 100549 100550 100551 100552 100553 100554 100555 100556 100557 100558 100559 100560 100561 100562 100563 100564 100565 100566 100567 100568 100569 100570 100571 100572 100573 100574 100575 100576 100577 100578 100579 100580 100581 100582 100583 100584 100585 100586 100587 100588 100589 100590 100591 100592 100593 100594 100595 100596 100597 100598 100599 100600 100601 100602 100603 100604 100605 100606 100607 100608 100609 100610 100611 100612 100613 100614 100615 100616 100617 100618 100619 100620 100621 100622 100623 100624 100625 100626 100627 100628 100629 100630 100631 100632 100633 100634 100635 100636 100637 100638 100639 100640 100641 100642 100643 100644 100645 100646 100647 100648 100649 100650 100651 100652 100653 100654 100655 100656 100657 100658 100659 100660 100661 100662 100663 100664 100665 100666 100667 100668 100669 100670 100671 100672 100673 100674 100675 100676 100677 100678 100679 100680 100681 100682 100683 100684 100685 100686 100687 100688 100689 100690 100691 100692 100693 100694 100695 100696 100697 100698 100699 100700 100701 100702 100703 100704 100705 100706 100707 100708 100709 100710 100711 100712 100713 100714 100715 100716 100717 100718 100719 100720 100721 100722 100723 100724 100725 100726 100727 100728 100729 100730 100731 100732 100733 100734 100735 100736 100737 100738 100739 100740 100741 100742 100743 100744 100745 100746 100747 100748 100749 100750 100751 100752 100753 100754 100755 100756 100757 100758 100759 100760 100761 100762 100763 100764 100765 100766 100767 100768 100769 100770 100771 100772 100773 100774 100775 100776 100777 100778 100779 100780 100781 100782 100783 100784 100785 100786 100787 100788 100789 100790 100791 100792 100793 100794 100795 100796 100797 100798 100799 100800 100801 100802 100803 100804 100805 100806 100807 100808 100809 100810 100811 100812 100813 100814 100815 100816 100817 100818 100819 100820 100821 100822 100823 100824 100825 100826 100827 100828 100829 100830 100831 100832 100833 100834 100835 100836 100837 100838 100839 100840 100841 100842 100843 100844 100845 100846 100847 100848 100849 100850 100851 100852 100853 100854 100855 100856 100857 100858 100859 100860 100861 100862 100863 100864 100865 100866 100867 100868 100869 100870 100871 100872 100873 100874 100875 100876 100877 100878 100879 100880 100881 100882 100883 100884 100885 100886 100887 100888 100889 100890 100891 100892 100893 100894 100895 100896 100897 100898 100899 100900 100901 100902 100903 100904 100905 100906 100907 100908 100909 100910 100911 100912 100913 100914 100915 100916 100917 100918 100919 100920 100921 100922 100923 100924 100925 100926 100927 100928 100929 100930 100931 100932 100933 100934 100935 100936 100937 100938 100939 100940 100941 100942 100943 100944 100945 100946 100947 100948 100949 100950 100951 100952 100953 100954 100955 100956 100957 100958 100959 100960 100961 100962 100963 100964 100965 100966 100967 100968 100969 100970 100971 100972 100973 100974 100975 100976 100977 100978 100979 100980 100981 100982 100983 100984 100985 100986 100987 100988 100989 100990 100991 100992 100993 100994 100995 100996 100997 100998 100999 1001000 1001001 1001002 1001003 1001004 1001005 1001006 1001007 1001008 1001009 10010010 10010011 10010012 10010013 10010014 10010015 10010016 10010017 10010018 10010019 100100100 100100101 100100102 100100103 100100104 100100105 100100106 100100107 100100108 100100109 100100110 100100111 100100112 100100113 100100114 100100115 100100116 100100117 100100118 100100119 100100120 100100121 100100122 100100123 100100124 100100125 100100126 100100127 100100128 100100129 100100130 100100131 100100132 100100133 100100134 100100135 100100136 100100137 100100138 100100139 100100140 100100141 100100142 100100143 100100144 100100145 100100146 100100147 100100148 100100149 100100150 100100151 100100152 100100153 100100154 100100155 100100156 100100157 100100158 100100159 100100160 100100161 100100162 100100163 100100164 100100165 100100166 100100167 100100168 100100169 100100170 100100171 100100172 100100173 100100174 100100175 100100176 100100177 100100178 100100179 100100180 100100181 100100182 100100183 100100184 100100185 100100186 100100187 100100188 100100189 100100190 100100191 100100192 100100193 100100194 100100195 100100196 100100197 100100198 100100199 100100100 100100101 100100102 100100103 100100104 100100105 100100106 100100107 100100108 100100109 100100110 100100111 100100112 100100113 100100114 100100115 100100116 100100117 100100118 100100119 100100120 100100121 100100122 100100123 100100124 100100125 100100126 100100127 100100128 100100129 100100130 100100131 100100132 100100133 100100134 100100135 100100136 100100137 100100138 100100139 100100140 100100141 100100142 100100143 100100144 100100145 100100146 100100147 100100148 100100149 100100150 100100151 100100152 100100153 100100154 100100155 100100156 100100157 100100158 100100159 100100160 100100161 100100162 100100163 100100164 100100165 100100166 100100167 100100168 100100169 100100170 100100171 100100172 100100173 100100174 10010017 |
| --- |

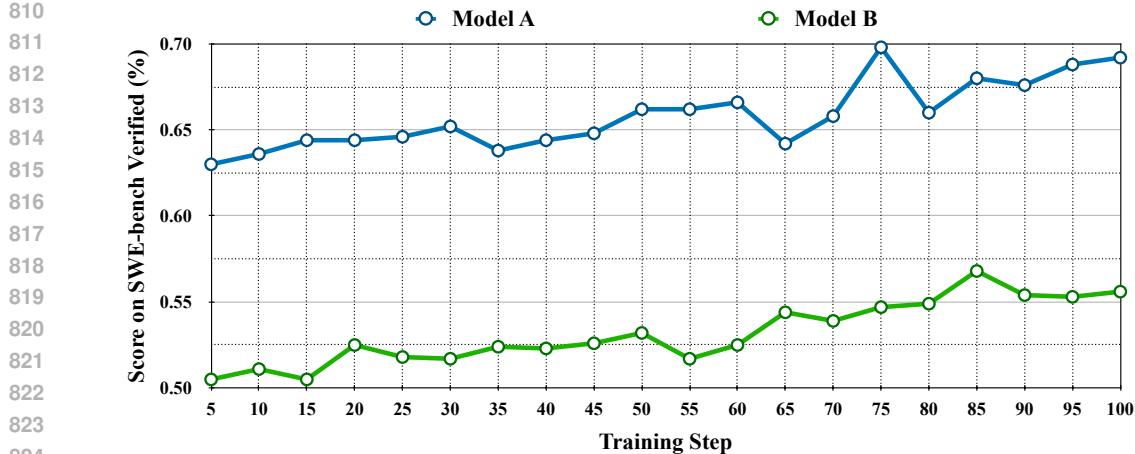


Figure 6: RL training dynamics on SWE-bench Verified. **Model A** is a 235 billion parameters MoE model and **Model B** is a 30 billion parameters MoE model. Scores are evaluated using the OpenHands scaffold across training steps 0–100.

prior attempts. The sampling temperature is set to 1.0, and the maximum response length per turn is limited to 4096 tokens. If the agent does not explicitly terminate the task within 100 rounds (i.e., does not call `finish`/`submit`), a fixed penalty of  $-0.5$  is applied. We optimize the model using a learning rate of  $1e-6$ , with positive and negative reward clipping thresholds set to  $4e-4$  and  $2e-4$  respectively.

## D.2 TRAINING RESULTS

Figure 6 reports the RL training dynamics evaluated on SWE-bench Verified using the OpenHands scaffold. We compare two models of different scales: **Model A**, a 235 billion parameters MoE model, and **Model B**, a 30 billion parameters MoE model. Both models show consistent improvement during RL training, with the larger model achieving substantially higher scores throughout training. These results demonstrate the effectiveness of large-scale agentic RL training for SWE-style tasks and highlight MegaFlow’s ability to sustain stable, fault-tolerant, and high-throughput distributed rollouts across heterogeneous agent frameworks.