Operand Quant: A Single-Agent Architecture for Autonomous Machine Learning Engineering

Operand Research*

2025

Abstract

We present **Operand Quant**, a single-agent, IDE-based architecture for autonomous machine learning engineering (MLE). Operand Quant departs from conventional multi-agent orchestration frameworks by consolidating all MLE lifecycle stages—exploration, modeling, experimentation, and deployment—within a single, context-aware agent. On the MLE-Benchmark (2025), Operand Quant achieved a new state-of-the-art (SOTA) result, with an overall medal rate of **0.3956 $\pm$ 0.0565 across 75 problems—the highest recorded performance among all evaluated systems to date**. The architecture demonstrates that a linear, non-blocking agent, operating autonomously within a controlled IDE environment, can outperform multi-agent and orchestrated systems under identical constraints.

1 Introduction

The automation of the machine learning engineering (MLE) pipeline has become a central objective in agentic AI research. Many recent systems rely on multi-agent orchestration, in which specialized agents independently handle data analysis, modeling, evaluation, and deployment. While such frameworks can parallelize work, they often incur coordination costs, context fragmentation, and synchronization errors.

Operand Quant explores an alternative paradigm: a single autonomous agent that continuously observes, plans, edits, executes, and evaluates within its own integrated development environment (IDE). This design hypothesis asserts that end-to-end contextual continuity can yield reliable and efficient performance without distributed orchestration.

The agent was evaluated on the **MLE-Benchmark** [1], which tests autonomous ML capabilities under isolation (no internet, fixed runtime, standardized hardware). Operand Quant's results establish that a single-agent architecture can achieve top-tier performance on complex machine learning engineering tasks without relying on external retrieval or multi-agent decomposition.

2 System Overview

2.1 Design Principles

Operand Quant is built as a single-agent system embedded within a simulated IDE. The IDE emulates the tools of a human ML engineer—file explorer, notebooks, scripts, execution kernels, and logs—allowing the agent to autonomously perform all phases of the MLE lifecycle: exploratory data analysis, feature engineering, modeling, evaluation, iteration, and productionization.

*Contributors: Ram Gorthi; Arjun Sahney; Javier Vega. All authors are affiliated with Operand's Research Team.

In contrast to orchestrated multi-agent setups, Operand Quant maintains a *unified reasoning state* throughout execution. The same LLM instance plans, codes, executes, and evaluates, eliminating context handoffs and preserving a consistent view of the workspace.

2.2 Non-Blocking Turn-Based Operation

The agent operates in *turns*, each representing one reasoning–execution cycle. During a turn: (i) the agent observes current IDE state (open files, kernel status, active processes, and outputs); (ii) decides an action and emits a single structured JSON command; (iii) the action is validated and executed asynchronously; and (iv) the result is persisted and integrated into history. If an execution process remains active, the agent continues working on other tasks while monitoring intermediate outputs. This non-blocking loop enables parallel reasoning and continuous iteration.

3 Deep-Thinking Ensemble

3.1 Motivation

Large language models (LLMs) exhibit context bias, a degradation of reasoning flexibility as prompt length increases. In long reasoning sessions, models can develop tunnel vision, reducing their ability to debug or reassess prior assumptions. Operand Quant mitigates this limitation via a mechanism called *deep-thinking*.

3.2 Ensemble Reasoning

When the agent encounters a reasoning bottleneck, it delegates the problem to an ensemble of high-capacity models—**GPT-5**, **Claude-4.1 Opus**, **Grok-4**, and **Gemini 2.5 Pro**—that independently generate analyses or hypotheses. Their outputs are synthesized into a consolidated "expert review," reintroduced into the agent's reasoning context as advisory input. From the agent's perspective, this appears as a consultation with domain experts; in practice, it constitutes a structured ensemble reasoning step performed under strict isolation.

3.3 Compliance

All ensemble models operated without internet access, satisfying MLE-Benchmark requirements. Ensemble interactions were limited to local inference; no external retrieval or tool augmentation was used.

4 Architecture Specification

4.1 Core Agent Loop

Each reasoning cycle proceeds as follows: (1) observe IDE and process state; (2) generate a structured JSON action conforming to a validated schema; (3) execute the specified operation; (4) persist results to disk; and (5) trigger compaction if nearing context length limits. A strict *single-tool-per-turn* rule enforces interpretability and deterministic replay.

4.2 Concurrent Execution

Operand Quant supports asynchronous notebook and script execution. When a process is running, it remains under continuous monitoring for status, output, and resource utilization. Example monitoring logic:

```
if primary_notebook and primary_notebook.is_cell_executing():
    continue_result = primary_notebook.continue_execution_if_running()
    if continue_result["status"] == "completed":
        final_output = continue_result.get("output", "[No Output]")
    elif continue_result["status"] == "still_executing":
        current_output = continue_result["current_output"]
        duration = continue_result["execution_duration_seconds"]
```

This enables non-blocking concurrent processing: while training runs execute, the agent continues editing, planning, or analyzing outputs.

4.3 Interruption Logic

The agent often chooses to interrupt execution processes when: (i) convergence is detected from loss or validation metrics; (ii) memory or runtime thresholds are exceeded; or (iii) non-convergent patterns appear in logs or errors. This dynamic interruption mechanism allows efficient allocation of the fixed runtime budget. The agent controls its code and execution processes through its own state management system.

4.4 State Persistence and Compaction

All process metadata, outputs, and dependencies are stored persistently. When cumulative context approaches model limits, Operand Quant performs hierarchical memory compaction: (1) exclude verbose notebook content; (2) summarize older turns using a dedicated summarization utility; (3) validate the summary; and (4) replace the original history upon successful validation. Each compaction prompt and output is saved in `agent_metadata/` for auditability.

4.5 IDE Context Awareness

The agent receives a real-time textual summary of its environment, e.g.:

```
Execution Status:
Cell 3 of model_training.ipynb executing (127 s)
validation_script.py running (45 s)
hyperparameter_search.ipynb idle
```

This enables reasoning about dependencies, parallelism, and resources with fine granularity.

5 Benchmark Governance and Evaluation Protocol

5.1 Compliance

Operand Quant fully complied with MLE-Benchmark 2025 governance: (i) no internet or API access; (ii) tools confined to local environment; and (iii) standardized submission via the `submit_final_answer` endpoint. An earlier alias, `submit_for_scoring`, was present in two logs but non-functional; no run received live score feedback. All runs were verified through manual log inspection.

5.2 Infrastructure

Each run was limited to a 24-hour execution window. The *Lite* subset used a GCP VM (234 GB RAM, 36 vCPUs, Tesla T4). The *Medium/Hard* subsets used Azure NV36AdsA10v5 (official MLE hardware). Hardware transition between providers accounts for timing differences across lite and medium/hard runs.

5.3 Submission Lifecycle

When the agent achieved a medal result, it called `submit_final_answer`, synchronized code and metadata to the host, and terminated the container. Logs were preserved for reproducibility. When the agent calls `submit_final_answer` and does not achieve a medal, if the time limit was not reached, the agent would be provided a generic 'continue improving' message and sent back to the reasoning loop. While this approach is not part of the canonical MLE-Benchmark protocol, the alternative would have been prompting the agent to defer submission until the time limit was reached—which would have resulted in substantially higher computational costs.

6 Experimental Results

6.1 Performance Summary

Operand Quant achieved the following medal rates on MLE-Benchmark 2025:

Table 1: MLE-Benchmark Medal Rates

Subset	**Medal Rate (mean $\pm$ std)**	**Problems (n)**
Overall	**0.3956 $\pm$ 0.0565**	75
Lite	**0.6364 $\pm$ 0.1050**	22
Medium	**0.3333 $\pm$ 0.0765**	38
Hard	**0.2000 $\pm$ 0.1069**	15

6.2 Leaderboard Comparison

Table 2: Leaderboard (2025-09)

Agent	**Lite (%)**	**Medium (%)**	**Hard (%)**	**All (%)**	**Hrs**	**Date**
Operand Quant	**63.64 $\pm$ 10.50**	**33.33 $\pm$ 7.65**	**20.00 $\pm$ 10.69**	**39.56 $\pm$ 5.65**	**24**	**2025-09-28**
InternAgent (DeepSeek-R1)	62.12 $\pm$ 3.03	26.32 $\pm$ 2.63	24.44 $\pm$ 2.22	36.44 $\pm$ 1.18	12	2025-09-12
R&D-Agent (GPT-5)	68.18 $\pm$ 2.62	21.05 $\pm$ 1.52	22.22 $\pm$ 2.22	35.11 $\pm$ 0.44	12	2025-09-26
Neo Multi-Agent	48.48 $\pm$ 1.52	29.82 $\pm$ 2.32	24.44 $\pm$ 2.22	34.22 $\pm$ 0.89	36	2025-07-28
R&D-Agent (o3 + GPT-4.1)	51.52 $\pm$ 4.00	19.30 $\pm$ 3.16	26.67 $\pm$ 0.00	30.22 $\pm$ 0.89	24	2025-08-15

Operand Quant's 24-hour runs yield a higher overall score than all other published agents, including multi-agent architectures with shorter runtimes.

6.3 Incomplete or Failed Problems

The following tasks failed due to data or environment issues and are reported as "no medal" across seeds: 3D Object Detection for Autonomous Vehicles; AI4Code; Billion Word Imputation; BMS Molecular Translation; Google Research Identify Contrails; HMS Harmful Brain Activity Classification; HuBMAP

Kidney Segmentation; Jigsaw Unintended Bias Classification; RSNA-MICCAI Brain Tumor Radiogenomic Classification; Statoil Iceberg Classifier; TensorFlow2 Question Answering. One outlier—Multi-Modal Gesture Recognition—was excluded after identifying a dataset leakage bug that led to an invalid perfect score. The Multi-Modal Gesture Recognition problem contains a data leak that can be exploited by an intelligent agent that reads the ground truth labels. After our agent discovered the data leak, it exploited it to achieve a perfect score. This score was invalidated by internal Operand reviewers. We encourage the MLE-Benchmark team to consider excluding this problem from future competitions.

7 Discussion and Limitations

Operand Quant's results demonstrate that a single-agent, turn-based architecture can outperform orchestrated systems on the MLE-Benchmark. Unified contextual reasoning and deterministic state persistence appear sufficient for competitive performance without distributed coordination. Limitations include: (i) context degradation despite compaction; (ii) one-tool-per-turn expressiveness limits; (iii) high compute cost from 24-hour runs; and (iv) incomplete fault tolerance for environment or kernel errors.

8 Conclusion

Operand Quant establishes a new state-of-the-art in autonomous machine learning engineering. With an overall score of **0.3956** $\pm$ **0.0565**, it currently ranks first on the MLE-Benchmark 2025 leaderboard, surpassing both single- and multi-agent baselines under identical governance conditions. Its success demonstrates that autonomous MLE systems can achieve leading performance using a unified, single-agent architecture grounded in continuous reasoning, concurrent execution, and structured context management. Future work will extend Operand Quant with adaptive ensemble reasoning, dynamic compaction, and fault-tolerant execution.

References

References

[1] OpenAI. MLE-Bench: Evaluating Autonomous Machine Learning Engineering Agents. 2024. arXiv:2410.07095. Available at https://arxiv.org/abs/2410.07095 and https://openai.com/index/mle-bench/.

[2] OpenAI. MLE-bench (GitHub repository). 2024–2025. Available at https://github.com/openai/mle-bench.

[3] Operand Research. Operand Linear-Quant: MLE-Bench Submission Artifacts (GitHub repository). 2025. Available at https://github.com/ramgorthi04/OperandLinear-MLE-Bench.