
The FM Agent

Annan Li*, Chufan Wu*, Zengle Ge*, Yee Hin Chong*, Zhinan Hou*, Lizhe Cao*,
Cheng Ju*, Jianmin Wu*, Huaiming Li, Haobo Zhang, Shenghao Feng,
Mo Zhao, Fengzhi Qiu, Rui Yang, Mengmeng Zhang, Wenyi Zhu,
Yingying Sun, Quan Sun, Shunhao Yan, Danyu Liu, Dawei Yin[†], Dou Shen[†]

FM Agent Team, Baidu AI Cloud

Abstract

Large language models (LLMs) are catalyzing the development of autonomous AI research agents for scientific and engineering discovery. We present FM Agent, a novel and general-purpose multi-agent framework that leverages a synergistic combination of LLM-based reasoning and large-scale evolutionary search to address complex real-world challenges. The core of FM Agent integrates several key innovations: 1) a cold-start initialization phase incorporating expert guidance, 2) a novel evolutionary sampling strategy for iterative optimization, 3) domain-specific evaluators that combine correctness, effectiveness, and LLM-supervised feedback, and 4) a distributed, asynchronous execution infrastructure built on Ray. Demonstrating broad applicability, our system has been evaluated across diverse domains, including operations research, machine learning, GPU kernel optimization, and classical mathematical problems. FM Agent reaches state-of-the-art results autonomously, without human interpretation or tuning — **1976.3** on ALE-Bench (+5.2%), **43.56%** on MLE-Bench (+4.0pp), up to **20×** speedups on KernelBench, and establishes new state-of-the-art(SOTA) results on several classical mathematical problems. Beyond academic benchmarks, FM Agent shows considerable promise for both large-scale enterprise R&D workflows and fundamental scientific research, where it can accelerate innovation, automate complex discovery processes, and deliver substantial engineering and scientific advances with broader societal impact.

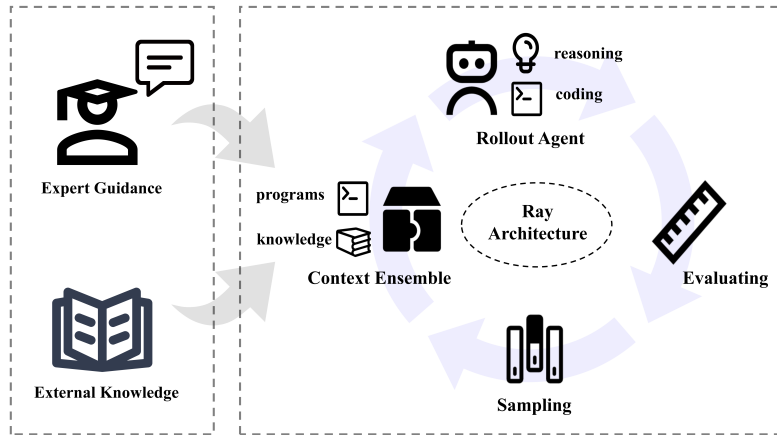


Figure 1: The workflow of FM Agent System to tackle a complex algorithm problem.

*core contributor

[†]project sponsor

1 Introduction

Recent advances in large language models (LLMs) have spurred the development of increasingly capable and autonomous AI research agents. A prominent line of work focuses on orchestrating multiple LLM-driven agents to tackle complex, open-ended discovery and optimization tasks. These systems often employ a search-driven paradigm, where agents explore solution spaces systematically through evolutionary or reinforcement-style loops. Pioneering systems demonstrate how LLMs can generate, mutate, and evaluate large populations of candidate solutions, enabling the discovery of novel and high-performing designs across various domains.

The applicability of such AI research agents extends far beyond academic benchmarks. In industrial settings, numerous high-impact challenges—from combinatorial optimization and time-series forecasting to high-performance kernel tuning—share a common structure: evaluating candidate solutions is relatively straightforward, whereas identifying truly effective ones is exceptionally difficult. Traditionally, progress in these areas has relied on specialized engineers who design specific algorithms and refine them through iterative, project-based optimization. This human-driven process intrinsically mirrors a research cycle: retrieving relevant knowledge, synthesizing ideas from diverse sources, and continuously testing and refining solutions. AI agents that embody large-scale search and evolutionary principles are particularly suited to automating this process, as they can maintain diverse candidate pools, apply intelligent variation operations, and leverage performance feedback to progressively evolve superior solutions.

To effectively harness these capabilities for real-world industrial problems, we propose FM Agent, a novel and general-purpose multi-agent framework. FM Agent is designed to be broadly applicable across domains such as operations research, machine learning, and system optimization. It integrates four key architectural innovations to achieve robust performance and scalability:

- **Cold-Start Initialization.** This phase integrates diverse generation agents to produce a broad yet high-quality initial solution space. Moreover, with an optional expert-in-the-loop design, the framework ensures evolutionary search begins from a pragmatically grounded foundation, significantly accelerating convergence, especially in some real-world complex cases.
- **Adaptive Diversity-Driven Sampling.** Our novel sampling strategy orchestrates multiple parallel evolutionary islands, adaptively balancing exploration and exploitation through dynamic resource allocation. This mechanism maintains productive diversity across algorithmic lineages while systematically steering the population toward global optima.
- **Domain-Specific Evaluation.** Custom evaluators synthesize multiple critical criteria—including functional correctness, operational effectiveness, and LLM-supervised quality assessment—to generate nuanced, multi-faceted feedback. This comprehensive scoring mechanism provides rich, cumulative signals that precisely guide the iterative refinement process.
- **Distributed Asynchronous Infrastructure.** Built on Ray, our scalable orchestration framework enables fine-grained, large-scale concurrent evaluation across distributed computing resources. This architecture ensures efficient resource utilization while facilitating rapid and systematic exploration of complex, high-dimensional solution spaces.

By unifying knowledge-augmented reasoning, autonomous evolution, and domain-aware evaluation within a scalable infrastructure, FM Agent constitutes a general self-improving system. It establishes new state-of-the-art results on authoritative benchmarks: achieving **1976.3** on ALE-Bench[1] (**+5.2%**), **43.56%** on MLE-Bench[2] (**+4.0pp**), and delivering **2.08× to 20.77×** speedups over `torch.compile` on KernelBench[3]. Furthermore, it demonstrates strong performance on classical mathematical problems and matches or surpasses real-world algorithmic practice in various industrial scenarios within and beyond Baidu. We believe FM Agent lays a foundation for a new generation of AI research agents capable of addressing tangible productivity challenges, thereby contributing to technological advancement and broader societal benefit.

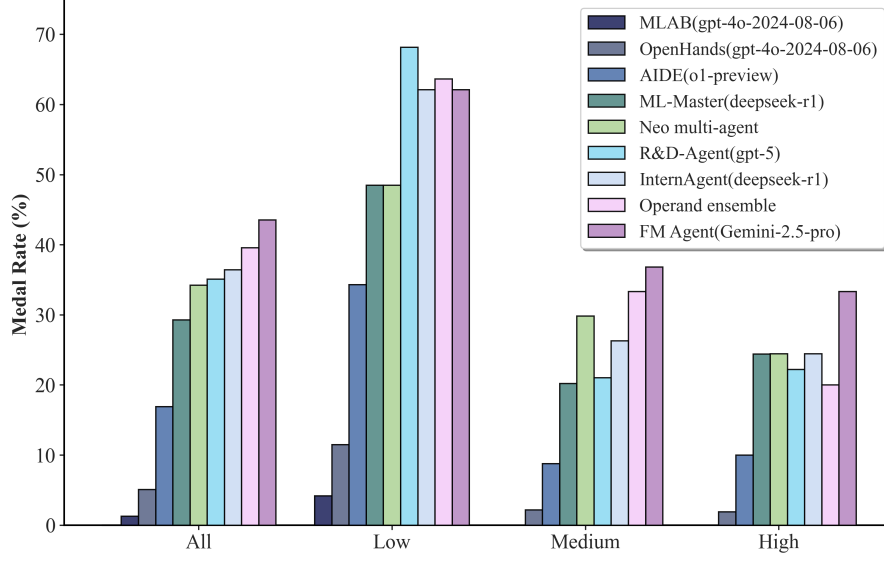


Figure 2: Performance of Agents on MLE-Bench: Medal Rate (%), evaluating FM Agent across real-world machine learning tasks sourced from Kaggle competitions.

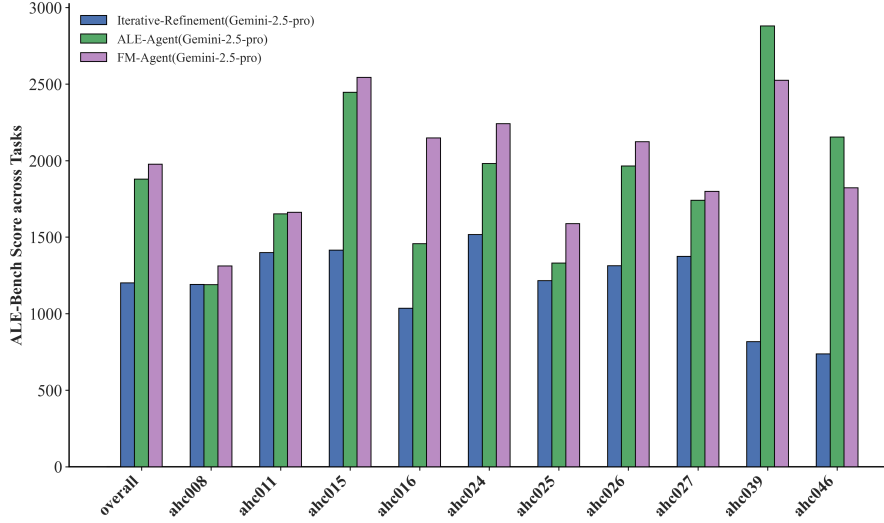


Figure 3: Performance of Agents on the ALE-Bench Lite, denoting the SOTA capability of FM Agent in tackling challenging heuristic-driven tasks from AtCoder Completion.

2 Scenario

2.1 Machine Learning

Machine Learning (ML), which enables computational models to learn patterns from data autonomously, has become fundamental to building intelligent systems, with critical applications including financial risk control [4], time series forecasting [5], and equipment fault diagnosis [6].

However, developing high-performance ML models remains challenging due to the inherent complexity of data and optimization. Engineers often rely on iterative experimentation and object-specific customization in feature and model design, requiring substantial expertise and remains difficult to fully automate.

In this context, FM Agent emerges as a promising solution by integrating large language models with evolutionary computing. Moving beyond conventional automation boundaries that focus primarily on model selection and hyperparameter tuning, FM Agent facilitates autonomous orchestration of the complete ML workflow—from intelligent feature engineering to adaptive model construction. This approach demonstrates the potential to significantly reduce manual intervention while maintaining competitive performance in diverse tasks.

In our practice, FM Agent revolutionizes machine learning workflows through four principal approaches:

- **Autonomous Feature Mining:** Feature engineering is a crucial yet highly expertise-dependent and time-consuming aspect of machine learning. Guided by an evolutionary framework, FM Agent can autonomously explore raw data, surpassing the limitations of traditional statistical methods. Through iterative generation, evaluation, and filtering, it constructs informative new features from raw data, even discovering feature representations that are non-intuitive to human analysts yet possess high predictive power, thereby providing a superior data foundation for subsequent model learning.
- **Intelligent Feature Combination:** The expressive capacity of individual features is limited, while manually creating effective feature interactions becomes exponentially complex with increasing dimensionality. FM Agent serves as an efficient "explorer" that systematically searches the high-dimensional feature combination space. It experiments with various mathematical operations and logical combinations to identify synergistic effects between features that maximize model performance. This process automates the critical step of discovering high-order nonlinear relationships, effectively enhancing the model's representational capability.
- **Adaptive Model Fusion:** Model fusion (ensemble learning), a commonly used and effective technique in modern machine learning, combines multiple base learners to enhance predictive performance and robustness. FM Agent can train new models for different purpose and develop more sophisticated fusion mechanisms beyond simple voting strategies. For example, it can autonomously design weighted ensemble schemes, allocating appropriate weights to different sub-models based on their performance across various data subsets, or construct stacked ensemble models while optimizing the structure and parameters of the meta-learner, ultimately achieving predictive accuracy superior to any single component model.
- **End-to-End Machine Learning Task Solving:** The most forward-looking application involves deploying FM Agent as an end-to-end machine learning system. In this paradigm, the agent receives a dataset and task objectives, then autonomously decides and executes the complete machine learning pipeline, including feature preprocessing, algorithm selection, model structure design, training, and validation. Through sequential decision-making guided by feedback from evolutionary cycles, it progressively builds an optimized machine learning pipeline tailored to specific tasks, significantly advancing the realization of fully automated machine learning.

In summary, these four directions clearly demonstrate FM Agent's evolution from a tool assisting specific tasks to a collaborative partner driving full-process automation. By leveraging its complex reasoning capabilities and integrating them with the systematic exploration of evolutionary frameworks, FM Agent has the potential to redefine the development efficiency and performance limits of machine learning solutions.

2.2 Combinatorial Optimization

Combinatorial optimization (CO), defined as the selection of optimal objects from finite solution sets, represents a foundational paradigm in operations research. This methodology formulates critical real-world applications including production scheduling [7], logistics transportation [8], and resource management [9], where solving efficiency and solution quality directly translate to substantial economic value.

However, the NP-hard nature of CO problems renders exact solutions computationally prohibitive or intractable in practice. Consequently, significant research efforts focus on accelerating CO

solving while preserving solution quality. Yet human-designed strategies often demand extensive domain expertise and costly trial-and-error processes [10]. FM Agent opens up a more transformative research frontier where agents actively participate in the discovery and refinement of new optimization strategies. Specifically, FM Agent is adopted into three principal avenues through which combinatorial optimization could be revolutionized:

- **Autonomous Design of Novel End-to-End Heuristics:** Heuristics are essential tools for tackling NP-hard CO problems, providing efficient, near-optimal solutions where exact methods fail. Guided by an evolutionary framework, FM Agent can explore the vast design space of heuristic algorithms. By iteratively generating, testing, and refining algorithmic components based on performance feedback, FM Agent can autonomously discover novel and powerful heuristics tailored to specific problem structures. This approach emulates the human process of innovation but at a scale and speed previously unattainable, potentially yielding strategies that diverge significantly from conventional designs.
- **Intelligent Augmentation of Optimization Solvers:** Rather than replacing traditional solvers, FM Agent can serve as specialized collaborators to enhance their performance. Modern solvers for problems like mixed integer programming or constraint programming are modular systems that can be improved by integrating high-quality, problem-specific components. FM agent could be set to design these critical modules. For instance, it could help generate cutting planes to tighten linear relaxations, design presolve policy to accelerate solution process.
- **Direct and Iterative Solution Generation:** Another significant application involves employing the FM Agent as a direct, end-to-end solver that iteratively constructs a high-quality solution. In this paradigm, FM Agent engages in a sequential decision-making process, where each step involves selecting a component of the solution. This method bypasses the need for an explicit intermediate algorithmic representation and instead enables the agent to develop a deep, implicit understanding of the problem’s structure, reasoning its way directly to an optimal or near-optimal solution.

2.3 Kernel Generation

The explosive growth of deep neural networks (DNNs), particularly large-scale models such as large language models (LLMs), has established GPUs as the dominant platform for AI workloads. At the foundation of this stack lie CUDA kernels for DNN operators, which directly determine how effectively parallelism and memory hierarchies are utilized. Their efficiency is critical for overall system performance [11], while inefficient kernels can significantly degrade model throughput.

Designing high-performance CUDA kernels is notoriously difficult due to the *complex interactions* among memory access patterns, thread block configurations, and instruction scheduling [12]. Manual tuning is often a labor-intensive trial-and-error process. Although AI compilers [13–15] and domain-specific languages (DSLs) [16–18] provide automation for common operators, their reliance on predefined schedules and rigid transformation rules limits generalization to new workloads and prevents full exploitation of hardware-specific opportunities. This leaves a persistent performance gap.

Instead of treating kernel optimization as a human-driven, knowledge-intensive task, FM Agent reformulate it as an autonomous, data-driven process. Advances in LLMs have made scalable code generation possible [19–21], but training them for high-quality performance is prohibitively expensive. FM Agent circumvents this by generating diverse candidates, evaluating their runtime performance, and using feedback to guide further exploration. This iterative loop transforms LLMs’ generative capacity into continuous, adaptive optimization. Crucially, scaling test-time computation yields increasingly specialized CUDA kernels.

2.4 Math

Beyond classical optimization, many mathematical tasks—such as theorem proving, inequality tightening, bound estimation, and geometric construction—can be reframed as search problems. Instead of seeking explicit optimal solutions, these tasks aim to uncover tighter analytical bounds or constructive proofs that approximate theoretical optima [22, 23]. Recent theorem-proving systems,

though highly advanced, typically depend on human-in-the-loop reasoning and rigid symbolic strategies, which may constrain scalability and exploration depth [24, 25].

FM Agent offers a approach by combining symbolic reasoning, numerical experimentation, and evolutionary exploration within a unified framework. It can iteratively generate alternative search strategies, construct target objects, and select relevant prior knowledge, guided by feedback on correctness, efficiency, and a deeper understanding of the current solution. Through such adaptive refinement, FM Agent can autonomously discover near-optimal solutions—achieving better performance criteria and occasionally revealing unexpected theoretical insights.

This perspective treats mathematics as an open-ended search landscape, where reasoning and evolution jointly drive discovery, enabling agents to assist in deriving stronger results and accelerating theoretical innovation.

3 Framework

The framework of FM Agent is designed as a two-stage process to autonomously discover and refine solutions for complex problems. In the Cold Start Stage, several generative agents are applied to solve the problem, aiming to rapidly generate a diverse pool of high-quality algorithms by learning from feedback and acting on instructions. In the subsequent phase, the generated algorithms are partitioned according to the maximum similarity between islands. During this process, we define a number of clusters equal to the number of islands and assign a set of clusters to each island, thereby initiating the subsequent Evolve Stage. In Evolve Stage, an evolutionary search is applied to innovate and improve upon these initial solutions through mutation and crossover mechanisms of island-based population. To achieve high evolutionary efficiency at scale, the framework is deployed on a high-performance distributed cluster that supports large-scale parallel execution. This design significantly enhances throughput, scalability, and overall convergence speed during the evolutionary process.

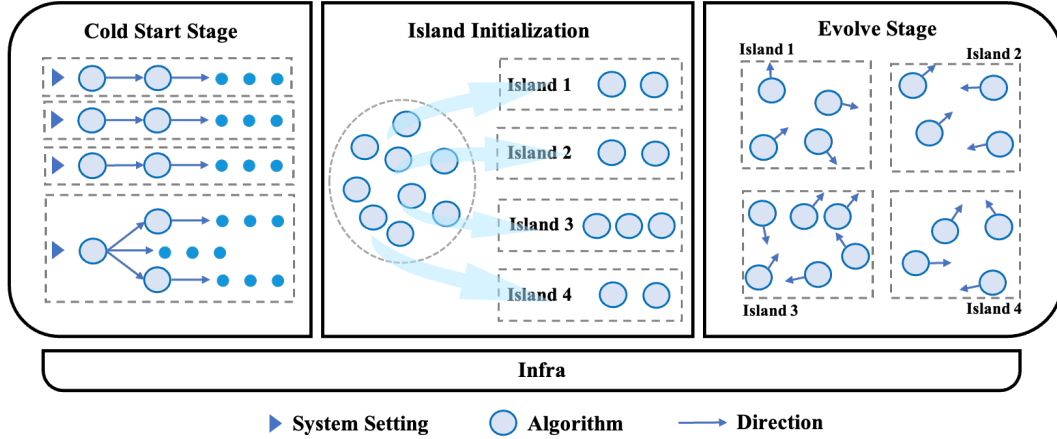


Figure 4: Framework of FM Agent with Cold Start Stage and Evolve Stage, both account for the final performance.

3.1 Cold Start Stage

The Cold Start Stage is dedicated to constructing an initial population of solutions with high diversity, thereby expanding the global solution space and laying a robust foundation for subsequent evolution, effectively mitigating the risk of premature convergence.

Multi-Agent Parallel Expansion. The system integrates diverse types of agents, supporting synchronous exploration of different generation strategies and optimization directions through differentiated configuration and parallel execution. This design significantly enhances the diversity of the initial population and provides the system with excellent scalability.

Proactive Solution Space Expansion. By guiding agents to explore divergent regions of the objective space, the system consciously broadens the coverage of the potential solution space during

initialization. This approach reduces the risk of the evolutionary process becoming trapped in local optima and creates favorable conditions for subsequent in-depth optimization.

3.2 Evolve Stage

The Evolve module implements the core logic of FM Agent, orchestrating a large-scale, population-based search to innovate and improve upon the initial solutions. Its design is centered on principles of diversity preservation, adaptive evolution, and multi-population dynamics, which are encapsulated in an Efficiency Evolution Strategy.

Multi-Population Island Model. FM Agent utilizes a multi-population approach, where solutions are segregated into parallel "islands". On the one hand, each island evolves its population independently in the most time, allowing FM Agent to explore distinct regions of the solution space simultaneously and maintain diverse algorithmic families. On the other hand, the framework also facilitates periodic interaction between these islands, promoting cross-pollination of ideas and preventing the overall search from stagnating in local optima.

Adaptive Diversity-Driven Sampling. The framework employs an adaptive control system to steer evolution within each island, emphasizing diversity preservation through semantic and structural metrics to avoid premature convergence. A novel cluster-based sampling strategy is adopted, which adaptively maintains a balance between exploration and exploitation by dynamically adjusting selective pressure according to real-time population diversity. Moreover, a curated elite pool retains top-performing solutions to guide future generations. The whole adaptive mechanism ensures both sustained innovation and efficient convergence across evolutionary islands.

Domain-Specific Evaluator. To address diverse evaluation demands, the framework employs a flexible, multi-faceted evaluation module providing both general and specialized feedback. General methods include a traditional single fitness score for quantitative ranking and LLM judge feedback for nuanced, qualitative assessment. For complex scenarios, such as machine learning or kernel generation, advanced domain-specific strategies assess multi-dimensional performance. These utilize special fitness metrics that are both numerical (e.g., balancing accuracy and latency) and contextual (e.g., resource utilization), ensuring the evolutionary process is guided by comprehensive domain requirements.

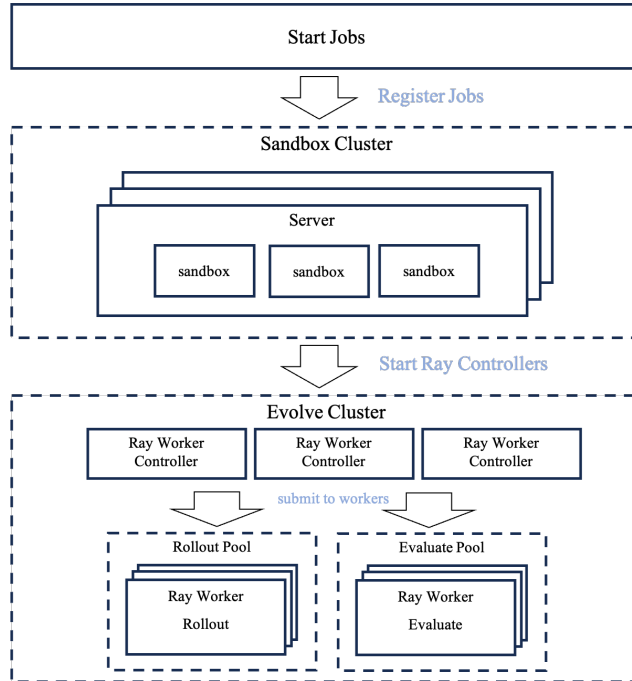


Figure 5: Architecture of the Large-Scale Distributed Evolutionary Cluster.

3.3 Infra

FM Agent operates on a scalable distributed infrastructure purpose-built for high-throughput evolutionary computation. Its design emphasizes two key principles:

Distributed Architecture. The system is orchestrated by the Ray framework, enabling seamless scaling from single-node to large multi-node clusters. Tasks are distributed across independent worker nodes, supporting concurrent execution of thousands of evolutionary processes with efficient resource utilization and fault tolerance.

Asynchronous Generation–Evaluation Pipeline. The two primary workloads, including program generation and program evaluation, are executed in separate, parallelized worker pools. This separation allows asynchronous scheduling and dynamic load balancing, ensuring that compute-intensive synthesis and evaluation phases do not block each other. The result is significantly improved throughput, stability, and overall evolutionary efficiency.

3.4 Human-Interactive Feedback Module

An optional but recommended Human-Interactive Feedback Module is designed to flexibly incorporate domain expertise into the autonomous evolution process. It centers on two key capabilities — real-time interaction and knowledge enhancement — enabling experts to guide and enrich the system without breaking its autonomy.

Real-Time Monitoring and Interactive Intervention. The system provides a panoramic visual interface for monitoring the evolution process, allowing experts to track key metrics such as fitness changes and population diversity in real time. Through natural language instructions (e.g., "prioritize model inference efficiency") or code-level interventions, experts can directly steer the evolutionary direction, ensuring alignment between optimization goals and business objectives.

Knowledge Base Integration and Retrieval Augmentation. The system supports the construction and maintenance of an expert knowledge base. By leveraging RAG technology, it enables efficient retrieval of structured knowledge, such as domain literature and best practices. When specific optimization challenges are encountered, the system automatically retrieves relevant knowledge fragments to inform mutation and crossover operations, thereby enhancing the rationality and interpretability of the search process.

4 Experiment

The efficacy and generalization capabilities of FM Agent were empirically evaluated through a battery of experiments conducted across three distinct yet computationally demanding domains. This evaluation was performed using a suite of established benchmarks, each selected to probe a core competency of the agent’s autonomous, self-evolutionary framework. The chosen benchmarks were ALE-Bench, to assess ability for solving combinatorial optimization problems; MLE-Bench, to evaluate competency in automating complex, real-world machine learning engineering workflows ; and KernelBench, to quantify proficiency in the specialized task of generating high-performance GPU kernels. The results indicate that FM Agent establishes new state-of-the-art performance on these benchmarks, thereby demonstrating the robustness and generalizability of its problem-solving architecture. It is important to note that for all three benchmarks, optimization was performed exclusively by the LLM without human intervention.

4.1 MLE-Bench

MLE-Bench [2], introduced by OpenAI, is an extensive benchmark created to assess systems on complex real-world machine learning tasks based on Kaggle competitions. MLE-Bench comprises 75 machine learning engineering–related competitions from Kaggle, forming a diverse collection of challenging tasks that evaluate practical ML engineering abilities, including model training, dataset processing, and experiment execution.

We evaluate FM Agent on complete MLE-Bench among 75 machine learning tasks. We use the same evaluation metric as MLE-Bench. The results are shown in Table 1 and demonstrate that:

- **Submission Reliability and Broad Competence:** FM Agent demonstrates exceptional robustness by achieving valid submissions in **96.89%** of tasks, a rate that matches or exceeds all other benchmarked agents. This near-perfect submission success rate underscores its remarkable capability to handle a wide array of machine learning challenges effectively and reliably, establishing a consistently high baseline performance.
- **Superior Performance Against Human Benchmark:** The agent’s results are particularly notable when compared to human performance. It surpasses more than half of all human submissions (Above Median) in **51.56%** of the tasks, significantly outperforming other advanced agents such as InternAgent (48.44%) and ML-Master (44.9%). This indicates a strong and generalizable problem-solving ability that is highly competitive within the community.
- **High Performance Ceiling and Medal Attainment:** FM Agent achieves the highest rate of medal acquisition (Any Medal) among all evaluated systems at **43.56%**, with a standout performance in securing Gold medals in **22.67%** of tasks. This exceptional medal distribution, especially the leading gold medal rate, highlights a superior performance ceiling and suggests that the agent can exceed the capabilities of the majority of human machine learning researchers in specific, complex task scenarios.

Table 1: FM Agent surpasses all baseline models across every evaluation dimension defined in MLE-Bench. All values are in percentage (%). The results for MLAB(gpt-4o-2024-08-06), OpenHands(gpt-4o-2024-08-06), AIDE(o1-preview), R&D-Agent(gpt-5), ML-Master(deepseek-r1), Neo multi-agent, InterAgent(deepseek-r1) and Operand ensemble(gpt-5, low verbosity/effort) are taken from the official MLE-Bench report. Results for FM Agent are averaged over three independent runs with different random seeds and are presented as the mean \pm one standard error of the mean (SEM). The best-performing model in each category is highlighted in bold.

Agent	Valid Submission	Above Median	Bronze	Silver	Gold	Any Medal
MLAB [26] gpt-4o-2024-08-06	44.3 \pm 2.6	1.9 \pm 0.7	0.0 \pm 0.0	0.0 \pm 0.0	0.8 \pm 0.5	1.3 \pm 0.5
OpenHands [27] gpt-4o-2024-08-06	52.0 \pm 3.3	7.1 \pm 1.7	0.4 \pm 0.4	1.3 \pm 0.8	2.7 \pm 1.1	5.1 \pm 1.3
AIDE [28] o1-preview	82.8 \pm 1.1	29.4 \pm 1.3	3.4 \pm 0.5	4.1 \pm 0.6	9.4 \pm 0.8	16.9 \pm 1.1
ML-Master [29] deepseek-r1	93.3 \pm 1.3	44.9 \pm 1.2	4.4 \pm 0.9	7.6 \pm 0.4	17.3 \pm 0.8	29.3 \pm 0.8
Neo multi-agent undisclosed	85.78 \pm 4.45	40.00 \pm 1.33	10.22 \pm 2.22	10.22 \pm 0.89	13.78 \pm 3.55	34.22 \pm 0.89
R&D-Agent [30] gpt-5	53.33 \pm 0.00	40.44 \pm 1.77	6.67 \pm 2.67	12.00 \pm 1.33	16.44 \pm 1.77	35.11 \pm 0.44
InternAgent [31] deepseek-r1	96.44 \pm 0.89	48.44 \pm 2.23	7.11 \pm 3.11	10.67 \pm 1.34	18.67 \pm 1.34	36.44 \pm 1.18
Operand ensemble gpt-5 (low verbosity/effort) [†]	55.11 \pm 14.22	40.89 \pm 3.11	20.89 \pm 2.22	7.11 \pm 2.22	11.56 \pm 0.89	39.56 \pm 3.26
FM Agent Gemini-2.5-pro	96.89 \pm 2.22	51.56 \pm 2.23	8.44 \pm 0.89	12.44 \pm 3.56	22.67 \pm 1.34	43.56 \pm 1.78

[†] With some light assistance from an ensemble of models including Gemini-2.5-Pro, Grok-4, and Claude 4.1 Opus, distilled by Gemini-2.5-Pro.

MLE-Bench is structured into three complexity levels: Low (dubbed Lite, containing 22 tasks), Medium (containing 38 tasks) and High (with 15 tasks). From the data presented in Figure 2, FM Agent achieved the remarkably better results on both the Medium and High complexity tasks, indicating more effective performance in the complex scenarios often encountered in real-world production environments. Furthermore, our evaluation across the complete set of 75 competitions reveals that FM Agent achieves top performance in 33 of them, surpassing all other participants on the leaderboard. For detailed results, please refer to the Appendix.

4.2 ALE-Bench

ALE-Bench [1] is designed for objective-driven algorithmic tasks, composed of computationally intractable optimization problems derived from the AtCoder Heuristic Contests, which lack known exact solutions. This format necessitates iterative solution refinement over extended time horizons to achieve higher scores, making it a suitable environment for evaluating advanced agent architectures that extend beyond simple code generation. The experimental protocol utilized the lite version of ALE-Bench, a curated subset of 10 diverse and challenging problems.

FM agent’s performance was benchmarked against two baseline methods presented in the original ALE-Bench publication: an iterative refinement baseline employing a Self-Refine methodology, and the purpose-built ALE-Agent, which incorporates domain-specific knowledge and a diversity-oriented search algorithm [1]. To ensure a controlled comparison, all evaluated systems, including baselines and FM Agent, were configured to use Gemini 2.5 Pro as their foundational large language model.

The empirical results, summarized in Table 2, demonstrate that:

- **State-of-the-Art Overall Performance:** FM Agent establishes a new state-of-the-art performance, achieving a mean overall score of **1976.3**. This surpasses the specialized ALE-Agent (1879.3) by 5.2% and significantly outperforms the iterative refinement baseline (1201.3) by 64.6%.
- **Superior Consistency at Expert Tiers:** While all agents achieved baseline competency (≥ 400 performance) on 100% of tasks, FM Agent showed superior reliability at higher performance levels. It surpassed the ≥ 1600 threshold on **80.0%** of problems, compared to 70.0% for ALE-Agent. More notably, FM Agent reached the expert "Yellow" tier (≥ 2000) on **50.0%** of tasks—a substantial improvement over ALE-Agent’s 30.0%.
- **Exceptional Performance in Long-Horizon Tasks:** The breakdown by contest format highlights a key strength. FM Agent holds a commanding lead in long contests with an average performance of **1701.8**, compared to ALE-Agent’s 1473.8. As success in these long-horizon contests often requires more sophisticated and novel solutions, this suggests FM Agent’s evolutionary approach is particularly effective for the deep reasoning and refinement these complex problems demand.

Table 2: FM Agent’s performance on the ALE-Bench lite subset compared against the Iterative-Refinement and ALE-Agent baselines. The table details the average performance across contest formats and the distribution of performance scores across expert tiers. All agents utilize Gemini 2.5 Pro for a controlled comparison.

Agent	Average Perf.			Perf. Distribution (%)			
	short	long	overall	≥ 400	≥ 1600	≥ 2000	≥ 2400
Iterative-Refinement(Gemini-2.5-pro)	1159.8	1242.8	1201.3	100.0	0.0	0.0	0.0
ALE-Agent(Gemini-2.5-pro)	2284.8	1473.8	1879.3	100.0	70.0	30.0	20.0
FM Agent(Gemini-2.5-pro)	2250.8	1701.8	1976.3	100.0	80.0	50.0	20.0

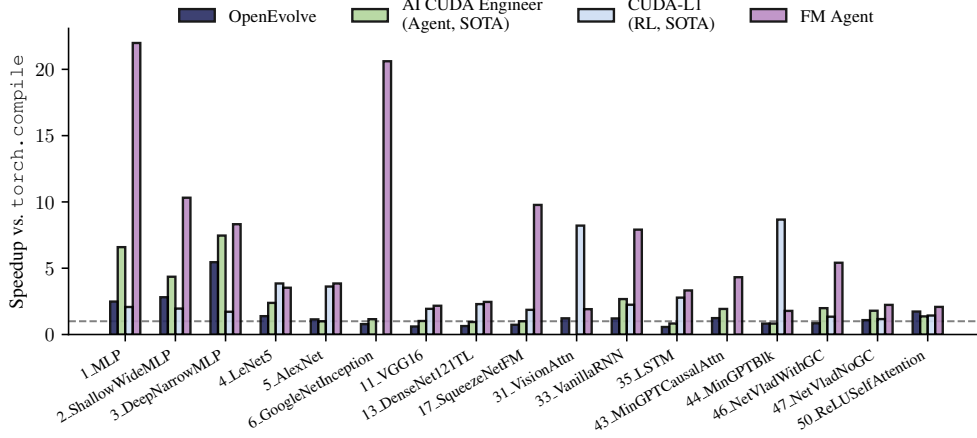
4.3 KernelBench

KernelBench [3] is designed to assess LLMs’ ability to generate efficient GPU kernels. We evaluate our approach on Level 3 of KernelBench, denoting the most challenging kernels, and cluster the problems and construct a workload set that balances coverage and diversity across task types, enabling assessment of both kernel generation and end-to-end model optimization.

To better reflect the demands of real-world production, we tighten the numerical tolerance from 10^{-2} to 10^{-4} , which aligns with the FP16 machine epsilon, reduces rounding errors, and maintains support for mixed precision, thus imposing stricter accuracy requirements. We also increase the warmup iterations ($1 \rightarrow 32$) and profiling iterations ($10 \rightarrow 128$). The benchmark environment is detailed in Table 3.

Table 3: Environment Setup for KernelBench.

Component	Specification
CPU	Intel(R) Xeon(R) Gold 6271C CPU @ 2.60GHz
GPU	NVIDIA A100 80GB
OS	Ubuntu 22.04.5 LTS
Packages	Python 3.12.11, PyTorch 2.6.0 with CUDA 12.4
Toolkit	CUDA Toolkit 12.4, Nsight Compute 2025.2.1, Compute Sanitizer 2023.2.0, Ninja 1.13.0

Figure 6: Comparison of speedup achieved relative to `torch.compile`. The dashed line at 1 indicates parity with `torch.compile`.

We compare against OpenEvolve³, AI CUDA Engineer [32], and CUDA-L1 [33], representing the major lines of progress in CUDA kernel generation—agentic optimization and reinforcement learning respectively. Specifically, AI CUDA Engineer serves as the agentic SOTA, while CUDA-L1 represents the RL-based SOTA. Both SOTA archives⁴⁵ include solutions produced by multiple LLMs (e.g., DeepSeek v3, o3-mini, GPT-4o-mini). To ensure a strictly fair comparison, both our agent and the OpenEvolve baseline are powered by the same LLM (Gemini-2.5-Pro), and we execute all SOTA kernels on an A100 without modification. We further verify that, except for the kernel launcher, all generated kernels are implemented without any dependence on the ATen interface.

Our approach achieves $2.08\times$ to $20.77\times$ speedups over `torch.compile`, consistently outperforming the previous SOTA reached while maintaining numerical accuracy within 10^{-5} .

5 Case Study

5.1 Machine Learning

In machine learning, feature engineering plays a crucial role in real-world applications by transforming raw input data into informative and interpretable representations, therefore, we focus our case study on this stage. However, manual design is slow, expertise-heavy, and hard to scale; existing Auto-FE methods rely on fixed search spaces and lack domain knowledge. Recently, LLM-based attempts help with feature generation and selection [34–36], but reliably discovering high-value features directly from raw data remains difficult.

³<https://github.com/codelion/open evolve>

⁴<https://huggingface.co/datasets/SakanaAI/AI-CUDA-Engineer-Archive>

⁵<https://github.com/deepreinforce-ai/CUDA-L1>

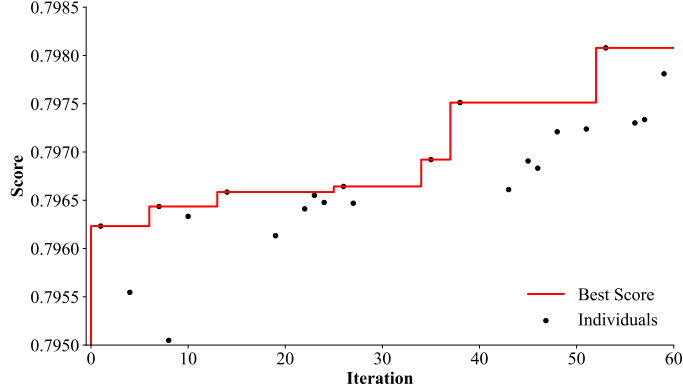


Figure 7: Convergence of the evaluation score on the American Express – Default Prediction task, where the score is the task’s original metric and higher values indicate better performance. The individual points represent the performance scores of mutated solutions in each iteration, while the solid red line tracks the best score achieved.

We introduce domain-specific evaluators and assess our framework on the American Express – Default Prediction task [37], where each sample is an 18-month customer sequence with a default label within 120 days of the last statement. The task maps irregular sequences to fixed-size vectors via temporal transformations. To evaluate the value of feature engineering, throughout the evaluation process, the downstream model and all hyperparameters remain fixed to ensure that performance gains are solely attributable to feature improvements.

As shown in Figure 7, the feature optimization process exhibits continuous performance improvement across iterations. To clarify the optimization trajectory and reporting convention, features are constructed cumulatively: each stage adds newly generated feature families to the existing set, and the reported feature counts represent post-accumulation totals. Below we summarize the cumulative feature sets produced by the agent:

- **Feature Set 1** (iteration 0, Score: 0.7951, 1,097 features). Baseline features comprise numeric aggregations (mean, standard deviation, minimum, maximum, last value), categorical aggregations (count, last, number of unique values), and differentials computed as the difference between the last and penultimate record for each numeric column. These operations summarize per-customer histories into compact temporal profiles.
- **Feature Set 2** (iteration 7, Score: 0.7964, 1,274 features). For each numeric column, the agent computes the least-squares regression slope over the entire sequence window in a fully vectorized manner, capturing global trends across the customer’s history.
- **Feature Set 3** (iteration 14, Score: 0.7966, 1,474 features). Two slope-based families are added: `full_slope_norm` (slopes over the full standardized history) and `recent_slope_norm` (slopes over the most recent six periods on standardized sequences), jointly encoding long- and short-term dynamics.
- **Feature Set 4** (iteration 35, Score: 0.7969, 1,574 features). Numeric variables receive near-window linear trends (local polyfit slopes) and mean absolute changes, while categorical variables receive consistency (proportion of records equal to the final value) and dominance (proportion of the most frequent value), describing local temporal stability and categorical persistence.
- **Feature Set 5** (iteration 53, Score: 0.7981, 1,734 features). Numeric features are enriched with Exponentially Weighted Moving Averages (EWMA) and Exponentially Weighted Volatility (EW variance), emphasizing recent behavior. Categorical features include time-decayed risk mappings (using the last-record target rate per category) and weighted change counts that quantify risk-aware categorical transitions over time.

Overall, these cumulative improvements generated by FM Agent increase the end-to-end score by **+0.003**. Notably, this progress is achieved under a feature-only optimization setting, demonstrating the effectiveness and potential of our framework in the feature engineering stage of machine learning.

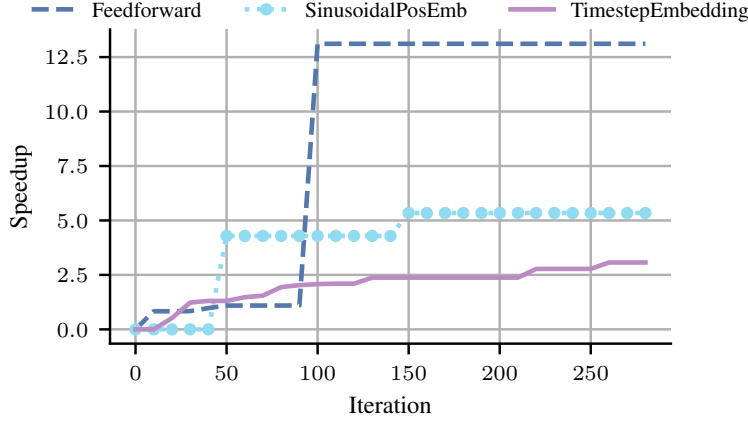


Figure 8: Speedup convergence of kernels in the CosyVoice2-0.5B Flow Matching Decoder against the official PyTorch-based implementation. FeedForward (fusion) and SinusoidalPosEmb (unrolling) converge quickly, while TimestepEmbedding shows slower convergence as it requires exploration of shared memory tiling.

5.2 Kernel Generation

To illustrate the power of this approach, we demonstrate a live optimization session on the Flow Matching Decoder in CosyVoice2-0.5B [38], a critical component whose performance directly impacts user experience. Within its `flow.decoder.estimator` module, our agent automatically optimizes three representative kernels, each posing unique challenges:

- **FeedForward:** FM Agent rapidly identifies operator fusion opportunities, eliminating intermediate memory traffic. Significant speedups emerge within only a few iterations, highlighting efficiency in handling common optimization patterns.
- **SinusoidalPosEmb:** For this lightweight scalar computation, FM Agent explores loop unrolling strategies, converging quickly on an unroll factor that maximizes instruction-level parallelism.
- **TimestepEmbedding:** This GEMM (General Matrix Multiplication) kernel presents a far more complex challenge. Here, FM Agent systematically explores tiling strategies and shared memory allocations. Unlike the other two kernels, improvements are slower but steady, showcasing persistence in navigating vast search spaces. Leveraging memory from past GEMM optimizations, the agent accelerates progress by transferring prior knowledge.

As Figure 8 illustrates, FM Agent adapts its optimization strategy to each kernel’s demands—achieving quick wins in straightforward cases while sustaining long-term exploration in complex, memory-intensive scenarios.

5.3 Math

In the following, we further display the results of applying our system to three mathematical problems introduced in AlphaEvolve [23], where our system outperforms AlphaEvolve and achieves state-of-the-art results. As we mainly focus on complex problems in real-world R&D processes, these early experiments on mathematical tasks primarily serve to deepen our understanding of and bring insights for the design of more effective system for practical optimization challenges encountered in real-world applications.

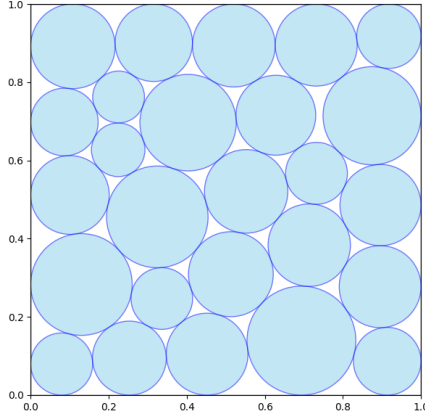
5.3.1 Circle Packing

TASK DESCRIPTION The circle packing optimization problem involves arranging 26 circles inside a unit square with the objective of maximizing the sum of their radii, under the constraints that no

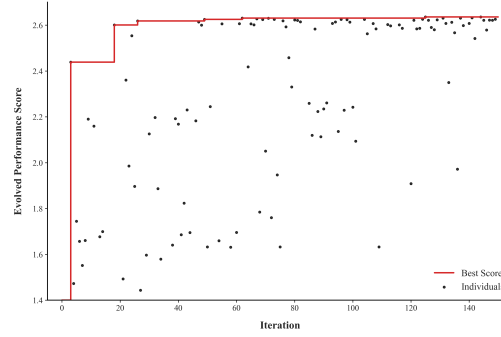
Table 4: Summary of the performance metrics for three mathematical problems. The table reports the previously best-known results (prior to AlphaEvolve), the results achieved by AlphaEvolve [23], and our FM agent.

	Previously best known	AlphaEvolve	FM
Circle packing [†]	2.634	2.635 8627564	2.635 9740012
Ratio minimization*	12.890	12.889 266112	12.889 230201
An uncertainty inequality*	0.3523	0.3520991044 225273	0.3520991044 160562

[†]Maximization task; *Minimization task.



(a) Final Solution for the 26-Circle Packing Problem



(b) Evolutionary Performance on the Circle Packing Problem

Figure 9: Solution and Performance for the Circle Packing Task. Fig 9a shows the final high-density packing configuration of 26 circles within a unit square, representing the state-of-the-art solution discovered by the FM Agent. Fig 9b illustrates the convergence of the evolutionary search over time. The individual points represent the performance scores of mutated solutions in each iteration, while the solid red line tracks the best score achieved so far by FM Agent.

circles overlap and all remain entirely within the square’s boundaries. This constrained optimization challenge integrates discrete placement choices with continuous radius adjustments, establishing it as a sophisticated benchmark in the field of evolutionary algorithms. The problem is characterized by multiple local optima, necessitating advanced search strategies to identify high-quality solutions. Naive methods tend to converge prematurely to suboptimal configurations that poorly utilize the available space.

SOLUTION The evolved solution constitutes a three-stage hybrid optimization methodology that yields a high-density packing configuration. The efficacy of the algorithm is rooted in its structured, sequential approach. The process commences with a geometric initialization phase, wherein a staggered, hexagonal-like lattice structure is generated to establish an advantageous initial condition consistent with dense packing theory. Subsequently, a staged, physics-informed simulation is executed, employing a two-stage gradient-based optimization routine. This routine initially applies a repulsive force to facilitate global exploration and mitigate premature convergence, followed by the introduction of an aggressive overlap penalty to refine the circle center coordinates. In the terminal stage, a Linear Programming (LP) polish is performed; for the determined set of centers, an LP solver is utilized to compute the mathematically exact optimal radii, thereby maximizing space utilization for the final configuration.

INSIGHTS The evolutionary process demonstrates both the efficiency and effectiveness of FM Agent. As shown in 9b, the system converges on a near-optimal solution remarkably quickly, with the best score making significant leaps within the first 25 iterations. This rapid improvement highlights

the system’s efficiency in identifying promising regions of the vast search space without wasteful exploration. The performance of individual mutations, shown as scattered points, further illuminates the agent’s strategy. Initially, the scores exhibit high variance, reflecting a broad exploration phase where diverse algorithmic approaches are tested. As the evolution progresses, the variance of these individual scores gradually decreases, and the points cluster more tightly around the best-known score. This transition from exploration to exploitation indicates the effectiveness of the framework’s sampling and database management. The system successfully identifies high-quality parent solutions and refines them, leading to consistent, incremental improvements in the later stages. This dynamic adjustment allows the agent to navigate the complex trade-offs of the problem, ultimately discovering and refining a state-of-the-art solution as depicted in 9a.

5.3.2 An uncertainty inequality

TASK DESCRIPTION The goal of this task is to find the minimal upper bound for a theoretically existing constant in an uncertainty inequality. Specifically, given a function $f : \mathbb{R} \rightarrow \mathbb{R}$, denote the Fourier transform $\hat{f}(\xi) := \int_{\mathbb{R}} f(x) e^{-2\pi i x \xi} dx$ and

$$A(f) := \inf\{r > 0 : f(x) \geq 0 \text{ for all } |x| \geq r\}.$$

Denote C^* as the largest constant such that

$$A(f)A(\hat{f}) \geq C^*, \text{ for all even function } f \text{ with } \max(f(0), \hat{f}(0)) < 0.$$

It is known that $0.2025 \leq C^* \leq 0.3523$ [22]. Our goal is to construct a function f^* such that the corresponding value of $A(f^*)A(\hat{f}^*)$, which serves as an upper bound of C^* , is as small as possible.

AlphaEvolve [23] further improved the upper bound to $C^* \leq 0.3520991044225273$, while FM Agent has achieved a better solution to $C^* \leq 0.3520991044160562$.

SOLUTION Searching for this function over the entire function space is infeasible. Therefore, [22] proposed a construction that restricts the search space to functions of the form $f^*(x) = P(x)e^{-\pi x^2}$, where $P(x) = \sum_{k=0}^K c_k H_{4k}(x)$, and $H_{4k}(x)$ are Hermite polynomials. Building on this approach, AlphaEvolve further imposed $P(0) = 0$, set $K = 3$, and searched for the three coefficients c_0, c_1, c_2 (with c_3 fully determined by them), thereby solve the problem.

Using the same construction by [22], our FM agent found three coefficients : $c_0 \approx 4.40581122518366186113780713640$, $c_1 \approx -0.1550236238960183143831272900$, $c_2 \approx -0.0011938260171886596119894541$, thus improved the upper bound to $C^* \leq 0.3520991044160562$.

Figure 10b visualizes the corresponding function. The strategy that produced the above results can be regarded as a variant of differential evolution algorithm (see Listing 4 in the appendix). A closer examination of the FM Agent’s evolutionary process reveals that before reaching the optimal solution, it explored several other search strategies, including random search, simulated annealing, L-BFGS (Limited-memory Broyden–Fletcher–Goldfarb–Shanno) and SLSQP (Sequential Least Squares Programming).

INSIGHTS In this task, “An uncertainty inequality” task involves the deliberate incorporation of domain knowledge, namely, the construction method proposed in [22]. As noted in [23] Appendix B.4, employing more advanced constructions [39], could enable AlphaEvolve to further improve the above results. Our experiment also proves that different problem settings could result in different achievements. This confirms that the search performance is highly sensitive to the problem abstractions derived from expert domain knowledge, and that different human-provided constructions can significantly impact the final results.

5.3.3 Minimizing the ratio of maximum to minimum distance in the 2-dimensional space

TASK DESCRIPTION The goal of this task is to find 16 points in the 2-dimensional space such that the ratio of the maximum and minimum pairwise distances is minimized.

AlphaEvolve [23] found 16 points with ratio $\approx \sqrt{12.889266112}$. FM Agent achieves the ratio $\approx \sqrt{12.889230201}$, with the 16 points are shown in Listing 1 and visualized in Figure 10a.

Listing 1: 16 points in 2D space founded by FM agent.

```
[ -1.47975561, 0.98098357], [ 0.85184808, 0.07211039], [ -0.73461161, 1.64788717],  
[ 0.15127319, -1.78975576], [ -0.71559290, 0.33596005], [ -1.54547711, -0.91892085],  
[ 1.73300231, -0.45160218], [ -1.82309280, 0.04177136], [ 1.73113866, 0.54839609],  
[ 1.15533190, 1.36598190], [ 0.40491725, -0.82245813], [ -0.58095076, -0.65493425],  
[ 0.16887753, 0.80255630], [ 0.25391499, 1.79893406], [ -0.83459483, -1.62223187],  
[ 1.26377170, -1.33467785]
```

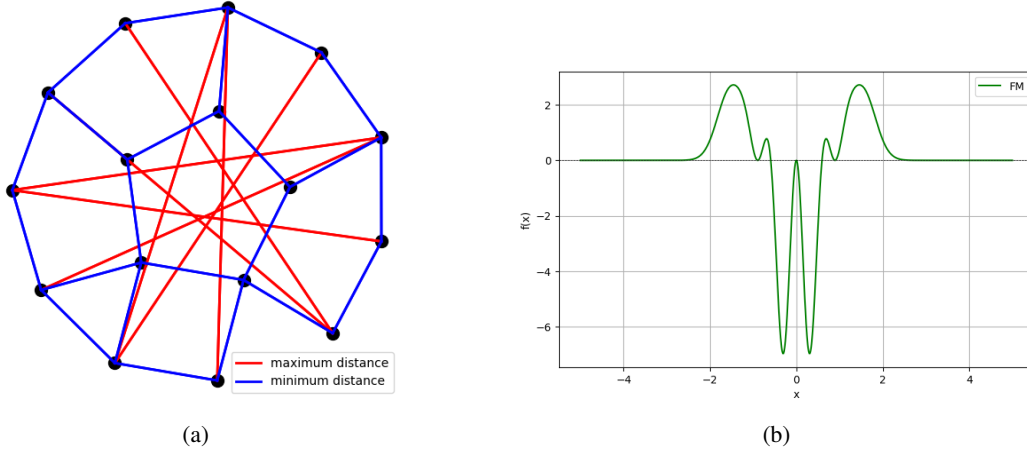


Figure 10: Visualization of the results using the verification code provided in AlphaEvolve [23]. Fig 10a: Minimizing the ratio of maximum to minimum distance in the 2-dimensional space; Fig 10b: An uncertainty inequality.

SOLUTION Delving into the specific strategy for generating these 16 points (see Listing 5 in the appendix), the optimization is performed using the SLSQP method (Sequential Least Squares Programming), a gradient-based algorithm suitable for smooth constrained optimization problems. To enhance robustness and reduce the risk of convergence to poor local minima, a multi-start strategy is employed with five diverse initial configurations: a hexagonal lattice, a 1–5–10 concentric ring structure, a 6–10 two-ring structure, a 4×4 regular grid, and a random layout. Each configuration is independently optimized, and the one yielding the lowest maximum-to-minimum distance ratio is selected as the final solution. Additionally, the program employs a vectorized Jacobian for the constraint gradients, which improves computational efficiency when evaluating the many pairwise distance constraints.

INSIGHTS In this task, we find that even without formal proofs, high-level guidance inspired by human intuition can effectively constrain the search space by informing the system setup. By seeding the optimization with several diverse structures, such as a hexagonal lattice, concentric rings, regular grids, and random layouts, FM Agent can explore promising regions more effectively and avoid poor local minima. Compared with fully zero initialization, these coarse, expert-inspired hints reduce the effective search space while still allowing the agent to refine the solution, achieving similarly strong or better performance. This demonstrates that even partial domain knowledge, such as rough spatial arrangements, can substantially guide the agent and improve overall optimization results.

6 Ablations

To rigorously quantify the contributions of our proposed components, we conducted a comprehensive ablation study against the baseline system. All evaluations were performed on a selected task, ahc016⁶, from ALE-Bench. The competition requires encoding integers as graphs so that, after noise and vertex shuffling, the original integer can be accurately recovered. Participants must generate a set

⁶https://atcoder.jp/contests/ahc016/tasks/ahc016_a

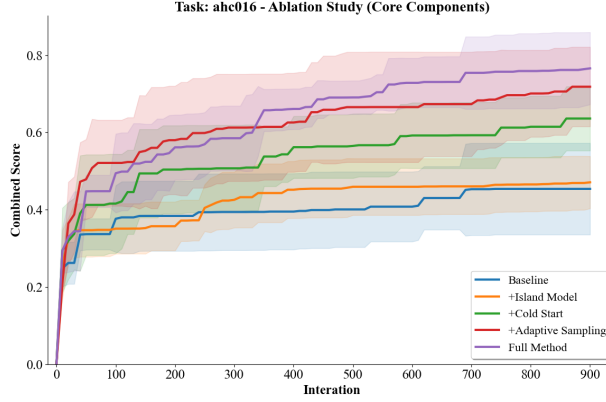


Figure 11: Ablation study results of FM Agent on the ahc016 task. Each curve displays the performance of a different experimental setting, averaged over five independent runs (where a higher combined score is better). The shading indicates the standard deviation.

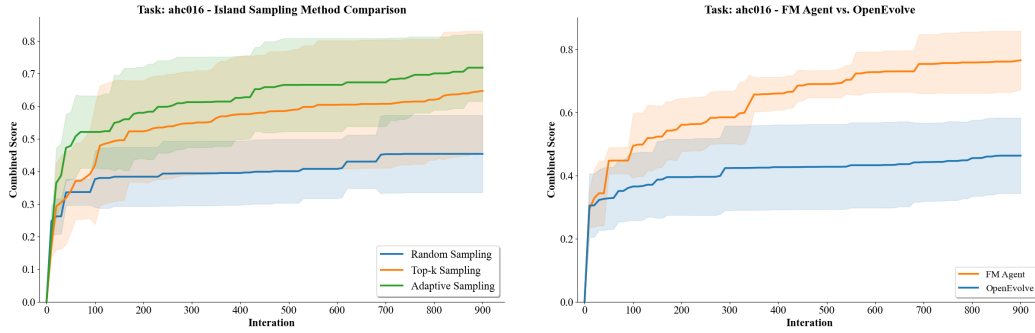


Figure 12: **Left:** Comparison of different sampling methods on the ahc016 task. **Right:** Compared with open-source baseline on the ahc016 task. Each curve displays the performance of a different experimental setting, averaged over five independent runs (where a higher combined score is better). The shading indicates the standard deviation.

of graphs and then predict the source of each noisy graph, aiming to maximize accuracy while minimizing graph size.

Due to the inherent stochasticity in our evolutionary framework and the relatively high computational cost of each run, we executed each experimental setting five times on the same task and report the averaged results. This procedure ensures that the observed performance differences reliably reflect the contributions of individual components rather than random fluctuations.

Overall Contribution. As shown in Figure 11, quantitative comparison demonstrates that each proposed component—adaptive sampling, cold start, and island model—meaningfully contributes to the overall performance improvement. The Full Method, which integrates all components, achieves the best overall performance, confirming a powerful synergistic effect by combining their individual strengths to surpass all other configurations in the high final score. Specifically, the cold start strategy provided a significant initial advantage, accelerating the system’s early convergence rate. The adaptive sampling strategy continuously drives performance improvement by achieving a better balance between exploration and exploitation. The island model strategy, by promoting solution diversity, rapidly reached the baseline’s peak performance level during the intermediate phase.

Adaptive Sampling Strategy. We benchmarked the adaptive sampling method against two baselines, including random sampling and top-k sampling, in Figure 12 (Left). Random sampling method selects programs uniformly, emphasizing pure exploration, while top-k sampling method propagates

only the top-k programs, representing pure exploitation. Our method achieves a final combined score of **0.7182**, outperforming top sampling by **10.99%** and random sampling by **58.26%**. Notably, the adaptive sampling surpasses the maximum score of random sampling as early as **Iteration40**, demonstrating faster convergence (vs **Iteration900** for random sampling) and a higher performance ceiling.

Comparison with Open-Source Baseline. The comparison of our complete FM Agent framework against OpenEvolve⁷ is presented in Figure 12 (Right). The results confirm the effectiveness of our integrated design: FM Agent achieves faster convergence, higher final scores, and a more robust evolutionary trajectory.

7 Related work

7.1 Multi-Agent Systems

The complexity of modern software development has driven a shift from single-agent solutions to multi-agent collaboration paradigms, mirroring human development teams. Initial pioneering systems, such as [40–42], established structured, specialized pipelines with fixed roles (e.g., analysis, coding, testing). Subsequent research aimed to enhance the efficiency and reliability of these collaborations, introducing mechanisms like "dual-collaboration" [43] for information reuse, declarative memory modules [44] to reduce redundancy, and rigorous generate-test-fix loops (e.g., LingMa [45], CodeCor [46]). Moving beyond simple code generation to complex R&D and optimization tasks, efforts have focused on systems employing specialized collaborative roles. For instance, R&D Agent [30] utilizes a dual Researcher and Developer agent collaboration, enhancing diversity by periodically merging superior results, and MLE-STAR [47] integrates collaborative agents with external knowledge, using ablation study guided refinement for targeted code improvement. More recently, systems like CoMAS [48] explored autonomous improvement by learning from interactions, and InternAgent [31] introduced a unified closed-loop research framework that seamlessly integrates idea generation, experiment execution, and result feedback. However, a limitation of most prior work is their primary focus on code generation tasks, and FM Agent addresses this by shifting the focus from simply generating correct code to autonomously discovering state-of-the-art solutions in a large-scale searching way across diverse, complex problem domains and real-world scenarios.

7.2 Search-driven and Evolutionary Paradigms

The Search-driven Paradigm leverages the generative power of Large Language Models (LLMs) combined with iterative evolutionary or reinforcement-style loops to systematically explore vast solution spaces. This was notably pioneered by FunSearch [49] with its foundational generate-score-evolve closed-loop mechanism. Subsequent work scaled this concept through frameworks like EoH [50] with its "Thought-Code" dual-evolution, and AlphaEvolve [23] for modifying entire codebases. Other efforts enhanced the search efficacy and strategy by shifting the evolutionary focus from individual solutions to the search space itself, such as X-evolve [51] which generates tunable, parameterized programs, and C-Evolve [52] which evolves consensus-driven prompts. Refinements to search strategy include the parallel beam-search strategy employed by ALE-Bench [1] to explore multiple promising paths, and ML-Master [29] which integrates Monte Carlo Tree Search (MCTS) inspired exploration with an adaptive memory mechanism. ShinkaEvolve[53] improves sample efficiency through synergistic use of parent program sampling strategy, rejection-sampling, and LLM ensemble selection. We further optimized this paradigm by integrating expert-guided cold-start initialization to jumpstart the search, a novel adaptive diversity-driven sampling strategy for superior exploration-exploitation balance, and a distributed, asynchronous Ray infrastructure that enables the required large-scale search, which is essential for achieving state-of-the-art results on industrial-scale scenarios.

References

- [1] Yuki Imajuku, Kohki Horie, Yoichi Iwata, Kensho Aoki, Naohiro Takahashi, and Takuya Akiba. Ale-bench: A benchmark for long-horizon objective-driven algorithm engineering. *arXiv*

⁷<https://github.com/codelion/openevolve>

preprint arXiv:2506.09050, 2025.

- [2] Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, and et al. Mle-bench: Evaluating machine learning agents on machine learning engineering. *arXiv preprint arXiv:2410.07095*, 2024.
- [3] Anne Ouyang, Simon Guo, Simran Arora, Alex L Zhang, William Hu, Christopher Re, and Azalia Mirhoseini. Kernelbench: Can LLMs write efficient GPU kernels? In *Scaling Self-Improving Foundation Models without Human Supervision*, 2025. URL <https://openreview.net/forum?id=k6V4jb8jkX>.
- [4] Zan Li and Rui Fan. Financial distress warning and risk path analysis for chinese listed companies: An interpretable machine learning approach. *Economic Modelling*, 152:107288, 2025.
- [5] Unknown Author. A new hybrid neural network framework inspired by biological systems for advanced financial forecasting. *Scientific Reports*, 15:36985, 2025.
- [6] Dongchan Cho, Jiho Han, Keumyeong Kang, Minsang Kim, Honggyu Ryu, and Namsoon Jung. Structured temporal causality for interpretable multivariate time series anomaly detection. *arXiv preprint arXiv:2510.16511*, 2025.
- [7] Christodoulos A Floudas and Xiaoxia Lin. Mixed integer linear programming in process scheduling: Modeling, algorithms, and applications. *Annals of Operations Research*, 139(1): 131–162, 2005.
- [8] Eray Demirel, Neslihan Demirel, and Hadi Gökçen. A mixed integer linear programming model to optimize reverse logistics activities of end-of-life vehicles in turkey. *Journal of Cleaner Production*, 112:2101–2113, 2016.
- [9] Steven Cheng, Christine W Chan, and Gordon H Huang. An integrated multi-criteria decision analysis and inexact mixed integer linear programming approach for solid waste management. *Engineering Applications of Artificial Intelligence*, 16(5-6):543–554, 2003.
- [10] Tobias Achterberg. Scip: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.
- [11] Yijia Zhang, Zhihong Gou, Shijie Cao, Weigang Feng, Sicheng Zhang, Guohao Dai, and Ningyi Xu. Automating energy-efficient gpu kernel generation: A fast search-based compilation approach, 2024. URL <https://arxiv.org/abs/2411.18873>.
- [12] Yuduo Wu, Yangzihao Wang, Yuechao Pan, Carl Yang, and John D. Owens. Performance characterization of high-level programming models for gpu graph analytics. In *2015 IEEE International Symposium on Workload Characterization*, pages 66–75, 2015. doi: 10.1109/IISWC.2015.13.
- [13] Philippe Tillet, H. T. Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, MAPL 2019, page 10–19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367196. doi: 10.1145/3315508.3329973. URL <https://doi.org/10.1145/3315508.3329973>.
- [14] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. Tvm: An automated end-to-end optimizing compiler for deep learning, 2018. URL <https://arxiv.org/abs/1802.04799>.
- [15] Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, C. K. Luk, Bert Maher, Yunjie Pan, Christian Puhersch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Shunting

- Zhang, Michael Suo, Phil Tillet, Xu Zhao, Eikan Wang, Keren Zhou, Richard Zou, Xiaodong Wang, Ajit Mathews, William Wen, Gregory Chanan, Peng Wu, and Soumith Chintala. Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '24, page 929–947, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703850. doi: 10.1145/3620665.3640366. URL <https://doi.org/10.1145/3620665.3640366>.
- [16] Mengdi Wu, Xinhao Cheng, Shengyu Liu, Chunan Shi, Jianan Ji, Man Kit Ao, Praveen Vellengiri, Xupeng Miao, Oded Padon, and Zhihao Jia. Mirage: A {Multi-Level} superoptimizer for tensor programs. In *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25)*, pages 21–38, 2025.
 - [17] Yu Cheng, Lei Wang, Yining Shi, Yuqing Xia, Lingxiao Ma, Jilong Xue, Yang Wang, Zhiwen Mo, Feiyang Chen, Fan Yang, et al. Pipethreader: Software-defined pipelining for efficient dnn execution. In *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25)*, 2025.
 - [18] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. Mlir: Scaling compiler infrastructure for domain specific computation. In *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 2–14, 2021. doi: 10.1109/CGO51591.2021.9370308.
 - [19] Ming Zhong, Fang Lv, Lulin Wang, Lei Qiu, Yingying Wang, Ying Liu, Huimin Cui, Xiaobing Feng, and Jingling Xue. Vega: Automatically generating compiler backends using a pre-trained transformer model. In *Proceedings of the 23rd ACM/IEEE International Symposium on Code Generation and Optimization, CGO '25*, page 90–106, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400712753. doi: 10.1145/3696443.3708931. URL <https://doi.org/10.1145/3696443.3708931>.
 - [20] Jubi Taneja, Avery Laird, Cong Yan, Madan Musuvathi, and Shuvendu K. Lahiri. Llm-vectorizer: Llm-based verified loop vectorizer. In *Proceedings of the 23rd ACM/IEEE International Symposium on Code Generation and Optimization, CGO '25*, page 137–149, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400712753. doi: 10.1145/3696443.3708929. URL <https://doi.org/10.1145/3696443.3708929>.
 - [21] Guoliang He and Eiko Yoneki. Cuasmrl: Optimizing gpu sass schedules via deep reinforcement learning. In *Proceedings of the 23rd ACM/IEEE International Symposium on Code Generation and Optimization, CGO '25*, page 493–506, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400712753. doi: 10.1145/3696443.3708943. URL <https://doi.org/10.1145/3696443.3708943>.
 - [22] Felipe Gonçalves, Diogo Oliveira e Silva, and Stefan Steinerberger. Hermite polynomials, linear flows on the torus, and an uncertainty principle for roots. *Journal of Mathematical Analysis and Applications*, 451(2):678–711, 2017.
 - [23] Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.
 - [24] Luoxin Chen, Jinming Gu, Liankai Huang, Wenhao Huang, Zhicheng Jiang, Allan Jie, Xiaoran Jin, Xing Jin, Chenggang Li, Kaijing Ma, et al. Seed-prover: Deep and broad reasoning for automated theorem proving. *arXiv preprint arXiv:2507.23726*, 2025.
 - [25] Yichen Huang and Lin F Yang. Gemini 2.5 pro capable of winning gold at imo 2025. *arXiv preprint arXiv:2507.15855*, 2025.
 - [26] Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. Mlagentbench: Evaluating language agents on machine learning experimentation. *arXiv preprint arXiv:2310.03302*, 2023.

- [27] Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741 [cs.SE]*, 2024. URL <https://arxiv.org/abs/2407.16741>.
- [28] Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jancenko, and Yuxiang Wu. Aide: Ai-driven exploration in the space of code. *arXiv preprint arXiv:2502.13138*, 2025.
- [29] Zexi Liu, Yuzhu Cai, Xinyu Zhu, Yujie Zheng, Runkun Chen, Ying Wen, Yanfeng Wang, Siheng Chen, et al. Ml-master: Towards ai-for-ai via integration of exploration and reasoning. *arXiv preprint arXiv:2506.16499*, 2025.
- [30] Xu Yang, Xiao Yang, Shikai Fang, Bowen Xian, Yuante Li, Jian Wang, Minrui Xu, Haoran Pan, Xinpeng Hong, Weiqing Liu, and et al. R&d-agent: Automating data-driven ai solution building through llm-powered automated research, development, and evolution. *arXiv preprint arXiv:2505.14738*, 2025.
- [31] Bo InternAgent Team: Zhang, Shiyang Feng, Xiangchao Yan, Jiakang Yuan, Runmin Ma, Yusong Hu, Zhiyin Yu, Xiaohan He, Songtao Huang, Shaowei Hou, Zheng Nie, Zhilong Wang, Jinyao Liu, Tianshuo Peng, Peng Ye, Dongzhan Zhou, Shufei Zhang, Xiaosong Wang, Yilan Zhang, Meng Li, Zhongying Tu, Xiangyu Yue, Wangli Ouyang, Bowen Zhou, and Lei Bai. Internagent: When agent becomes the scientist – building closed-loop system from hypothesis to verification. *arXiv preprint arXiv:2505.16938 [cs.AI]*, 2025. URL <https://arxiv.org/abs/2505.16938>.
- [32] Robert Tjarko Lange, Aaditya Prasad, Qi Sun, Maxence Faldor, Yujin Tang, and David Ha. The ai cuda engineer: Agentic cuda kernel discovery, optimization and composition. Technical report, Technical report, Sakana AI, 02 2025, 2025.
- [33] Xiaoya Li, Xiaofei Sun, Albert Wang, Jiwei Li, and Chris Shum. Cuda-1l: Improving cuda optimization via contrastive reinforcement learning, 2025. URL <https://arxiv.org/abs/2507.14111>.
- [34] Noah Hollmann, Samuel Müller, and Frank Hutter. Large language models for automated data science: Introducing caafe for context-aware automated feature engineering. *Advances in Neural Information Processing Systems*, 36:44753–44775, 2023.
- [35] Sungwon Han, Jinsung Yoon, Sercan O Arik, and Tomas Pfister. Large language models can automatically engineer features for few-shot tabular learning. In *International Conference on Machine Learning*, pages 17454–17479. PMLR, 2024.
- [36] Nanxu Gong, Xinyuan Wang, Wangyang Ying, Haoyue Bai, Sixun Dong, Haifeng Chen, and Yanjie Fu. Unsupervised feature transformation via in-context generation, generator-critic llm agents, and duet-play teaming. In James Kwok, editor, *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI-25*, pages 2820–2828. International Joint Conferences on Artificial Intelligence Organization, 8 2025. doi: 10.24963/ijcai.2025/314. URL <https://doi.org/10.24963/ijcai.2025/314>. Main Track.
- [37] Addison Howard, AritraAmex, Di Xu, Hossein Vashani, inversion, Negin, and Sohler Dane. American express - default prediction. <https://kaggle.com/competitions/amex-default-prediction>, 2022. Kaggle.
- [38] Zhihao Du, Yuxuan Wang, Qian Chen, Xian Shi, Xiang Lv, Tianyu Zhao, Zhifu Gao, Yexin Yang, Changfeng Gao, Hui Wang, Fan Yu, Huadai Liu, Zhengyan Sheng, Yue Gu, Chong Deng, Wen Wang, Shiliang Zhang, Zhijie Yan, and Jingren Zhou. Cosyvoice 2: Scalable streaming speech synthesis with large language models, 2024. URL <https://arxiv.org/abs/2412.10117>.
- [39] Henry Cohn and Felipe Gonçalves. An optimal uncertainty principle in twelve dimensions via modular forms. *Inventiones mathematicae*, 217(3):799–831, 2019.

- [40] Yihong Dong, Xue Jiang, Zhi Jin, and Ge Li. Self-collaboration code generation via chatgpt. *ACM Transactions on Software Engineering and Methodology*, 33(7):1–38, 2024.
- [41] D Huang, JM Zhang, M Luck, Q Bu, Y Qing, and H Cui. Agentcoder: Multi-agent code generation with effective testing and self-optimization. *University of Hong Kong, King’s College London, University of Sussex, Shanghai Jiao Tong University*, 2024.
- [42] Zeeshan Rasheed, Malik Abdul Sami, Kai-Kristian Kemell, Muhammad Waseem, Mika Saari, Kari Systä, and Pekka Abrahamsson. Codepori: Large-scale system for autonomous software development using multi-agent technology. *arXiv preprint arXiv:2402.01411*, 2024.
- [43] Dake Chen, Hanbin Wang, Yunhao Huo, Yuzhao Li, and Haoyang Zhang. Gamegpt: Multi-agent collaborative framework for game development. *arXiv preprint arXiv:2310.08067*, 2023.
- [44] Danrui Qi, Zhengjie Miao, and Jiannan Wang. Cleanagent: Automating data standardization with llm-based agents. *arXiv preprint arXiv:2403.08291*, 2024.
- [45] Yingwei Ma, Rongyu Cao, Yongchang Cao, Yue Zhang, Jue Chen, Yibo Liu, Yuchen Liu, Binhua Li, Fei Huang, and Yongbin Li. Lingma swe-gpt: An open development-process-centric language model for automated software improvement. *arXiv preprint arXiv:2411.00622*, 2024.
- [46] Ruwei Pan, Hongyu Zhang, and Chao Liu. Codecor: An llm-based self-reflective multi-agent framework for code generation. *arXiv preprint arXiv:2501.07811*, 2025.
- [47] Jaehyun Nam, Jinsung Yoon, Jiefeng Chen, Jinwoo Shin, Sercan Ö Arık, and Tomas Pfister. Mle-star: Machine learning engineering agent via search and targeted refinement. *arXiv preprint arXiv:2506.15692*, 2025.
- [48] Xiangyuan Xue, Yifan Zhou, Guibin Zhang, Zaibin Zhang, Yijiang Li, Chen Zhang, Zhenfei Yin, Philip Torr, Wanli Ouyang, and Lei Bai. Comas: Co-evolving multi-agent systems via interaction rewards, 2025. URL <https://arxiv.org/abs/2510.08529>.
- [49] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.
- [50] Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. *arXiv preprint arXiv:2401.02051*, 2024.
- [51] Yi Zhai, Zhiqiang Wei, Ruohan Li, Keyu Pan, Shuo Liu, Lu Zhang, Jianmin Ji, Wuyang Zhang, Yu Zhang, and Yanyong Zhang. $\backslash(x\backslash)$ -evolve: Solution space evolution powered by large language models. *arXiv preprint arXiv:2508.07932*, 2025.
- [52] Tiancheng Li, Yuhang Wang, Zhiyang Chen, Zijun Wang, Liyuan Ma, and Guo-jun Qi. C-evolve: Consensus-based evolution for prompt groups. *arXiv preprint arXiv:2509.23331*, 2025.
- [53] Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. Shinkaevolve: Towards open-ended and sample-efficient program evolution. *arXiv preprint arXiv:2509.19349*, 2025.

A. MLE-Bench: Full results

Full table of results. We present the complete scores of FM Agent across all 75 competitions in MLE-Bench. In accordance with the official MLE-Bench protocol requiring three independent submissions per competition, we report the highest score achieved across these three runs for each competition. These results are compared against several top-performing participants on the leaderboard, including Operand ensemble, InternAgent, R&D-Agent, Neo multi-agent, and ML-Master. As shown in the table, FM Agent achieved first place among all agents in 33 out of the 75 competitions. Its performance is even more pronounced on the hard split, where it secured first place in 9 out of the 15 total challenges. This result strongly demonstrates FM Agent’s enhanced capability in handling complex tasks.

B. Feature Mining: American Express Default Prediction task

As discussed in Section 5.1, we illustrate FM Agent’s automated feature construction using Feature Set 5. Each sample is a customer’s irregular sequence of statements ordered by the statement date S_2 (the per-record timestamp). For a given numerical time series $\{x_1, x_2, \dots, x_n\}$, where x_i denotes the value at the i -th record in chronological order, $i \in \{1, \dots, n\}$, and n is the number of records for that customer, the agent applies an exponential recency scheme governed by a decay hyperparameter $\alpha = 0.3$. The temporal weight w_i for record i is

$$w_i = (1 - \alpha)^{(n-i)},$$

so more recent observations (larger i) receive larger influence.

For numerical variables, the agent constructs two exponentially weighted statistics. The exponentially weighted moving average $\text{EWMA}(x)$ summarizes the recent level of the series, and the exponentially weighted volatility $\text{EWVOL}(x)$ summarizes its recent dispersion:

$$\text{EWMA}(x) = \frac{\sum_{i=1}^n w_i x_i}{\sum_{i=1}^n w_i}, \quad \text{EWVOL}(x) = \sqrt{\frac{\sum_{i=1}^n w_i (x_i - \text{EWMA}(x))^2}{\sum_{i=1}^n w_i}}.$$

Here, $\text{EWMA}(x)$ and $\text{EWVOL}(x)$ are scalar aggregations per customer and feature, computed from the weighted deviations $(x_i - \text{EWMA}(x))$ under weights $\{w_i\}_{i=1}^n$.

For categorical variables, let c_i denote the category observed at record i in a customer’s sequence, and let $y \in \{0, 1\}$ denote the default label (1 if default occurs within 120 days of the last statement, 0 otherwise). FM Agent first estimates a category-to-risk mapping $r(\cdot)$ using the final record of each customer, where $r(v)$ is the empirical mean of y for category value v . This produces a risk-encoded sequence $\{r(c_i)\}_{i=1}^n$. The agent then constructs a recency-weighted risk exposure, $\text{RiskScore}(c)$, and a recency-weighted switching intensity, $\text{EWChanges}(c)$:

$$\text{RiskScore}(c) = \frac{\sum_{i=1}^n w_i r(c_i)}{\sum_{i=1}^n w_i}, \quad \text{EWChanges}(c) = \sum_{i=2}^n w_i \cdot \mathbb{I}(c_i \neq c_{i-1}),$$

where $\mathbb{I}(\cdot)$ is the indicator function (equals 1 if its condition is true and 0 otherwise). Together, $\text{EWMA}(x)$, $\text{EWVOL}(x)$, $\text{RiskScore}(c)$, and $\text{EWChanges}(c)$ provide an automated, interpretable representation of recent behavioral levels, short-term variability, risk-aligned categorical tendencies, and temporal transition patterns, respectively, all produced by FM Agent under a fixed downstream model.

This feature set enables the downstream model to capture recent behavioral levels, short-term instability, and risk-aware categorical transition patterns, forming a fully automated and interpretable temporal representation contributed by the agent.

Listing 2: Agent-generated program implementing Feature Set 5

```
import pandas as pd
import numpy as np
from joblib import Parallel, delayed
import pyarrow.parquet as pq
import gc
```

```
from tqdm.auto import tqdm

CAT_FEATURES = [
    "B_30", "B_38", "D_114", "D_116", "D_117",
    "D_120", "D_126", "D_63", "D_64", "D_66", "D_68"
]
PARALLEL_WORKERS = -1
MAX_NUMERICAL = 70
```

```

MAX_CATEGORICAL = 10
ALPHA = 0.3

def process_customer(group, num_cols, cat_cols,
                    risk_cols):
    n = len(group)
    w = np.array([(1-ALPHA)**(n-1-i) for i in range(
        n)])
    f = {}
    for col in num_cols:
        v = group[col].fillna(0).values.astype(np.
            float32)
        ewma = np.dot(w,v)/w.sum()
        ewol = np.sqrt(np.dot(w,(v-ewma)**2)/w.sum(
            ))
        f[f"{col}_ewma"] = ewma
        f[f"{col}_ewol"] = ewol
    for i,col in enumerate(cat_cols):
        r = group[risk_cols[i]].values.astype(np.
            float32)
        ch = (group[col]!=group[col].shift(1)).
            fillna(0).astype(int)
        f[f"{col}_risk_score"] = np.dot(w,r)/w.sum()
        f[f"{col}_ew_changes"] = np.dot(w,ch)
    return {"customer_ID": group["customer_ID"].iloc
        [0],
        **f,
        "target": group["target"].iloc[0]}

def main():
    path = "/mnt/.../train_with_labels.parquet"
    schema = pq.ParquetFile(path).schema
    all_cols = schema.names
    base_cols = ["customer_ID", "S_2", "target"]

```

```

cand = [c for c in all_cols if c not in
        base_cols+CAT_FEATURES
        and c.startswith(("B_", "D_", "S_", "P_", "R_
            ")))]
num_cols = cand[:MAX_NUMERICAL]
cat_cols = [c for c in CAT_FEATURES if c in
    all_cols][:MAX_CATEGORICAL]
df = pd.read_parquet(path, columns=base_cols+
    num_cols+cat_cols)
for c in num_cols: df[c] = df[c].fillna(0).
    astype(np.float16)
df_last = df.sort_values(["customer_ID", "S_2"]
    ).groupby("customer_ID").tail(1)
gmean = df_last["target"].mean()
risk_cols = []
for c in cat_cols:
    rm = df_last.groupby(c)["target"].mean()
    rc = f"{c}_risk"
    df[rc] = df[c].map(rm).fillna(gmean).astype(
        np.float16)
    risk_cols.append(rc)
df = df.sort_values(["customer_ID", "S_2"])
groups = [g for _,g in df.groupby("customer_ID")
    ]
res = Parallel(n_jobs=PARALLEL_WORKERS)(
    delayed(process_customer)(g,num_cols,
        cat_cols,risk_cols)
    for g in tqdm(groups))
pd.DataFrame(res).to_parquet("features.parquet",
    index=False)

if __name__ == "__main__":
    main()

```

C. Math

In this appendix, we provide the Python code of the best programs obtained by FM agent for the three mathematical tasks in Section 5.3. Their outputs reproduce the corresponding results shown in Section 5.3.

Listing 3: The best program found by FM agent for the task in Section 5.3.1

```

# Your rewritten program here
# EVOLVE-BLOCK-START
import numpy as np
from scipy.optimize import linprog

def solve_radii_lp(centers):
    """
    Calculates the maximum possible radii for a
    given set of circle centers
    using Linear Programming. This provides the
    exact optimal solution.

    Args:
        centers (np.array): An array of shape (n, 2)
            of circle center coordinates.

    Returns:
        np.array: An array of shape (n,) containing
            the optimal radius for each circle.
            Returns zeros if the solver fails.
    """
    n = centers.shape[0]

    # Objective function: Maximize sum(r_i), which
    # is equivalent to
    # minimizing sum(-1 * r_i).
    c = -np.ones(n)

    # We will build the constraints for A_ub * r <=
    # b_ub
    num_pair_constraints = n * (n - 1) // 2
    num_wall_constraints = 4 * n

```

```

num_constraints = num_pair_constraints +
    num_wall_constraints

A_ub = np.zeros((num_constraints, n))
b_ub = np.zeros(num_constraints)

# Constraint 1: Inter-circle non-overlap (r_i +
# r_j <= dist_ij)
row = 0
for i in range(n):
    for j in range(i + 1, n):
        dist = np.linalg.norm(centers[i] -
            centers[j])
        A_ub[row, i] = 1
        A_ub[row, j] = 1
        b_ub[row] = dist
        row += 1

# Constraint 2: Boundary non-overlap (r_i <=
# dist_to_wall)
for i in range(n):
    x, y = centers[i]
    # r_i <= x
    A_ub[row, i] = 1
    b_ub[row] = x
    row += 1
    # r_i <= 1 - x
    A_ub[row, i] = 1
    b_ub[row] = 1 - x
    row += 1
    # r_i <= y
    A_ub[row, i] = 1
    b_ub[row] = y
    row += 1
    # r_i <= 1 - y

```



```

        A_ub[row, i] = 1
        b_ub[row] = 1 - y
        row += 1

# Bounds for radii: r_i >= 0
bounds = (0, None)

# Solve the linear program
res = linprog(c, A_ub=A_ub, b_ub=b_ub, bounds=bounds, method='highs')

if res.success:
    return res.x
else:
    # Fallback in case the solver fails
    return np.zeros(n)

def construct_packing():
    """
    Constructs a high-quality arrangement of 26
    circles using a three-stage
    hybrid optimization strategy.

    1. **Geometric Initialization (The Builder):**
    - Instead of a simple grid, the circles are
      initialized in a staggered,
      geometrically-informed 5-6-5-6-4 hexagonal-
      like lattice. This provides
      a significantly better starting point that
      respects the principles of
      dense packing.

    2. **Staged Physical Simulation (The Explorer +
    Refiner):**
    - A gradient-based Adam optimizer simulates
      physical forces over two phases:
      a) **Exploration Phase (~70% of iterations)**
      : ** A strong, long-range
        repulsive force is applied between all
        circle centers. This force,
        which anneals over time, pushes circles
        apart to prevent premature
        clumping and helps the system discover a
        globally superior layout.
      b) **Refinement Phase (~30% of iterations)**
      : ** The repulsive force is
        disabled, and a powerful overlap penalty
        is aggressively ramped up.
        This forces the circles to settle into a
        precise, valid, and
        tightly-packed final configuration.

    3. **Linear Programming Polish (The Finisher):**
    - After the physical simulation finds a near-
      optimal set of center
      positions, the 'solve_radii_lp' function is
      called. This LP solver
      calculates the mathematically exact maximum
      radii for the final centers,
      guaranteeing a perfectly valid and optimal
      packing for that arrangement.
    """
    n = 26
    np.random.seed(42)

    # 1. Initialization: Staggered hexagonal lattice
    centers = []
    y_step = 0.175
    y_start = (1.0 - 4 * y_step) / 2.0 # Center the
    pattern vertically

    # Define the structure of the 5-6-5-6-4 lattice
    row_configs = [
        (5, (np.arange(5) + 1.5) / 7.0), # Staggered
        relative to a 6-circle row
        (6, (np.arange(6) + 1.0) / 7.0), # Main row
        (5, (np.arange(5) + 1.5) / 7.0), # Staggered
        (6, (np.arange(6) + 1.0) / 7.0), # Main row
        (4, (np.arange(4) + 2.0) / 7.0) # Staggered
        relative to the 6-circle row below

current_y = y_start
for _, (count, xs) in enumerate(row_configs):
    for x in xs:
        centers.append([x, current_y])
        current_y += y_step

centers = np.array(centers)
centers += (np.random.rand(n, 2) - 0.5) * 0.005
# Add smaller jitter

log_radii = np.full(n, np.log(0.05))

# 2. Adam Optimizer Parameters & State
N_iterations = 18000
initial_lr = 150 / 1e5
k_initial = float(150)
k_final = float(100000)
repulsion_initial = 10 / 1e7
exploration_phase_fraction = 50 / 100.0

final_lr = 1e-6
beta1, beta2, epsilon = 0.9, 0.999, 1e-8
m_centers, v_centers = np.zeros_like(centers),
np.zeros_like(centers)
m_log_radii, v_log_radii = np.zeros_like(
log_radii), np.zeros_like(log_radii)

# Penalty annealing (starts gentler, ends
stronger)
k_ratio = (k_final / k_initial) ** (1.0 /
N_iterations)
penalty_coeff = k_initial

# Staged repulsive force annealing
repulsion_final = 1e-9
exploration_phase_iterations = int(N_iterations
* exploration_phase_fraction)
repulsion_ratio = (repulsion_final /
repulsion_initial) ** (1.0 /
exploration_phase_iterations)
repulsion_coeff = repulsion_initial

# 3. Main Optimization Loop
for t in range(1, N_iterations + 1):
    radii = np.exp(log_radii)

    # --- Forward Pass: Calculate Overlaps &
    Distances ---
    diffs = centers[:, np.newaxis, :] - centers[
np.newaxis, :, :]
    dists_sq = np.sum(diffs**2, axis=-1)
    dists = np.sqrt(dists_sq + epsilon)

    radii_sums = radii[:, np.newaxis] + radii[np.
newaxis, :]
    inter_circle_overlaps = np.maximum(0,
radii_sums - dists)
    np.fill_diagonal(inter_circle_overlaps, 0)

    overlap_left = np.maximum(0, radii - centers
[:, 0])
    overlap_right = np.maximum(0, centers[:, 0] +
radii - 1)
    overlap_bottom = np.maximum(0, radii -
centers[:, 1])
    overlap_top = np.maximum(0, centers[:, 1] +
radii - 1)

    # --- Backward Pass: Calculate Gradients ---
    grad_log_radii_obj = -radii

    grad_term_inter = 2 * penalty_coeff *
inter_circle_overlaps
    grad_centers_inter = np.sum(-grad_term_inter
[... , np.newaxis] * (difs / (dists[... ,
np.newaxis])), axis=1)
    grad_log_radii_inter = np.sum(
grad_term_inter, axis=1) * radii

```

```

grad_centers_boundary_x = penalty_coeff *
    (-2 * overlap_left + 2 * overlap_right)
grad_centers_boundary_y = penalty_coeff *
    (-2 * overlap_bottom + 2 * overlap_top)
grad_log_rad_ii_boundary = 2 * penalty_coeff *
    (overlap_left + overlap_right +
    overlap_bottom + overlap_top) * radii

# Repulsive force gradient (from potential
energy U ~ sum(1/d))
grad_centers_repulsion = repulsion_coeff *
    np.sum(-(diffs / (dists**3))[..., np.
    newaxis]), axis=1)

# Total gradients
grad_centers = grad_centers_inter + np.stack
    ([grad_centers_boundary_x,
    grad_centers_boundary_y], axis=1) +
    grad_centers_repulsion
grad_log_rad_ii = grad_log_rad_ii_obj +
    grad_log_rad_ii_inter +
    grad_log_rad_ii_boundary

# --- Adam Optimizer Update with Cosine
Learning Rate Decay ---
t_ratio = t / N_iterations
decay_factor = 0.5 * (1 + np.cos(np.pi *
    t_ratio))
current_lr = final_lr + (initial_lr -
    final_lr) * decay_factor

m_centers = beta1 * m_centers + (1 - beta1) *
    grad_centers
v_centers = beta2 * v_centers + (1 - beta2) *
    (grad_centers**2)
m_hat_centers = m_centers / (1 - beta1**t)

```

```

v_hat_centers = v_centers / (1 - beta2**t)
centers -= current_lr * m_hat_centers / (np.
    sqrt(v_hat_centers) + epsilon)

m_log_rad_ii = beta1 * m_log_rad_ii + (1 -
    beta1) * grad_log_rad_ii
v_log_rad_ii = beta2 * v_log_rad_ii + (1 -
    beta2) * (grad_log_rad_ii**2)
m_hat_log_rad_ii = m_log_rad_ii / (1 - beta1**
    t)
v_hat_log_rad_ii = v_log_rad_ii / (1 - beta2**
    t)
log_rad_ii -= current_lr * m_hat_log_rad_ii /
    (np.sqrt(v_hat_log_rad_ii) + epsilon)

centers = np.clip(centers, 0.001, 0.999)

# Update annealing coefficients based on the
current phase
penalty_coeff *= k_ratio
if t < exploration_phase_iterations:
    repulsion_coeff *= repulsion_ratio
else:
    repulsion_coeff = 0.0 # End of
    exploration phase, turn off
    repulsion

# 4. Final Refinement with LP Solver
final_centers = centers
final_rad_ii = solve_rad_ii_lp(final_centers)
final_sum_rad_ii = np.sum(final_rad_ii)

return final_centers, final_rad_ii,
    final_sum_rad_ii
# EVOLVE-BLOCK-END

```

Listing 4: The best program found by FM agent for the task in Section 5.3.2

```

# EVOLVE-BLOCK-START
"""
Hermite polynomial coefficient optimization for
uncertainty inequality.

This program implements a state-of-the-art
optimization strategy by combining
symbolic pre-computation within an object-oriented
structure with a powerful
global optimization algorithm. This approach
addresses the critical performance
and accuracy limitations of previous methods.

Key Features:
1. **Hybrid Symbolic-Numeric Class Architecture:**
The core logic is encapsulated
in a class, 'HermiteOptimizer'. Its constructor
performs all slow, one-time
symbolic computations using 'sympy'. It then "
compiles" the resulting
mathematical expressions into highly efficient
numerical functions using
'sympy.lambdify', which are stored as instance
attributes. This design
combines symbolic precision with numerical speed and
avoids module-level
side effects.

2. **High-Performance Objective Function:** The
objective function is a class
method that operates purely on numerical data using
'numpy'. It leverages
the pre-compiled functions for instantaneous
calculation of polynomial
coefficients and uses the fast 'numpy.roots' solver.
This makes each
evaluation orders of magnitude faster than a
symbolic approach.

```

```

3. **Advanced Global Optimization with '
differential_evolution':** The simple
random search is replaced by 'scipy.optimize.
differential_evolution', a
robust algorithm for finding global minima. It is
fine-tuned with:
- **Informed Asymmetric Bounds:** The search space
is focused on the most
promising region, a proven technique from past
successes.
- **High-Precision Tuning:** Extremely tight
tolerances ('atol=1e-15') and
an integrated polishing step ('polish=True') ensure
the final result is
found with maximum accuracy.

4. **Correctness by Design:** The 'P(0) = 0'
constraint is embedded
symbolically, reducing the search space
dimensionality. The physical
constraint that 'P(x)' must be positive for large '|
x|' is correctly
handled by checking the leading coefficient's sign.'
"""
import numpy as np
import sympy
from scipy.optimize import differential_evolution

class HermiteOptimizer:
"""
Manages the optimization process by separating one-
time symbolic setup
from the fast, repetitive numerical evaluation.
"""

def __init__(self):
"""
Performs the one-time symbolic pre-computation.
"""
# 1. Define symbolic variables
x, c0, c1, c2 = sympy.symbols('x c0 c1 c2')

```

```

# 2. Define the even-order Hermite polynomials H_0,
H_4, H_8, H_12
hps_sym = [
    sympy.polys.orthopolys.hermite_poly(n, x=x, polys=
        False)
    for n in [0, 4, 8, 12]
]

# 3. Symbolically compute c_3 to enforce P(0) = 0
h_vals_at_0 = [hp.subs(x, 0) for hp in hps_sym]
p_partial_at_0 = c0 * h_vals_at_0[0] + c1 *
    h_vals_at_0[1] + c2 * h_vals_at_0[2]
c3_expr = -p_partial_at_0 / h_vals_at_0[3]

# 4. Construct the full polynomial P(x) symbolically
P_full_expr = c0*hps_sym[0] + c1*hps_sym[1] + c2*
    hps_sym[2] + c3_expr*hps_sym[3]

# 5. Get the quotient polynomial gq(x) = P(x)/x^2
    for root finding
gq_expr = sympy.exquo(P_full_expr, x**2)

# 6. Extract symbolic expressions for coefficients
    of gq(x) and P(x)
gq_poly_in_x = sympy.Poly(gq_expr, x)
P_full_poly_in_x = sympy.Poly(P_full_expr, x)

# 7. "Compile" symbolic expressions into fast
    numerical functions
self.gq_coeffs_func = sympy.lambdify([c0, c1, c2],
    gq_poly_in_x.all_coeffs(), 'numpy')
self.leading_coeff_P_func = sympy.lambdify([c0, c1,
    c2], P_full_poly_in_x.LC(), 'numpy')

def objective_function(self, coeffs: np.ndarray) ->
    float:
    """
    The fast, numerical objective function for the
    optimizer.
    """
    PENALTY = 1e12

# 1. Calculate numerical coefficients for gq(x)
try:
    poly_coeffs_gq = np.array(self.gq_coeffs_func(*
        coeffs), dtype=float)
except (ValueError, TypeError):
    return PENALTY

# 2. Enforce P(x) > 0 for large |x|
if self.leading_coeff_P_func(*coeffs) < 0:
    poly_coeffs_gq = -poly_coeffs_gq

# 3. Guard against degenerate polynomials
if abs(poly_coeffs_gq[0]) < 1e-12:
    return PENALTY

# 4. Find roots of gq(x) numerically
try:
    roots = np.roots(poly_coeffs_gq)

except np.linalg.LinAlgError:
    return PENALTY

# 5. Filter for positive real roots
real_roots = roots[np.isclose(roots.imag, 0)].real
positive_roots = real_roots[real_roots > 1e-9]

# 6. If no positive roots, A(f) = 0, the ideal
    global minimum
if positive_roots.size == 0:
    return 0.0

# 7. Objective is to minimize r_max^2 / (2*pi)
r_max = np.max(positive_roots)
return r_max**2 / (2 * np.pi)

def run_search():
    """
    Runs the optimization using a fine-tuned
    Differential Evolution strategy.
    """
    # Instantiate the optimizer to perform the one-time
    symbolic setup
    optimizer = HermiteOptimizer()

    # Define asymmetric search bounds based on problem
    knowledge
    bounds = [(-5.0, 5.0), (-1.0, 1.0), (-0.1, 0.1)]

    # Run the Differential Evolution optimizer
    result = differential_evolution(
        func=optimizer.objective_function,
        bounds=bounds,
        strategy='best1bin',
        maxiter=300,
        popsize=30,
        tol=1e-10,
        atol=1e-15, # Critical for high-precision results
        recombination=0.7,
        seed=42,
        polish=True,
        workers=1 # Ensures compatibility and avoids
            pickling errors
    )

    best_coeffs = result.x
    best_value = result.fun

    return best_coeffs[0], best_coeffs[1], best_coeffs
        [2], best_value

# EVOLVE-BLOCK-END
if __name__ == "__main__":
    print("Starting optimization search...")
    coeff1, coeff2, coeff3, value = run_search()
    print("\nOptimization finished.")
    print(f"Found optimal coefficients: ({coeff1:.8f}, {
        coeff2:.8f}, {coeff3:.8f})")
    print(f"Minimized upper bound value: {value:.12f}")

```

Listing 5: The best program found by FM agent for the task in Section 5.3.3

```

"""
A high-performance program for constructing optimal
16-point configurations
in 2D space by minimizing the ratio of maximum to
minimum pairwise distance.

This program solves a smooth, constrained
optimization problem reformulated from the
original non-smooth ratio objective. It minimizes
the maximum squared distance,
subject to the constraint that the minimum squared
distance is at least 1.

Key improvements include:
- A fully vectorized Jacobian for the constraint
    function, providing a significant
    performance boost over iterative calculations.
- Correction of a variable scope bug present in the
    previous version.
- An expanded multi-start strategy with an
    additional initial configuration to
    more robustly explore the solution space.
- Increased optimizer iterations to achieve higher
    precision.
"""
import numpy as np
from scipy.optimize import minimize
from scipy.spatial.distance import pdist
from itertools import combinations

def calculate_ratio(points):

```

```

"""Calculate the ratio between maximum and minimum
pairwise distances."""
if points is None or len(points) < 2:
    return float('inf')
distances = pdist(points)
if len(distances) == 0:
    return float('inf')
d_max = np.max(distances)
d_min = np.min(distances)
if d_min < 1e-9: # Treat very small distances as
    zero to avoid instability
return float('inf')
return d_max / d_min

def construct_16_points():
    """
    Construct 16 points in 2D space to minimize the
    ratio d_max/d_min
    using a multi-start constrained optimization
    approach (SLSQP) with a
    highly efficient vectorized Jacobian.
    """
    N_POINTS = 16
    N_VARS = N_POINTS * 2
    best_points = None
    best_ratio_sq = float('inf')

    # --- Pre-compute indices for vectorization ---
    # This is done once to speed up the Jacobian
    calculation inside the solver loop.
    indices = list(combinations(range(N_POINTS), 2))
    N_PAIRS = len(indices)
    I = np.array([i for i, j in indices])
    J = np.array([j for i, j in indices])

    # --- Initial Configurations ---
    # A diverse set of starting points is crucial for
    finding a good global minimum.

    # Config 1: Hexagonal Lattice Section
    points_hex = []
    sqrt3_div_2 = np.sqrt(3) / 2.0
    for v_idx in range(4):
        for u_idx in range(4):
            x = (u_idx - 1.5) + 0.5 * (v_idx - 1.5)
            y = (v_idx - 1.5) * sqrt3_div_2
            points_hex.append([x, y])
    initial_hex = np.array(points_hex)

    # Config 2: 1-5-10 Concentric Ring Structure (known
    to be near-optimal)
    points_1_5_10 = [[0, 0]]
    r1 = 1.0
    for i in range(5):
        angle = i * 2 * np.pi / 5
        points_1_5_10.append([r1 * np.cos(angle), r1 * np.
            sin(angle)])
    r2 = 1.992 # Fine-tuned radius based on known good
    solutions
    initial_rotation = np.pi / 10
    for i in range(10):
        angle = i * 2 * np.pi / 10 + initial_rotation
        points_1_5_10.append([r2 * np.cos(angle), r2 * np.
            sin(angle)])
    initial_1_5_10 = np.array(points_1_5_10)

    # Config 3: Two-Ring Structure (6-10)
    points_6_10 = []
    for i in range(6):
        angle = i * np.pi / 3
        points_6_10.append([1.0 * np.cos(angle), 1.0 * np.
            sin(angle)])
    for i in range(10):
        angle = i * 2 * np.pi / 10 + np.pi/10
        points_6_10.append([1.9 * np.cos(angle), 1.9 * np.
            sin(angle)])
    initial_6_10 = np.array(points_6_10)

    # Config 4: 4x4 Grid
    points_grid = []
    for i in range(4):
        for j in range(4):
            points_grid.append([i - 1.5, j - 1.5])
    initial_grid = np.array(points_grid)

    # Config 5: Random Start
    np.random.seed(42)
    initial_random = np.random.rand(N_POINTS, 2) * 5 -
        2.5

    initial_configs = {
        "hexagonal": initial_hex,
        "concentric_1_5_10": initial_1_5_10,
        "concentric_6_10": initial_6_10,
        "grid_4x4": initial_grid,
        "random": initial_random,
    }

    # --- Setup for SLSQP Optimization ---
    # The optimization variable 'x' is a flat array: [x1
    , y1, ..., x16, y16, D_max_sq]
    objective_func = lambda x: x[-1]
    def objective_jac(x):
        grad = np.zeros_like(x)
        grad[-1] = 1.0
        return grad

    # --- Vectorized Constraint and Jacobian Functions
    ---
    # Defined within this scope to have access to N_VARS
    , N_POINTS, etc.
    def constraints_func(x):
        points = x[:N_VARS].reshape(N_POINTS, 2)
        D_max_sq = x[-1]
        sq_dists = pdist(points, 'sqeuclidean')
        # c1: d_ij^2 >= 1 => d_ij^2 - 1 >= 0
        c1 = sq_dists - 1.0
        # c2: d_ij^2 <= D_max_sq => D_max_sq - d_ij^2 >= 0
        c2 = D_max_sq - sq_dists
        return np.concatenate((c1, c2))

    def constraints_jac(x):
        points = x[:N_VARS].reshape(N_POINTS, 2)
        jac = np.zeros((2 * N_PAIRS, N_VARS + 1))
        # Calculate all 2*(pi - pj) vectors in a single
        operation
        diffs = 2 * (points[I] - points[J])
        # Row indices for the first block of constraints
        k = np.arange(N_PAIRS)
        # Populate Jacobian for c1 constraints (d_ij^2 - 1)
        using vectorized assignment
        jac[k, 2 * I] = diffs[:, 0]
        jac[k, 2 * I + 1] = diffs[:, 1]
        jac[k, 2 * J] = -diffs[:, 0]
        jac[k, 2 * J + 1] = -diffs[:, 1]
        # Populate Jacobian for c2 constraints (D_max_sq -
        d_ij^2)
        jac[k + N_PAIRS, 2 * I] = -diffs[:, 0]
        jac[k + N_PAIRS, 2 * I + 1] = -diffs[:, 1]
        jac[k + N_PAIRS, 2 * J] = diffs[:, 0]
        jac[k + N_PAIRS, 2 * J + 1] = diffs[:, 1]
        # Derivative of c2 with respect to D_max_sq is 1
        jac[N_PAIRS:, -1] = 1.0
        return jac

    cons = {'type': 'ineq', 'fun': constraints_func, '
        jac': constraints_jac}
    optimizer_options = {'maxiter': 3000, 'ftol': 1e-12,
        'disp': False}

    for name, config in initial_configs.items():
        initial_points = config.copy()
        dists = pdist(initial_points)
        min_dist = np.min(dists)
        if min_dist > 1e-9:
            initial_points /= min_dist
        initial_d_max_sq = np.max(pdist(initial_points)**2)
        x0 = np.append(initial_points.flatten(),
            initial_d_max_sq)
        result = minimize(
            objective_func,
            x0,

```

```

method='SLSQP',
jac=objective_jac,
constraints=cons,
options=optimizer_options
)
if result.success:
    current_ratio_sq = result.fun
    if current_ratio_sq < best_ratio_sq:
        best_ratio_sq = current_ratio_sq
    best_points = result.x[:N_VARS].reshape(N_POINTS, 2)
    if best_points is None:
        best_points = initial_1_5_10
    return best_points

def run_construction():
    """Main function that runs the construction and
    returns results."""
    try:
        points = construct_16_points()
        if points is not None:
            # Center and normalize the final configuration for a
            canonical representation
            points -= np.mean(points, axis=0)
            min_dist = np.min(pdist(points))
            if min_dist > 1e-9:
                points /= min_dist
            return points
        except Exception as e:
            print(f"An error occurred during construction: {e}")
            return None

    if __name__ == "__main__":

        points = run_construction()
        if points is not None:
            ratio = calculate_ratio(points)
            ratio_squared = ratio**2
            print(f"Achieved ratio squared: {ratio_squared:.20f}
            ")
            target_sq = 12.889266112
            print(f"Target ratio squared: {target_sq:.20f} (
            ratio = {np.sqrt(target_sq):.20f})")
            if ratio_squared < target_sq:
                print("\nSuccess: Target beaten!")
            else:
                print("\nFailure: Target not beaten.")
            else:
                print("Construction failed.")

#@title Construction verification
import scipy as sp

if points is not None:
    print(f'\nConstruction has {len(points)} points in {
    points.shape[1]} dimensions.')
    pairwise_distances = sp.spatial.distance.pdist(
        points)
    min_distance = np.min(pairwise_distances)
    max_distance = np.max(pairwise_distances)

    final_ratio_squared = (max_distance / min_distance)
    **2
    print(f"Ratio of max distance to min distance: sqrt
    ({final_ratio_squared:.20f})")

```

Table 5: FM Agent surpasses all baseline across every evaluation dimension defined in MLE-Bench. All values represent competition-specific metrics (consistent with official MLE-Bench definitions). The results for MLAB(gpt-4o-2024-08-06), OpenHands(gpt-4o-2024-08-06), AIDE(o1-preview), R&D-Agent(gpt-5), ML-Master(deepseek-r1), Neo multi-agent, InterAgent(deepseek-r1) and Operand ensemble(gpt-5, low verbosity/effort) are taken from the official MLE-Bench report. Results for FM Agent are averaged over three independent runs with different random seeds and are presented as the mean \pm one standard error of the mean (SEM). The best-performing model in each category is highlighted in bold.(Continued on next page)

Competition	FM Agent	Operand ensemble	InternAgent [31]	R&D-Agent [30]	Neo multi-agent	ML-Master [29]
aerial-cactus-identification	1.0	1.0	1.0	1.0	1.0	1.0
aptos2019-blindness-detection	0.93556	0.9039	0.92125	0.92034	0.92498	0.92954
denoising-dirty-documents*	0.01311	0.01081	0.0116	0.01009	0.00742	0.00763
detecting-insults-in-social-commentary	0.95712	0.9128	0.94484	0.94918	<i>None</i>	0.94838
dog-breed-identification*	0.3502	0.53653	0.30026	<i>None</i>	0.43795	0.32345
dogs-vs-cats-redux-kernels-edition*	0.00945	0.02172	0.00287	0.0117	0.00792	0.00309
histopathologic-cancer-detection	0.99393	0.93029	0.99833	0.99604	0.99521	0.99754
jigsaw-toxic-comment-classification-challenge	0.98694	0.98047	0.98732	0.98661	0.98713	0.9864
leaf-classification*	0.11865	0.00883	0.01856	0.00158	0.2459	0.01661
mlsp-2013-birds	0.91413	0.90827	0.88582	0.90731	0.93904	0.78133
new-york-city-taxi-fare-prediction*	3.96969	4.69105	5.75886	<i>None</i>	<i>None</i>	6.125
nomad2018-predict-transparent-conductors*	0.05259	0.06018	0.0585	0.05833	0.05978	0.059
plant-pathology-2020-fgvc7	0.99957	0.98971	0.99688	0.99818	0.99752	0.99041
random-acts-of-pizza	0.7588	0.70063	0.67654	0.79623	0.79216	0.6516
ranzcr-clip-catheter-line-classification	0.96028	0.95833	0.931	<i>None</i>	0.95585	0.95979
siim-isic-melanoma-classification	0.9279	0.76117	0.9414	<i>None</i>	0.84593	0.91251
spooky-author-identification*	0.23578	0.27869	0.21544	0.22926	0.23527	0.26013
tabular-playground-series-dec-2021	0.96137	0.96335	0.96302	0.96312	0.95998	0.96302
tabular-playground-series-may-2022	0.99566	0.69736	0.9945	<i>None</i>	0.99296	0.99463
text-normalization-challenge-english-language	0.99059	0.99221	0.99053	0.99283	0.95377	0.99182
text-normalization-challenge-russian-language	0.97021	0.97968	0.97924	0.98277	0.98304	0.97348
the-icml-2013-whale-challenge-right-whale-redux	0.99391	0.94555	0.99413	0.99261	<i>None</i>	0.99082
A14Code	0.503	<i>None</i>	0.62471	<i>None</i>	0.40432	0.72299
alaska2-image-steganalysis	0.86242	<i>None</i>	0.62477	<i>None</i>	0.61367	0.76772
billion-word-imputation*	6.69267	<i>None</i>	<i>None</i>	<i>None</i>	7.0074	<i>None</i>
cassava-leaf-disease-classification	0.9006	0.89948	0.89649	0.88976	0.89126	0.89163
cdiscout-image-classification-challenge	0.72038	<i>None</i>	0.64694	<i>None</i>	0.65588	0.65574
chaii-hindi-and-tamil-question-answering	0.7433	0.7189	0.27026	<i>None</i>	0.72366	0.62283
champs-scalar-coupling*	1.23566	1.99777	1.43002	<i>None</i>	1.15637	1.7062
facebook-recruiting-iii-keyword-extraction	0.57231	0.1991	0.52234	<i>None</i>	0.53506	0.4469
freesound-audio-tagging-2019	0.69921	0.68063	0.71638	0.68569	0.59279	0.68586
google-quest-challenge	0.41448	0.4105	0.41797	0.42004	0.43395	0.41889
h-and-m-personalized-fashion-recommendations	0.02714	<i>None</i>	0.02456	0.0	0.02411	0.02518
herbarium-2020-fgvc7	0.47074	0.09606	0.38989	0.45351	0.44275	0.23544
herbarium-2021-fgvc8	0.32257	0.34083	0.28441	0.23133	0.43683	0.37574
herbarium-2022-fgvc9	0.77877	0.65859	0.48576	0.31697	0.7735	0.61428
hotel-id-2021-fgvc8	0.44257	0.22754	0.70941	0.49623	0.43788	0.61294
hubmap-kidney-segmentation	0.0	<i>None</i>	0.92562	0.9991	0.05064	0.04435
icecube-neutrinos-in-deep-ice*	1.53458	<i>None</i>	1.51821	<i>None</i>	<i>None</i>	1.55861
imet-2020-fgvc7	0.60932	0.27955	0.62573	<i>None</i>	0.5865	0.60408
inaturalist-2019-fgvc6*	0.99814	0.13767	0.13544	0.21165	0.19454	0.18964
iwildcam-2020-fgvc7	0.83522	0.73007	0.8288	0.70202	0.74062	0.82057
jigsaw-unintended-bias-in-toxicity-classification	0.80032	<i>None</i>	0.78648	<i>None</i>	0.81502	0.82847
kuzushiji-recognition	0.68409	0.72021	0.58578	0.83033	0.95238	0.62457
learning-agency-lab-automated-essay-scoring-2	0.84839	0.83013	0.83995	0.83751	0.83881	0.82104
lmsys-chatbot-arena*	1.00886	<i>None</i>	1.00457	<i>None</i>	0.99092	1.05199
multi-modal-gesture-recognition*	0.86342	0.5383	0.6872	<i>None</i>	0.86824	0.90841
osic-pulmonary-fibrosis-progression	-7.62814	-7.19604	-7.22947	<i>None</i>	-8.52062	<i>None</i>
petfinder-pawpularity-score*	17.47187	16.89225	18.19441	<i>None</i>	18.77071	17.80862
plant-pathology-2021-fgvc8	0.9226	0.93141	0.91868	0.91845	0.9156	0.93039
seti-breakthrough-listen	0.86216	0.52149	0.84358	0.77904	0.79164	0.83485
statoil-iceberg-classifier-challenge*	0.6963	<i>None</i>	0.19655	<i>None</i>	0.22671	0.24349
tensorflow-speech-recognition-challenge	0.33647	<i>None</i>	0.3527	<i>None</i>	0.35316	0.34837
tensorflow2-question-answering	0.57823	<i>None</i>	0.57117	<i>None</i>	0.57117	0.2207
tgis-salt-identification-challenge	0.7127	<i>None</i>	0.7927	<i>None</i>	0.7831	0.5221
tweet-sentiment-extraction	0.71906	0.71958	0.64334	0.71159	0.75393	0.70943
us-patent-phrase-to-phrase-matching	0.87466	0.84507	0.87046	0.84449	0.83287	0.85901
uw-madison-gi-tract-image-segmentation	0.75471	<i>None</i>	0.83297	<i>None</i>	0.44372	0.10015
ventilator-pressure-prediction*	0.22428	17.65486	0.47338	<i>None</i>	0.90693	0.33547
whale-categorization-playground	0.50367	0.43053	0.31698	0.27506	0.42718	0.2423

* Indicates minimization tasks, where smaller values represent better performance.

None Indicates that either no valid submission file was generated or the file format was incorrect.

Table 5: FM Agent surpasses all baseline models across every evaluation dimension defined in MLE-Bench. All values represent competition-specific metrics (consistent with official MLE-Bench definitions). The results for MLAB(gpt-4o-2024-08-06), OpenHands(gpt-4o-2024-08-06), AIDE(o1-preview), R&D-Agent(gpt-5), ML-Master(deepseek-r1), Neo multi-agent, InterAgent(deepseek-r1) and Operand ensemble(gpt-5, low verbosity/effort) are taken from the official MLE-Bench report. Results for FM Agent are averaged over three independent runs with different random seeds and are presented as the mean \pm one standard error of the mean (SEM). The best-performing model in each category is highlighted in bold.

Competition	FM Agent	Operand ensemble	InterAgent [31]	R&D-Agent [30]	Neo multi-agent	ML-Master [29]
3d-object-detection-for-autonomous-vehicles	0.0	<i>None</i>	0.0	0.01299	0.0	0.0
bms-molecular-translation*	42.11198	<i>None</i>	73.919	<i>None</i>	89.48078	96.89238
google-research-identify-contrails-reduce-global-warming	0.01126	<i>None</i>	0.07186	<i>None</i>	0.05841	0.42954
hms-harmful-brain-activity-classification*	0.65027	<i>None</i>	0.75958	<i>None</i>	0.83488	0.93875
iwildcam-2019-fgvc6	0.37797	0.39804	0.29297	0.4711	0.47775	0.48054
nfl-player-contact-detection	0.64045	<i>None</i>	0.60575	<i>None</i>	0.0604	0.54446
predict-volcanic-eruptions-ingv-oe*	3705196.0	2188576	2499380.0	2044627.0	2902638.0	2826639.0
rsna-2022-cervical-spine-fracture-detection*	0.59289	0.69315	0.56419	<i>None</i>	0.5639	0.5723
rsna-breast-cancer-detection	0.08437	0.06863	0.06232	<i>None</i>	0.04624	0.04901
rsna-miccai-brain-tumor-radiogenomic-classification	0.66059	<i>None</i>	0.59647	0.58588	0.60941	0.61176
siim-covid19-detection	0.44507	0.30458	0.42364	<i>None</i>	0.19073	0.36954
smartphone-decimeter-2022*	5.9367	<i>None</i>	6.09075	<i>None</i>	3122773.6564	15642.40004
stanford-covid-vaccine*	0.31981	0.31841	0.22919	0.24187	0.28491	0.23945
vesuvius-challenge-ink-detection	0.48622	0.20266	0.19222	<i>None</i>	0.14406	0.12163
vinbigdata-chest-xray-abnormalities-detection	0.29848	<i>None</i>	<i>None</i>	<i>None</i>	0.02371	<i>None</i>

* Indicates minimization tasks, where smaller values represent better performance.

None Indicates that either no valid submission file was generated or the file format was incorrect.