When LLM Agents Encounter MLE Automation: A Comprehensive Review of Challenges and Future Prospects

Zucan Lyu(吕祖灿)
PB2311729

摘 要—The integration of Large Language Model(LLM) agents with Machine Learning Engineering(MLE) Automation has emerged as a core pillar of the AI-for-AI(AI4AI) paradigm, aiming to shift from human-led to autonomous AI development. Despite rapid advancements, existing research remains fragmented across AutoML, coding agents, and specialized MLE systems, lacking a unified synthesis of foundations, state-of-the-art progress, and future directions. This review addresses this gap by providing the first comprehensive analysis of LLM agents for MLE Automation. We first formalize foundational definitions, framing the MLE environment as a goal-conditioned Partially Observable Markov Decision Process(POMDP) and defining agent architectures with modular components of LLMs, context management, and tool integration. A three-tier taxonomy is proposed to categorize existing systems: pre-LLM AutoML(isolated stage optimization), general-purpose coding agents(broad software engineering capabilities with limited ML domain expertise), and specialized MLE agents(domain-tailored systems with structured exploration and lifecycle awareness). We then systematically compare evaluation benchmarks across software engineering, MLE-specific, and general-purpose agent domains, highlighting their complementary roles and gaps in real-world adaptability. Critical challenges in current systems are identified, including rigid framework structures, inadequate tool context protocols, deficient process-aware memory, and limited self-evolving capabilities. Corresponding future directions are outlined, such as adaptive graph-based architectures, semantic tool interoperability, and meta-reasoning-driven self-improvement. Finally, we articulate MLE Automation as a pragmatic stepping stone toward AI4AI, mapping a clear pathway from assistant-like automation to fully autonomous AI-driven AI development. This work unifies fragmented research, establishes consistent terminology, and provides a structured roadmap to guide future innovations in this emerging field.

***Index Terms*—LLM Agents, Machine Learning, Data Science, Survey**

I. Introduction

An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future[1]

–Franklin and Graesser (1997)

The rapid advancement of Large Language Models (LLMs) has revolutionized the landscape of autonomous agents, endowing them with unprecedented capabilities in natural language reasoning, coding, and task planning [37][11]. Concurrently, Machine Learning Engineering(MLE)—the discipline of designing, deploying, and maintaining end-to-end ML systems—has evolved into a cornerstone of AI advancement, with workflows spanning data preprocessing, model design, hyperparameter tuning, deployment, and monitoring,

[1]Franklin, S., & Graesser, A. C. (1997). Is it an agent, or just a program?: A taxonomy for autonomous agents. *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages (ATAL)*, 21-35.

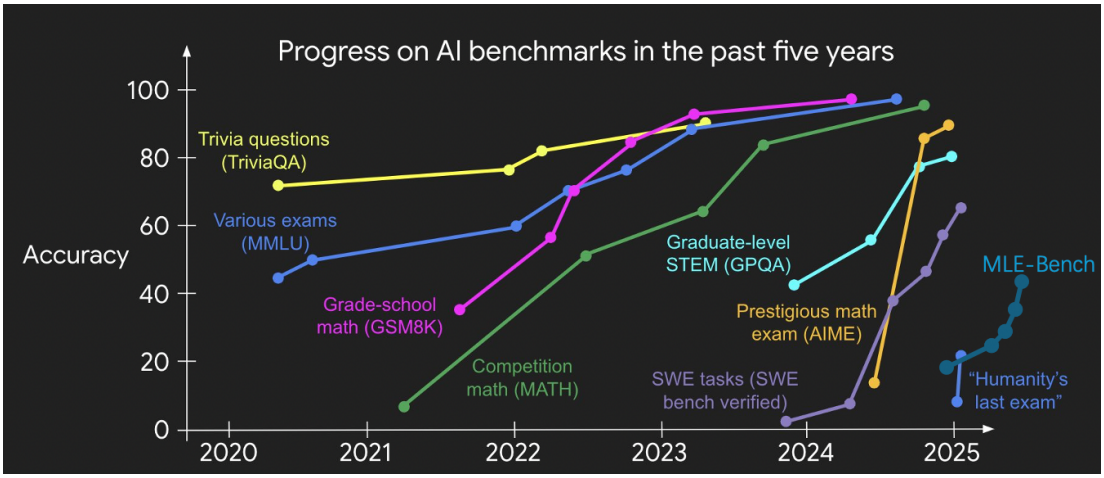

图 1. Positioned at the frontier of AI-for-AI progress, within several months, multiple methods boosted the performance of MLE-Bench, one of the most authoritative autonomous machine learning engineering benchmarks, highlighting the rapidly evolving path toward AI4AI. This picture is modified from [55]

as shown in Figure 1. The core driver behind MLE Automation lies in the broader pursuit and experimentation of AI-for-AI (AI4AI)—a paradigm where AI systems autonomously optimize, advance, and scale AI development itself. This pursuit has spurred the exploration of LLM-based agents as a transformative solution for MLE Automation: unlike traditional rule-based or narrow AI tools, LLM agents possess the unique ability to understand open-ended tasks, generate executable code, and interact with domain-specific tools, making them ideal for transcending fragmented automation and realizing end-to-end MLE workflows. While the inherent complexity of modern MLE pipelines (e.g., dynamic data distributions, multi-stage trade-offs) underscores the practical need for such automation, it is the vision of AI4AI that fundamentally motivates the integration of LLM agents into MLE, aiming to shift from human-led AI development to autonomous, self-evolving AI systems.

Despite the growing body of work at this intersection, existing research lacks a comprehensive synthesis that unifies the foundations, state-of-the-art, evaluation benchmarks, and future directions of LLM agents for MLE Automation. Pre-LLM AutoML frameworks focused on isolated pipeline stages rather than holistic workflows, while general-purpose coding agents lack MLE-specific domain knowledge. Specialized MLE agents, though advancing rapidly, are scattered across diverse exploration strategies and evaluation settings, with no systematic analysis of their convergent patterns and inherent limitations. Additionally, the connection between MLE Automation and the broader vision of AI4AI remains underexplored in existing literature. These gaps highlight the need for a structured review to consolidate insights, identify critical challenges, and guide future research in this emerging field.

This review aims to fill these gaps by providing a comprehensive analysis of LLM agents for MLE Automation. Its scope encompasses four core dimensions: (1) foundational definitions tailored to the MLE context, including formalizations of agent systems and their environments; (2) a taxonomy and critical review of existing LLM-based agents for MLE, spanning pre-LLM AutoML, general-purpose coding agents, and specialized MLE agents; (3) a systematic comparison of evaluation benchmarks, covering software engineering, MLE-specific, and general-purpose agent benchmarks; and (4) an in-depth discussion of current challenges and future prospects, including the transition from MLE Automation to AI4AI.

The remainder of this paper is organized as follows: Section II establishes key definitions and foundational frameworks for MLE environments and agent systems. Section III reviews existing LLM agents for MLE Au-

tomation, categorizing them by their design paradigms and analyzing their strengths and limitations. Section IV surveys relevant evaluation benchmarks, comparing their scope and suitability for assessing MLE agents. Section V discusses core challenges in current systems and outlines future research directions, including the pathway from MLE Automation to AI4AI. Finally, Section VI concludes the review.

This work makes four primary contributions:

1) It provides the first comprehensive review of LLM agents for MLE Automation, unifying fragmented research across AutoML, coding agents, and MLE-specific systems;
2) It formalizes foundational definitions for MLE-oriented agent environments and architectures, establishing a consistent terminology for the field;
3) It systematically compares existing evaluation benchmarks, highlighting their complementary roles and gaps in assessing real-world MLE capabilities;
4) It articulates a clear pathway from MLE Automation to AI4AI, identifying critical research milestones to realize fully autonomous AI-driven AI development.

II. Definitions and Foundations

Taxonomy(Figure 2) Before diving into a comprehensive survey, we first put forward a dual taxonomy to structure the landscape of LLM agents for MLE automation, encompassing both agent design paradigms and benchmark evaluation categories. This taxonomy not only clarifies the distinctions between different agent types but also aligns them with appropriate evaluation frameworks, laying a foundation for understanding the unique characteristics and assessment needs of MLE agents.

Machine Learning Engineering(MLE) refers to the discipline focused on designing, implementing, deploying, and maintaining end-to-end machine learning systems. It integrates algorithmic research, software engineering, and operational practices to transform theoretical ML models into reliable, scalable, and production-ready solutions that address real-world tasks.

MLE Automation refers to the process of leveraging intelligent systems (e.g., agent systems, automated pipelines) to streamline, optimize, and autonomously execute key stages of machine learning engineering workflows. This includes but is not limited to automated data preprocessing, model selection, hyperparameter tuning, deployment orchestration, performance monitoring, and iterative refinement. The goal of MLE automation is to reduce manual intervention, accelerate development cycles, enhance reproducibility, and enable scalable maintenance of ML systems while adapting to dynamic data distributions and evolving task requirements.

Environment We first define the environment of an agent system as a partially observable Markov Decision Process (POMDP), represented as a tuple $E = (\mathcal{G}, \mathcal{S}, \mathcal{A}, T, R, \Omega, O, \gamma)$ where each component is tailored to machine learning engineering (MLE) as follows(Modified from[4]):

- $\mathcal{G}$ is a set of MLE task goals. Each $g \in \mathcal{G}$ ties to predefined evaluation metrics and task types (e.g., "binary classification minimizing F1-score").
- $\mathcal{S}$ is a set of states. Each $s \in \mathcal{S}$ represents the internal state of the environment.
- $\mathcal{A}$ is a set of actions. Each action $a \in \mathcal{A}$ can be a combination of textual reasoning, retrieval of external knowledge, and tool calls.
- T is the state transition probability function which takes a state-action pair (s, a) and outputs the probability distribution $T(s' \mid s, a)$ of the next state.
- $R : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \to \mathcal{R}$ is the feedback/reward function, conditioned on the specific goal $g \in \mathcal{G}$. The feedback $r = R(s, a, g)$ typically takes the form of a scalar score or textual feedback.
- Ω is a set of observations accessible to the agent.
- O is the observation probability function which takes a state-action pair (s, a) and outputs the probability distribution $O(o' \mid s, a)$ of the next observation for the agent.

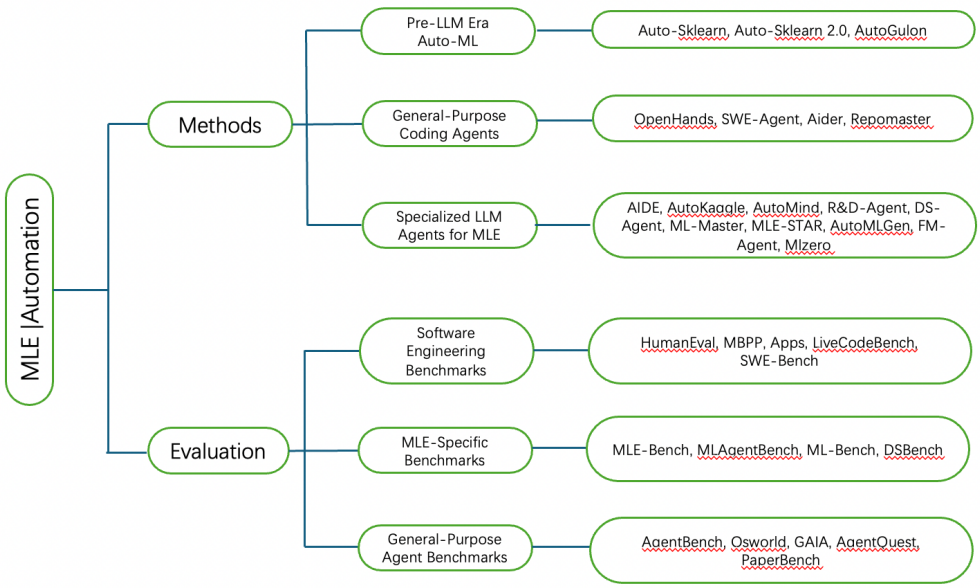

图 2. Taxonomy of MLE Autoamtion

- γ is the discount factor.

Agent system We define a (multi-)agent system as $\Pi = (\Gamma, \{\psi_i\}, \{C_i\}, \{\mathcal{W}_i\})$. The architecture Γ determines the control flow of the agent system or collaborative structures between multiple agents. It is typically represented as a sequence of nodes ($N_1, N_2, \ldots$) organized by graph or code structures. Each node N_i consists of the following components:

- ψ_i : the underlying LLM/MLLM.
- C_i : the context information, e.g., prompt P_i and memory M_i.
- $\mathcal{W}_i$: the set of available tools/APIs.

At each node, the agent policy is a function $\pi_{\theta_i}(\cdot \mid o)$ that takes an observation and outputs the probability distribution of the next action, where $\theta_i = (\psi_i, C_i)$. The actual action space here is the union of the natural language space and the tool space $\mathcal{W}_i$. For a given task $\mathcal{T} = (E, g)$, represented by an environment E and a corresponding goal $g \in \mathcal{G}$, the agent system follows the topology Γ to generate a trajectory $\tau = (o_0, a_0, o_1, a_1, \ldots)$, and receives a feedback r either from the external environment or from internal signals (e.g., self-confidence or feedback from an evaluator).

III. LLM Agents for MLE Automation

Benefiting from the increasing capability in coding and task planning, Large Language Models (LLMs)[37][11] are shifting from simple code assistants[40][20] to autonomous agents of sophisticated Machine Learning Engineering (MLE)[3][7]. In the realm of MLE, LLM agents are required to enhance specific metrics for the given task by iteratively optimizing code, which requires a comprehensive consideration of various factors such as data, model architectures, and so on.

A. Pre-LLM Era Auto-ML

In the pre-LLM era, while development of AutoML[18][14] has brought about significant progress in optimizing discrete stages such as data processing, they often fall short of managing the entire end-to-end MLE workflow (i.e., from data preparation to model training and inference).

Automl[18] systematically surveyed the state-of-the-art(SOTA) at that time of AutoML, summarizing its core value in automating repetitive, parameter-intensive tasks (e.g., hyperparameter optimization, feature engineering) via techniques like Bayesian optimization and evolutionary algorithms, which greatly re-

duced manual labor in model development. [14] further advanced this direction with Auto-Sklearn 2.0, a meta-learning-enhanced framework that leveraged historical pipeline performance data to improve optimization efficiency, enabling hands-free automation of classical machine learning pipelines for tabular data.

Despite these advancements, both works reflected the inherent limitations of pre-LLM AutoML: they were confined to optimizing isolated stages or predefined pipeline scopes, lacked adaptability to complex domain-specific workflows (e.g., custom NLP/vision tasks), and relied on manual definition of problem boundaries. These gaps laid the groundwork for the emergence of LLM agents, which, equipped with reasoning capabilities and coding proficiency, aim to transcend fragmented automation and enable more holistic, adaptive MLE workflows.

B. General-Purpose Coding Agents

Recent advances in Large Language Models have led to the development of powerful LLM-based agents[2][49][48] designed to tackle general software engineering tasks. Most early LLM-based agents were designed as general coding assistants, providing a flexible architecture without domain-specific tuning. For example, OpenHands[49] integrates LLM reasoning with tool use for complex software engineering tasks. SWE-Agent[53] offers comprehensive command sets for navigating codebases and implementing solutions, achieving notable performance on software engineering benchmarks.

However, despite their versatility in general software engineering scenarios, these general-purpose coding agents have inherent limitations when applied to specialized MLE tasks. They often lack domain-specific knowledge of machine learning workflows, struggle to prioritize MLE-critical optimizations (e.g., model generalization, computational efficiency), and tend to treat coding tasks as isolated implementations rather than iterative optimization processes tailored to ML metrics. These gaps highlight the need for specialized LLM agents designed explicitly for MLE, which combine coding proficiency with deep ML domain expertise to address the unique challenges of end-to-end machine learning engineering workflows.

C. Specialized LLM Agents for MLE

To address the unique challenges of machine learning engineering, a dedicated class of coding agents has been developed[24][35][13][32][22][16], which leverage the planning and execution capabilities of LLMs to optimize task performance across a broader search space, with many evaluated on comprehensive benchmarks like MLE-Bench[8]. The comparative results of these agents on MLE-Bench are presented in Table I. These agents primarily frame the problem as a search for an optimal code-based solution.

As a pioneer, AIDE[24] reformulates the exploration process of optimizing codes as a tree search, employ a greedy search strategy, and achieves gold medals in some Kaggle competitions. However, it can be susceptible to local optima. To overcome this, subsequent frameworks have adopted more sophisticated exploration strategies. Multi-agent collaboration approaches like AutoKaggle[30] distribute tasks among specialized agents. Tree search has also emerged as a dominant paradigm. AutoMind[39] introduces an agentic tree search grounded by an expert knowledge base, while R&D-Agent[54] manages parallel exploration traces, iteratively refines codes through the cooperation of the researcher agent and the developer agent. DS-Agent[16] uses case-based reasoning[28][50] to discover strategies for solution generation by utilizing manually curated cases (primarily from Kaggle). ML-Master[32] introduces a selectively scoped memory mechanism and standard MCTS to integrate exploration and reasoning. MLE-STAR[35] utilize a search engine to retrieve effective models and then explore various strategies targeting specific ML pipeline components to improve the solution. FM-Agent leverages a synergistic combination of LLM-based reasoning and large-scale evolutionary search. AutoMLGen[12] combines a curated ML knowledge base with Monte Carlo Graph Search (MCGS) to address key limitations of current MLE approaches.

D. Discussion

However, a critical analysis of these frameworks reveals both convergent patterns and notable limitations that warrant deeper scrutiny. Collectively, they converge on a core paradigm: framing MLE tasks as a structured search problem, where the solution space is navigated through either tree-based exploration (e.g., AutoMind's[39] knowledge-grounded tree search, ML-Master's[32] MCTS) or distributed collaboration (e.g., AutoKaggle's[30] multi-agent partitioning). This shared focus on systematic exploration reflects a recognition that MLE tasks—characterized by complex pipelines, hyperparameter interactions, and domain-specific constraints—demand more than ad-hoc reasoning.

Yet, significant divergences emerge in their handling of exploration-exploitation tradeoffs. Methods like AIDE[24], reliant on greedy search, prioritize efficiency but risk prematurely converging to local optima, especially in high-dimensional spaces (e.g., deep learning hyperparameter tuning). In contrast, approaches integrating evolutionary strategies (FM-Agent[29]) or parallel trace management (R&D-Agent[54]) enhance exploration breadth but often incur higher computational overhead, limiting scalability for large-scale datasets or real-time applications.

Another key distinction lies in knowledge integration: AutoMLGen[12] and AutoMind[39] anchor search in curated ML knowledge bases to prune irrelevant paths, while MLE-STAR[35] relies on external search engines for dynamic model retrieval. While knowledge grounding accelerates convergence, it risks bias toward established techniques, potentially stifling innovation in novel problem settings (e.g., emerging architectures like transformers for tabular data). Conversely, search-engine-dependent strategies may introduce noise from low-quality or outdated retrieval results, undermining solution robustness.

A persistent limitation across most frameworks is their narrow focus on code-centric optimization, often neglecting broader MLE lifecycle stages—such as data validation, pipeline monitoring, or deployment scalability—that are critical for real-world impact. Additionally, evaluation remains predominantly benchmark-bound (e.g., MLE-Bench[8], Kaggle competitions), with limited evidence of generalization to messy, real-world datasets characterized by missing values, concept drift, or ethical constraints. These gaps highlight the need for future work to balance search efficiency with adaptability, integrate holistic MLE workflows, and validate performance in ecologically valid settings.

Agent	LLM(s) used	Low (%)	Medium (%)	High (%)	All (%)	Running Time (hours)	Date	Source Code Available
FM Agent[29]	Gemini-2.5-Pro	62.12 ± 3.03	36.84 ± 2.63	33.33 ± 0	43.56 ± 1.78	24	2025-10-10	X
Operand ensemble	gpt-5 (low verbosity/effort)	63.64 ± 5.92	33.33 ± 4.42	20.00 ± 5.96	39.56 ± 3.26	24	2025-10-06	X
MLE-STAR-Pro	Gemini-2.5-Pro	66.67 ± 1.52	25.44 ± 0.88	31.11 ± 2.22	38.67 ± 0.77	12	2025-11-03	X
AutoMLGen[12]	deepseek-r1	62.12 ± 3.03	26.32 ± 2.63	24.44 ± 2.22	36.44 ± 1.18	12	2025-09-12	X
R&D-Agent	gpt-5	68.18 ± 2.62	21.05 ± 1.52	22.22 ± 2.22	35.11 ± 0.44	12	2025-09-26	✓
Neo multi-agent	undisclosed	48.48 ± 1.52	29.82 ± 2.32	24.44 ± 2.22	34.22 ± 0.89	36	2025-07-28	X
AIRA-dojo	o3	55.0 ± 1.5	22.0 ± 1.2	21.7 ± 1.1	31.6 ± 0.8	24	2025-05-15	✓
R&D-Agent[54]	o3 + GPT-4.1	51.52 ± 4	19.3 ± 3.16	26.67 ± 0	30.22 ± 0.89	24	2025-08-15	✓
ML-Master[32]	deepseek-r1	48.5 ± 1.5	20.2 ± 2.3	24.4 ± 2.2	29.3 ± 0.8	12	2025-06-17	✓
R&D-Agent[54]	o1-preview	48.18 ± 1.1	8.95 ± 1.05	18.67 ± 1.33	22.4 ± 0.5	24	2025-05-14	✓
AIDE[24]	o1-preview	34.3 ± 2.4	8.8 ± 1.1	10.0 ± 1.9	16.9 ± 1.1	24	2024-10-08	✓
AIDE[24]	gpt-4o-2024-08-06	19.0 ± 1.3	3.2 ± 0.5	5.6 ± 1.0	8.6 ± 0.5	24	2024-10-08	✓
AIDE[24]	claude-3-5-sonnet-20240620	19.4 ± 4.9	2.6 ± 1.5	2.3 ± 2.3	7.5 ± 1.8	24	2024-10-08	✓
OpenHands[49]	gpt-4o-2024-08-06	11.5 ± 3.4	2.2 ± 1.3	1.9 ± 1.9	5.1 ± 1.3	24	2024-10-08	✓
AIDE[24]	llama-3.1-405b-instruct	8.3 ± 2.6	1.2 ± 0.8	0.0 ± 0.0	3.1 ± 0.9	24	2024-10-08	✓
MLAB[49]	gpt-4o-2024-08-06	4.2 ± 1.5	0.0 ± 0.0	0.0 ± 0.0	1.3 ± 0.5	24	2024-10-08	✓

表 I

Comparison of LLM Agents for MLE Automation

All results are taken from the GitHub repository of MLE-bench paper[8]. Scores represent the mean and one standard error of the mean.

IV. Evaluation and Benchmarks for MLE Agents

As the field of MLE agents rapidly advances, rigorous evaluation frameworks and standardized benchmarks become indispensable for assessing their capabilities, identifying limitations, and guiding future development. MLE agents[32][12][24][35][30], which aim to automate complex machine learning engineering workflows spanning data processing, model development, deployment, and maintenance, operate in a multifaceted ecosystem that demands evaluation across diverse dimensions—from code generation proficiency and task-specific problem-solving to system integration and adaptive decision-making. Traditional evaluation metrics for standalone LLMs or narrow AI systems are insufficient here, as they fail to capture the holistic agency, tool utilization, and contextual reasoning required of MLE agents. This section reviews existing benchmarks relevant to MLE agents, categorizing them into software engineering benchmarks (foundational for coding tasks), MLE-specific benchmarks (tailored to ML engineering workflows), and general-purpose agent benchmarks (assessing broader agency capabilities). By synthesizing these evaluation efforts, we highlight gaps in current methodologies and outline the need for more comprehensive frameworks that align with the unique challenges of MLE automation.

A. Software Engineering Benchmarks

The integration of large language models (LLMs) into software development workflows has spurred the creation of benchmarks to evaluate their code generation and understanding capabilities, laying a critical foundation for assessing MLE agents—given that coding constitutes a core component of machine learning engineering.

Early benchmarks focused on fundamental code synthesis tasks, where models generate code snippets from natural language descriptions or incomplete specifications. HumanEval [9] introduced a curated set of 164 handwritten programming problems, primarily in Python, to measure functional correctness via unit tests, becoming a staple for initial LLM coding evaluations. MBPP (Mostly Basic Programming Problems) [5] expanded this with 974 problems, emphasizing accessibility to non-experts and including test cases for validation.

As LLMs advanced, more challenging benchmarks emerged. The Apps benchmark [19] raised the bar by incorporating 10,000+ problems from competitive programming platforms (e.g., Codeforces), requiring not just syntax correctness but also algorithmic reasoning and optimization. LiveCodeBench [23] addressed limitations of static benchmarks by introducing dynamic, real-world coding scenarios with continuous updates to mitigate data contamination, evaluating not only code generation but also debugging, refactoring, and collaboration with tools like linters.

While frontier models[36] have achieved near-saturating performance on many of these synthetic benchmarks [21], their success in automating end-to-end software engineering remains limited. This gap motivated benchmarks rooted in real-world development workflows, such as SWE-bench [25], which tasks models with resolving 2,294 genuine pull requests from open-source repositories (e.g., Django, scikit-learn) by fixing bugs or adding features. Such benchmarks highlight the disparity between synthetic task performance and the complexity of real-world software development—including understanding legacy code, adhering to project conventions, and reasoning about system-wide impacts—lessons directly relevant to evaluating MLE agents, which must similarly navigate real-world ML codebases and infrastructure.

B. MLE-Specific Benchmarks

While software engineering benchmarks lay the groundwork for evaluating coding proficiency, MLE agents operate in a domain-specific ecosystem that demands specialized benchmarks tailored to machine learning workflows—encompassing data preprocessing, model selection, hyperparameter tuning, evaluation, and deployment. These benchmarks must account for the unique interplay between code generation, domain

knowledge (e.g., understanding ML algorithms, statistical properties), and tool utilization (e.g., interacting with frameworks like scikit-learn, PyTorch, or cloud ML services), which distinguishes MLE automation from general software development.

MLAgentBench[22] takes 13 tasks from Kaggle and bespoke ML tasks, provides a simple baseline solution for each, and evaluates how often agents can achieve at least a 10% improvement over the baseline solution. ML-Bench[47], tests agents' abilities to generate code and execute commands to interact with popular ML repositories. DSBench[26] consists of 466 data analysis tasks and 74 data modeling tasksk, designed to evaluate data science agents with realistic tasks.

Among these, MLE-bench [8] has emerged as a widely adopted standard, offering 75 carefully curated tasks derived from offline Kaggle competitions, each designed by a team of ML engineers to reflect realistic challenges in classification, regression, and unsupervised learning. What distinguishes MLE-bench is its focus on end-to-end ML engineering: agents must independently handle data exploration, feature engineering, model selection, hyperparameter tuning, and result validation, with performance measured against competition leaderboard metrics. Its large task diversity, emphasis on holistic workflow automation, and alignment with real-world competition dynamics have made it a benchmark of choice for assessing MLE agents' practical capabilities.

Collectively, these MLE-specific benchmarks address critical gaps in general software engineering evaluations, but they vary in scope—from incremental improvement (MLAgentBench) to library-specific tool use (ML-Bench) and full lifecycle automation (DSBench, MLE-bench). MLE-bench, in particular, has gained recognition for its balance of realism, task diversity, and focus on end-to-end proficiency, making it a reference point for comparing state-of-the-art MLE agents. However, even these benchmarks primarily focus on technical ML tasks, leaving gaps in evaluating broader agency capabilities such as long-term planning, dynamic resource allocation, and adaptation to evolving requirements—areas where general-purpose agent benchmarks provide complementary insights.

C. General-Purpose Agent Benchmarks

SWE-bench, MLAgentBench, and MLE-bench are multi-step benchmarks evaluating AI agents in the software domain. Here, components such as LMs, retrieval, and external tools are "scaffolded" together via code, unlocking new levels of autonomy unattainable via a single inference call[57].

AgentBench[31] provides environments for agents to complete multi-turn challenges, such as editing permissions on a Linux OS. OSworld[52] is the first-of-its-kind scalable, real computer environment for multimodal agents, supporting task setup, execution-based evaluation, and interactive learning across various operating systems such as Ubuntu, Windows, and macOS. GAIA[34] focuses on agent interactions with the real world, providing 466 questions that are conceptually simple for humans but challenging for current AI systems. [15] propose AgentQuest, a modular agent evaluation framework designed for extensibility, and [27] provide an analysis of agent evaluation efforts so far. PaperBench[45] is a benchmark consisting of 20 ML research papers and author-approved rubrics, and an automated grading workflow using LLM-based judges。 It is designed to evaluate the ability of AI agents to replicate state-of-the-art AI research.

Compared to MLE-specific benchmarks, these general-purpose agent benchmarks emphasize broader cognitive and operational capabilities beyond technical ML workflows. For instance, multimodal understanding (as in OSworld) enables agents to process visual interfaces, GUI elements, and non-textual system feedback—skills less critical in scripted ML pipelines but essential for real-world tool operation. Others, like PaperBench, assess scientific reasoning through complete research replication, encompassing literature review, methodology adaptation, and result interpretation—steps that extend beyond the technical execution focus of MLE benchmarks. Meanwhile, frameworks like AgentBench and GAIA prioritize general problem-

solving across diverse domains, testing an agent's ability to decompose novel tasks, adapt to unfamiliar environments, and recover from errors—competencies that underpin robustness but are not specific to ML engineering contexts. Collectively, these benchmarks complement MLE-specific evaluations by measuring the "general intelligence" foundations required for flexible, real-world agency.

D. Discussion

The distinct focus of MLE-specific benchmarks remains indispensable for advancing ML engineering automation, as they address the unique confluence of domain knowledge, tool proficiency, and workflow orchestration that defines the MLE agent's role. Unlike general-purpose benchmarks, which prioritize breadth of capability across diverse tasks, MLE-specific frameworks like MLE-bench[8] and DSBench[26] are uniquely positioned to measure mastery of specialized ML workflows—from statistical feature engineering to model deployment best practices—that demand deep expertise in ML systems. Similarly, while software engineering benchmarks excel at evaluating coding proficiency, they lack the domain-specific metrics (e.g., model accuracy, training efficiency, generalization performance) that are central to assessing ML engineering success.

That said, the boundaries between these categories are not rigid. General-purpose agent benchmarks offer critical insights into capabilities that enhance MLE agents' practical utility, such as multimodal interaction with ML dashboards (OSworld[52]), long-term project planning (AgentQuest[15]), and research-driven innovation (PaperBench[45]). Future evaluation frameworks may benefit from integrating these complementary strengths: combining MLE-specific technical rigor with the broader agency metrics of general benchmarks. Such hybrid frameworks could, for example, evaluate an agent's ability to optimize a model's performance while adapting to changing data requirements, documenting decisions for stakeholders, and troubleshooting deployment issues across cloud environments—mirroring the end-to-end responsibilities of human ML engineers. Ultimately, advancing MLE agent evaluation requires both specialized benchmarks that drill into domain-specific competence and integrative frameworks that capture the holistic, real-world complexity of ML engineering practice.

V. Challenges and Outlook

The rapid evolution of LLM-based agents for machine learning engineering (MLE) automation, coupled with the development of specialized benchmarks, has demonstrated significant progress in automating complex ML workflows. However, as these agents transition from controlled benchmark environments to real-world MLE scenarios, a host of challenges emerge—ranging from limitations in exploration strategies and knowledge integration to gaps in handling the full MLE lifecycle. Addressing these challenges will be critical for unlocking the full potential of MLE agents, enabling them to match or exceed human ML engineers in both technical proficiency and practical impact. This section examines key challenges in current MLE agent systems and outlines promising directions for future research, with a focus on advancing exploration mechanisms, framework architectures, alignment with real-world MLE demands and much more ambitious target(AI4AI).

A. MLE Agent System

1) ***Current Challenges****:* The rapid advancement of MLE agents has been accompanied by persistent challenges across multiple dimensions, from structural design to adaptive capabilities:

Framework Structure Limitations: Existing agent architectures struggle to balance flexibility and efficiency in navigating complex MLE solution spaces. Early tree search approaches (e.g., AIDE[24]) suffer from rigid exploration paths that exacerbate local optimum trapping. While MCTS-based frameworks (ML-Master[32]) improve exploration-exploitation balance, they retain tree-structured constraints that fail to capture cyclic dependencies in ML pipelines (e.g., iterative feature engineering and model validation loops). More

表 II
COMPARISON OF EXPLORATION STRATEGIES AND PARALLEL EXECUTION CAPABILITIES OF DIFFERENT METHODS

Method	Exploration Strategy	Parallel Execution
MLAB[22]	Chain	×
OpenHands[49]	Chain	×
SELA[10]	Tree & Predefined	×
AIDE[24]	Tree & Greedy	×
Mlzero[13]	Iterative	×
MLE-STAR[35]	Search Engine	×
Agent Laboratory[41]	Tree & Greedy	✓
R&D-Agent[54]	Multi-chain & Fusion	✓
AutoKaggle[30]	Chain & Iterative	✓
ML-Master[32]	Tree & Balanced	✓
AutoMLGen[12]	Graph	✓
FM Agent[29]	Evolutionary Algorithm	✓

recent graph-based methods (AutoMLGen[12]) address this partially but lack dynamic adjustment of graph complexity based on task difficulty, leading to either under-exploration or computational overload.

Tool Utilization and Model Context Protocol (MCP) Gaps: Despite progress in tool integration, current agents exhibit limited proficiency in dynamically managing context across diverse tools and model interactions. AutoKaggle's[30] multi-agent collaboration relies on rigid context-sharing protocols that hinder seamless information flow between specialized sub-agents (e.g., data preprocessing). MLE-STAR[35] demonstrates effective search engine integration but lacks standardized MCP for aligning retrieved external knowledge with internal model states, leading to context mismatches when applying search results to specific pipeline stages.

Context Engineering Deficiencies: Context management remains a critical bottleneck, with agents struggling to maintain relevant information across extended problem-solving trajectories. While ML-Master[32] introduces a selectively scoped memory mechanism, it lacks fine-grained awareness of intermediate MLE processes—such as incremental performance changes during hyperparameter tuning, data drift detection signals, or trade-offs between model complexity and inference latency. This deficit leads to incomplete context representations, where agents may retain high-level metrics (e.g., final accuracy) but discard granular insights (e.g., which feature transformations most influenced performance gains) that are critical for iterative refinement.

Limited Self-Evolving Capabilities: While frameworks like ML-Master[32] and AutoMLGen[12] incorporate basic iterative improvement—using past results to guide subsequent search steps—their self-evolving capabilities remain constrained. Both frameworks struggle to determine *what to evolve*: they often refine structural components(e.g., pre-training model selection) while overlooking how to make good use of these structures(e.g., hyperparameter finetuning), and they lack meta-level reasoning to prioritize evolution directions based on task complexity or resource constraints.

2) ***Outlook****:* The evolution of MLE agent systems is poised to address these challenges through targeted innovations across framework design, tool utilization, context management, and adaptive learning:

Framework Structure: Adaptive Graphs with Dynamic Complexity: Building on the trajectory from tree search (AIDE[24]) to MCTS (ML-Master[32]) and MCGS (AutoMLGen[12]), future frameworks may prioritize flexible graph structures that adapt to task complexity. This direction aims to resolve tree-based rigidity and static graph inefficiency by dynamically adjusting representation granularity

and capturing cyclic dependencies inherent to ML pipelines, balancing exploration breadth with computational feasibility.

Tool Use and Model Context Protocol (MCP): Semantic Interoperability: To address rigid context sharing and alignment gaps, future advancements may center on standardized, semantic MCP frameworks. This direction extends AutoKaggle' s[30] collaboration and MLE-STAR' s[35] search integration by enabling seamless cross-tool and multi-agent information flow, resolving context mismatches between external knowledge and internal model states, and simplifying dynamic tool integration.

Context Engineering: Process-Aware Memory Systems: Moving beyond ML-Master' s[32] selective memory, future context management may focus on capturing fine-grained intermediate MLE processes. This direction emphasizes retaining granular insights from iterative workflows and organizing context hierarchically to support informed, cross-stage decision-making, addressing gaps in incomplete context representation.

Self-Evolving Agents: Targeted Meta-Reasoning: To overcome unfocused and slow evolution in frameworks like ML-Master[32] and AutoMLGen[12], future agents may integrate meta-reasoning-driven adaptation. This direction prioritizes identifying high-impact evolution targets, aligning refinement with task complexity and resource constraints, and moving beyond trivial component tweaks to meaningful, context-aware self-improvement. By advancing these core directions, MLE agents may evolve from task-specific automation tools to versatile, adaptive systems capable of navigating the full complexity of real-world machine learning engineering.

B. From MLE Automation to AI4AI

Artificial Intelligence (AI) has profoundly reshaped human civilization, driving transformative advancements across diverse areas[51][17]. As AI advances toward surpassing human-level intelligence, it is crucial to advocate for the emergence of AI-for-AI (AI4AI), which leverages AI techniques to automate and optimize the design, training, and deployment of AI systems themselves[60][46]. AI-for-AI can be understood as a progressive paradigm unfolding in three stages: human-led human-AI collaboration, AI-led human-AI collaboration, and fully autonomous AI systems. We are likely in the midst of transitioning from the first to the second stage. The ultimate form of AI-for-AI envisions fully autonomous systems capable of end-to-end AI research and development—from hypothesis generation and experimental design to algorithmic discovery and validation. One possible vision for the development of AI4AI can be drawn from the progression seen in systems like AlphaGo[43] and AlphaZero[44], where the journey began with human-guided training to improve the machine' s Go-playing skills, eventually leading to machines surpassing human-level performance and further enhancing their skills through self-play.

*1) **Current AI4AI Challenge**:* Although recent breakthroughs in Large Language Models (LLMs)[11][38][1] and autonomous agents[49][58][59][41] have provided evidence supporting the feasibility of AI4AI, most studies still encounter significant challenges. Previous works[33][10][22][56][42] like AI Scientist[33], SELA[10], and Dolphin[56] primarily emphasize exploration strategies without sufficiently leveraging the analytical reasoning capabilities of advanced reasoning models, thus missing valuable insights and limiting their adaptability in complex scenarios.

Specifically, AI Scientist[33] is capable of generating novel research ideas, writing code, executing experiments, visualizing results, and describing its findings by composing complete scientific papers, followed by conducting simulated review processes for evaluation. SELA[10] incorporates tree-based search mechanisms to enhance the decision-making capabilities of large language models. Dolphin[56] introduces retrieval-augmented generation and an iterative refinement strategy to improve the quality of idea proposal processes. Paper2Code[42] demonstrates exceptional performance in translating research papers into

executable code.

2) ***Outlook: From MLE Automation to AI4AI****:* The evolution of artificial intelligence has traversed distinct stages, with the role of MLE marking a pivotal phase in human-driven AI development. MLE have been the architects of AI advancement, translating theoretical insights into practical systems, optimizing model performance, and bridging the gap between research and deployment. As AI systems grow increasingly complex(encompassing trillion-parameter models[11], multi-modal architectures[6], and dynamic real-world deployment scenarios), the demands on human MLE have surged beyond sustainable limits, necessitating a paradigm shift toward greater automation.

This shift naturally converges with the broader vision of AI4AI: a future where AI systems autonomously advance AI research and development. Within this trajectory, MLE Automation (or 'AI for MLE') emerges as a critical and pragmatic stepping stone. By automating the repetitive, high-complexity tasks central to MLE workflows(such as hyperparameter tuning, architecture optimization, deployment pipeline management, and performance debugging), MLE Automation not only alleviates human bottlenecks but also establishes a foundational framework for AI4AI. It codifies the expertise, heuristics, and iterative processes inherent to MLE work into structured, learnable tasks that AI systems can progressively master and refine.

In essence, MLE Automation represents the first frontier of AI4AI, where AI begins to assume the role of an "assistant MLE," then a "collaborative MLE," and ultimately an "autonomous MLE." This incremental progression—rooted in solving tangible, domain-specific challenges of AI development—provides a concrete pathway to realizing the broader AI4AI vision, ensuring that the transition from human-led to AI-augmented to fully autonomous AI development is both feasible and robust.

VI. Conclusions

The intersection of Large Language Models (LLMs) and Machine Learning Engineering (MLE) Automation has emerged as a defining frontier in the broader pursuit of AI-for-AI (AI4AI), marking a pivotal shift from fragmented workflow optimization to holistic, autonomous AI-driven development. This review provides the first comprehensive synthesis of this rapidly evolving field, systematically organizing the foundational theories, state-of-the-art systems, evaluation benchmarks, and future trajectories of LLM agents for MLE Automation.

At its core, this work establishes a unified framework for understanding MLE-oriented agent systems—formalizing the MLE environment as a goal-conditioned Partially Observable Markov Decision Process (POMDP) and defining agent architectures through modular components of LLMs, context management, and tool integration. Through a three-tier taxonomy, we trace the evolutionary path of MLE automation: from pre-LLM AutoML systems focused on isolated pipeline stages, to general-purpose coding agents with broad software engineering capabilities but limited ML domain expertise, to specialized MLE agents that integrate structured exploration (e.g., tree/graph search, multi-agent collaboration) and ML lifecycle awareness to tackle end-to-end workflows. This taxonomy highlights a clear convergent trend: the most effective MLE agents frame automation as a structured search problem, balanced by domain knowledge integration and iterative refinement.

Our systematic analysis of evaluation benchmarks reveals a complementary ecosystem: software engineering benchmarks lay the groundwork for coding proficiency, MLE-specific benchmarks (e.g., MLE-bench) address domain-specific workflow mastery, and general-purpose agent benchmarks enhance assessments of broader cognitive and operational capabilities. Together, these benchmarks expose critical gaps in current evaluations—particularly in real-world adaptability, lifecycle-wide optimization, and alignment with human MLE responsibilities—which future frameworks must address to bridge the gap between benchmark performance and practical deployment.

We further identify four core challenges con-

straining current MLE agent systems: rigid framework structures that struggle with cyclic MLE dependencies, inadequate tool integration via standardized Model Context Protocols (MCP), deficient context engineering that overlooks granular intermediate processes, and limited self-evolving capabilities lacking targeted meta-reasoning. Corresponding future directions—adaptive graph-based architectures, semantic interoperability for tool use, process-aware memory systems, and meta-reasoning-driven self-improvement—offer actionable pathways to enhance agent flexibility, efficiency, and robustness.

Fundamentally, this review articulates MLE Automation as a pragmatic and indispensable stepping stone toward AI4AI. By automating the high-complexity, repetitive tasks central to MLE workflows, LLM agents not only alleviate human bottlenecks but also codify ML engineering expertise into structured, learnable processes that pave the way for fully autonomous AI development.

The four primary contributions of this work—unifying fragmented research, formalizing foundational definitions, systematizing benchmark comparisons, and mapping the MLE-to-AI4AI pathway—provide a critical knowledge base for researchers and practitioners alike, establishing a consistent terminology and structured framework to guide future innovations.

As the field advances, the transition from MLE Automation to AI4AI may hinge on embracing incremental progress: refining agent systems to master real-world MLE complexities, integrating cross-domain capabilities from general-purpose agent research, and validating performance in ecologically valid settings beyond synthetic benchmarks. In doing so, LLM agents for MLE Automation may not only transform how AI systems are built and maintained but also unlock the broader vision of self-evolving AI—where AI itself becomes the primary driver of its own advancement. This review serves as both a snapshot of the current state and a roadmap for the journey ahead, emphasizing that the success of AI4AI may be rooted in solving the tangible, domain-specific challenges of MLE Automation.

参考文献

[1] Anthropic AI. System card: Claude opus 4 claude sonnet 4, 2024. Accessed: [2025-11-11].

[2] Aider-AI. Aider: Ai pair programming in your terminal. https://github.com/Aider-AI/aider, 2025. Accessed: [2025-11-10].

[3] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 291–300, 2019.

[4] Huan ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, Hongru Wang, Han Xiao, Yuhang Zhou, Shaokun Zhang, Jiayi Zhang, Jinyu Xiang, Yixiong Fang, Qiwen Zhao, Dongrui Liu, Qihan Ren, Cheng Qian, Zhenhailong Wang, Minda Hu, Huazheng Wang, Qingyun Wu, Heng Ji, and Mengdi Wang. A survey of self-evolving agents: On path to artificial super intelligence, 2025.

[5] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021.

[6] Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-vl technical report, 2025.

[7] Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. Mle-bench: Evaluating machine learning agents on machine learning engineering. *arXiv preprint arXiv:2410.07095*, 2024.

[8] Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Mądry. Mle-bench: Evaluating machine learning agents on machine learning engineering, 2025.

[9] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021.

[10] Yizhou Chi, Yizhang Lin, Sirui Hong, Duyi Pan, Yaying Fei, Guanghao Mei, Bangbang Liu, Tianqi Pang, Jacky Kwok, Ceyao Zhang, Bang Liu, and Chenglin Wu. Sela: Tree-search enhanced llm agents for automated machine learning, 2024.

[11] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, and Shirong Ma etc. Qihao Zhu and. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025.

[12] Shangheng Du, Xiangchao Yan, Dengyang Jiang, Jiakang Yuan, Yusong Hu, Xin Li, Liang He, Bo Zhang, and Lei Bai. Automlgen: Navigating fine-grained optimization for coding agents, 2025.

[13] Haoyang Fang, Boran Han, Nick Erickson, Xiyuan Zhang, Su Zhou, Anirudh Dagar, Jiani Zhang, Ali Caner Turkmen, Cuixiong Hu, Huzefa Rangwala, Ying Nian Wu, Bernie Wang, and George Karypis. Mlzero: A multi-agent system for end-to-end machine learning automation, 2025.

[14] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. Auto-sklearn 2.0: Hands-free automl via meta-learning, 2022.

[15] Luca Gioacchini, Giuseppe Siracusano, Davide Sanvito, Kiril Gashteovski, David Friede, Roberto Bifulco, and Carolin Lawrence. Agentquest: A modular benchmark framework to measure progress and improve llm agents, 2024.

[16] Siyuan Guo, Cheng Deng, Ying Wen, Hechang Chen, Yi Chang, and Jun Wang. Ds-agent: Automated data science by empowering large language models with case-based reasoning, 2024.

[17] Kunal Handa, Alex Tamkin, Miles McCain, Saffron Huang, Esin Durmus, Sarah Heck, Jared Mueller, Jerry Hong, Stuart Ritchie, Tim Belonax, Kevin K. Troy, Dario Amodei, Jared Kaplan, Jack Clark, and Deep Ganguli. Which economic tasks are performed with ai? evidence from millions of claude conversations, 2025.

[18] Xin He, Kaiyong Zhao, and Xiaowen Chu. Automl: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212:106622, January 2021.

[19] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring coding challenge competence with apps, 2021.

[20] Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for a multi-agent collaborative framework, 2024.

[21] Dong Huang, Jie M. Zhang, Michael Luck, Qingwen Bu, Yuhao Qing, and Heming Cui. Agentcoder: Multi-agent-based code generation with iterative testing and optimisation, 2024.

[22] Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. Mlagentbench: Evaluating language agents on machine learning experimentation. *arXiv preprint arXiv:2310.03302*, 2023.

[23] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code, 2024.

[24] Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. Aide: Ai-driven exploration in the space of code, 2025.

[25] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues?, 2024.

[26] Liqiang Jing, Zhehui Huang, Xiaoyang Wang, Wenlin Yao, Wenhao Yu, Kaixin Ma, Hongming Zhang, Xinya Du, and Dong Yu. Dsbench: How far are data science agents from becoming data science experts?, 2025.

[27] Sayash Kapoor, Benedikt Stroebl, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. Ai agents that matter, 2024.

[28] Janet Kolodner. An introduction to case-based reasoning. *Artificial Intelligence Review*, 6:3–34, 03 1992.

[29] 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, and Dou Shen. The fm agent, 2025.

[30] Ziming Li, Qianbo Zang, David Ma, Jiawei Guo, Tuney Zheng, Minghao Liu, Xinyao Niu, Yue Wang, Jian Yang, Jiaheng Liu, Wanjun Zhong, Wangchunshu Zhou, Wenhao Huang, and Ge Zhang. Autokaggle: A multi-agent framework for autonomous data science competitions, 2024.

[31] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating llms as agents, 2025.

[32] Zexi Liu, Yuzhu Cai, Xinyu Zhu, Yujie Zheng, Runkun Chen, Ying Wen, Yanfeng Wang, Weinan E, and Siheng Chen. Ml-master: Towards ai-for-ai via integration of exploration and reasoning, 2025.

[33] Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist: Towards fully automated open-ended scientific discovery, 2024.

[34] Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants, 2023.

[35] 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, 2025.

[36] OpenAI. Gpt-4 technical report, 2024.

[37] OpenAI. Gpt-4o system card, 2024.

[38] OpenAI. Introducing openai o1. https://openai.com/o1/, 2024. Accessed: [2025-11-11].

[39] Yixin Ou, Yujie Luo, Jingsheng Zheng, Lanning Wei, Zhuoyun Yu, Shuofei Qiao, Jintian Zhang, Da Zheng, Yuren Mao, Yunjun Gao, Huajun Chen, and Ningyu Zhang. Automind: Adaptive knowledgeable agent for automated data science, 2025.

[40] Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. Chatdev: Communicative agents for software development, 2024.

[41] Samuel Schmidgall, Yusheng Su, Ze Wang, Ximeng Sun, Jialian Wu, Xiaodong Yu, Jiang Liu, Michael Moor, Zicheng Liu, and Emad Barsoum. Agent laboratory: Using llm agents as research assistants, 2025.

[42] Minju Seo, Jinheon Baek, Seongyun Lee, and Sung Ju Hwang. Paper2code: Automating code generation from scientific papers in machine learning, 2025.

[43] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, L. Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.

[44] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[45] Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, et al. Paperbench: Evaluating ai's ability to replicate ai research. *arXiv preprint arXiv:2504.01848*, 2025.

[46] Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. Cognitive architectures for language agents, 2024.

[47] Xiangru Tang, Yuliang Liu, Zefan Cai, Yanjun Shao, Junjie Lu, Yichi Zhang, Zexuan Deng, Helan Hu, Kaikai An, Ruijun Huang, Shuzheng Si, Sheng Chen, Haozhe Zhao, Liang Chen, Yan Wang, Tianyu Liu, Zhiwei Jiang, Baobao Chang, Yin Fang, Yujia Qin, Wangchunshu Zhou, Yilun Zhao, Arman Cohan, and Mark Gerstein. Ml-bench: Evaluating large language models and agents for machine learning tasks on repository-level code, 2024.

[48] Huacan Wang, Ziyi Ni, Shuo Zhang, Shuo Lu, Sen Hu, Ziyang He, Chen Hu, Jiaye Lin, Yifu Guo, Ronghao Chen, Xin Li, Daxin Jiang, Yuntao Du, and Pin Lyu. Repomaster: Autonomous exploration and understanding of github repositories for complex task solving, 2025.

[49] 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, 2025.

[50] Ian Watson and Farhi Marir. Case-based reasoning: A review. *The Knowledge Engineering Review*, 9(4):327–354, 1994.

[51] Justus Wolff, Josch Pauling, Andreas Keck, and Jan Baumbach. The economic impact of artificial intelligence in health care: Systematic review. *J Med Internet Res*, 22(2):e16866, Feb 2020.

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

[53] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering, 2024.

[54] Xu Yang, Xiao Yang, Shikai Fang, Yifei Zhang, Jian Wang, Bowen Xian, Qizheng Li, Jingyuan Li, Minrui Xu, Yuante Li, Haoran Pan, Yuge Zhang, Weiqing Liu, Yelong Shen, Weizhu Chen, and Jiang Bian. Rd-agent: An llm-agent framework towards autonomous data science, 2025.

[55] Shunyu Yao. The second half. https://ysymyth.github.io/The-Second-Half/, 2025. Accessed: [2025-11-11].

[56] Jiakang Yuan, Xiangchao Yan, Shiyang Feng, Bo Zhang, Tao Chen, Botian Shi, Wanli Ouyang, Yu Qiao, Lei Bai, and Bowen Zhou. Dolphin: Moving towards closed-loop auto-research through thinking, practice, and feedback, 2025.

[57] Matei Zaharia, Omar Khattab, Lingjiao Chen, Jared Quincy Davis, Heather Miller, Chris Potts, James Zou, Michael Carbin, Jonathan Frankle, Naveen Rao, and Ali Ghodsi. The shift from models to compound ai systems. https://bair.berkeley.edu/blog/2024/02/18/compound-ai-systems/, 2024.

[58] Shujian Zhang, Chengyue Gong, Lemeng Wu, Xingchao Liu, and Mingyuan Zhou. Automl-gpt: Automatic machine learning with gpt, 2023.

[59] Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. Deepresearcher: Scaling deep research via reinforcement learning in real-world environments, 2025.

[60] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning, 2017.