RESEARCH-ARTICLE

Large Language Models for Constructing and Optimizing Machine Learning Workflows: A Survey

YANG GU, Shanghai Jiao Tong University, Shanghai, China

HENGYU YOU, Shanghai Jiao Tong University, Shanghai, China

JIAN CAO, Shanghai Jiao Tong University, Shanghai, China

MURAN YU, Stanford University, Stanford, CA, United States

HAORAN FAN, Shanghai Jiao Tong University, Shanghai, China

SHIYOU QIAN, Shanghai Jiao Tong University, Shanghai, China

Accepted: 20 October 2025
Revised: 23 August 2025
Received: 11 February 2025

Large Language Models for Constructing and Optimizing Machine Learning Workflows: A Survey

YANG GU, Shanghai Jiao Tong University, Peking University, China
HENGYU YOU and JIAN CAO*, Shanghai Jiao Tong University, China
MURAN YU, Stanford University, USA
HAORAN FAN and SHIYOU QIAN, Shanghai Jiao Tong University, China

Machine Learning (ML) workflows—spanning data preprocessing and feature engineering, model selection and hyperparameter optimization, and workflow evaluation—are increasingly embedded in complex software systems. Building these workflows manually demands substantial ML expertise, domain knowledge, and engineering effort. Automated ML (AutoML) frameworks address parts of this challenge but often suffer from constrained search spaces, limited adaptability, and low interpretability. Recent advances in Large Language Models (LLMs) have opened new opportunities to automate and enhance ML workflows by leveraging their capabilities in language understanding, reasoning, interaction, and code generation, posing new practical and theoretical challenges for software engineering (SE). This survey provides the first SE-oriented, stage-wise review of LLM-based ML workflow automation. We introduce a taxonomy covering all three workflow stages, systematically compare and analyze state-of-the-art methods, and synthesize both stage-specific and cross-stage trends. Our analysis yields SE-oriented implications, including the need for robust verification, quality management, context-aware deployment, and risk mitigation, alongside ensuring key quality attributes such as usability, modularity, traceability, and performance. The findings also call for adapting development models, rethinking lifecycle boundaries, and formalizing uncertainty handling to address the probabilistic and collaborative nature of LLM-assisted workflow generation. We further identify major open challenges and outline future research directions to guide the reliable and effective adoption of LLMs in ML workflow development. Our artifacts are publicly available at https://github.com/t-harden/LLM4AutoML.

CCS Concepts: • **Computing methodologies** → **Machine learning**; • **Software and its engineering** → **Automatic programming**.

Additional Key Words and Phrases: Machine Learning Workflows, Large Language Models, Software Engineering, AutoML, Survey

1 INTRODUCTION

In the era of big data, machine learning (ML) has become pervasive across a wide range of domains and industries [126, 178]. One increasingly prominent approach to solving ML problems is code generation, which enables machines to automatically produce programs that reflect human intent, as expressed through ML task specifications [87]. This paradigm has long been a core topic in software engineering (SE), encompassing research areas such as program synthesis, code transformation, and automated software construction [17, 61, 70]. A

*Corresponding author

Authors' Contact Information: Yang Gu, gbtharden@gmail.com, Shanghai Jiao Tong University, Peking University, China; Hengyu You, virusyou@sjtu.edu.cn; Jian Cao, cao-jian@sjtu.edu.cn, Shanghai Jiao Tong University, China; Muran Yu, yumuran@stanford.edu, Stanford University, USA; Haoran Fan, fhr2022@sjtu.edu.cn; Shiyou Qian, qshiyou@sjtu.edu.cn, Shanghai Jiao Tong University, China.

ACM 1557-7392/2025/10-ART
https://doi.org/10.1145/3773084

functional ML application typically involves not only model training but also a full stack of components—such as data preprocessing, feature engineering, and postprocessing—that together comprise a complete **ML workflow** (Fig. 1) [179, 180, 183]. As these workflows grow in scale and complexity, software engineers increasingly adopt code generation and computational workflow techniques to automate the development of ML programs that integrate both data and model engineering [20, 49].

Traditional code generation techniques, typically based on supervised learning, have evolved to incorporate code-specific features such as abstract syntax trees (ASTs) [139, 156] and API call patterns [52]. More advanced approaches leverage static analysis [122] or formal grammars and pushdown automata to enforce syntactic correctness [32, 87]. However, despite technical advances, these methods often struggle to generate semantically valid and domain-adapted ML code, thereby limiting their effectiveness in reducing developer workload [149]. In addition, many ML tasks are domain-specific, and training or fine-tuning models for each target domain remains time-consuming and resource-intensive [111]. To address this, low-code platforms such as KNIME [131] and RapidMiner [66] have emerged, enabling workflow construction via GUI-based configuration of reusable components [86]. Yet, these platforms often expose a limited set of configurable elements, reducing flexibility and generalizability. Furthermore, building performant ML workflows for domain-specific tasks still requires considerable domain knowledge—such as identifying relevant features or tuning model hyperparameters.

At the intersection of SE and ML, a prominent class of task-specific code generation and optimization approaches is known as automated machine learning (AutoML). AutoML aims to streamline ML workflows by automating key steps such as model selection, hyperparameter tuning, and pipeline composition [15, 56, 80], often culminating in the synthesis of executable ML pipelines or scripts. Despite notable progress [6, 56, 126], traditional AutoML frameworks face persistent challenges. The underlying optimization process is typically a black-box, compute-intensive search over large configuration spaces [129]. These systems also struggle to incorporate reusable human knowledge, prior design artifacts, or domain-specific constraints—elements central to SE practices like software reuse and maintainability. Even with meta-learning and Bayesian optimization [42, 142], AutoML often lacks human-in-the-loop capabilities, limiting adaptability in real-world settings. Moreover, its outputs—while potentially effective—are often opaque and hard to interpret, which hinders their adoption in domains that require transparency, traceability, and trust [146, 201].

These limitations point to the need for approaches that combine automation with flexible, context-aware reasoning, while allowing human expertise to guide the process. This is where large language models (LLMs) have begun to reshape the landscape of AI-driven workflow automation. Models such as OpenAI's o1 [130], DeepSeek's R1 [29], and Meta AI's LLaMA-3 [164] exhibit powerful capabilities in both natural language understanding and code generation [152], enabled by large-scale pretraining on diverse corpora spanning text and source code [39, 51, 70]. Advances in multimodal LLMs [74, 113, 157] further expand their applicability to complex, real-world ML tasks that require integrating textual, visual, and structured inputs. From an SE perspective, LLMs offer new opportunities for program synthesis, automated pipeline construction, and human–AI co-creation, narrowing the gap between high-level specification and executable implementation.

In this survey, we define an ML workflow (Fig. 1) as a sequence of interconnected stages that transform a task specification into a functional ML solution [28]. We focus on three core stages: data and feature engineering, model selection and hyperparameter optimization, and workflow evaluation. From an SE standpoint, these stages involve quality attributes—such as usability, modularity, traceability, and performance—that influence maintainability, reliability, and human–AI interaction.

While prior surveys have examined the interplay between LLMs and AutoML [163] or Data Science [154], they neither adopt an SE perspective nor systematically address the full-process construction of ML workflows. This gap is critical: ML workflows are increasingly embedded in complex, evolving software systems, yet the role of LLMs in automated code generation, program synthesis, maintainability, and workflow lifecycle management remains underexplored. Our survey bridges *AI for SE*—where LLMs are used for ML code generation—and *SE for*

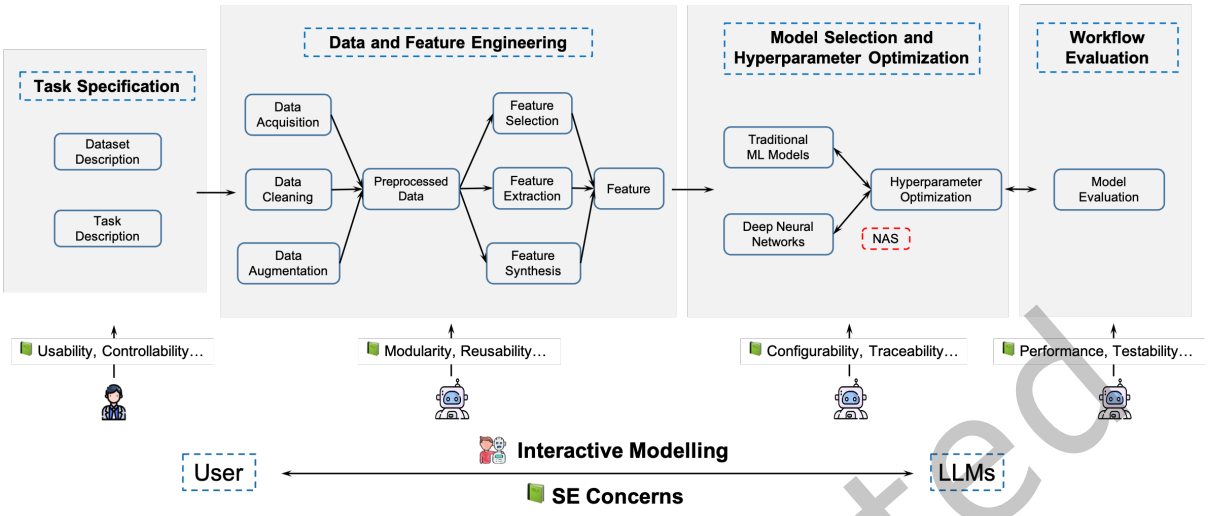


Fig. 1. Structure of the ML workflow addressed in this survey. The process unfolds across three stages, each annotated with representative SE concerns that highlight quality attributes guiding both workflow automation and evaluation.

AI—which tackles core ML development challenges—by mapping how LLMs can support **every stage** of the ML workflow. Framing the problem around established SE concerns, we identify unique opportunities and constraints for integrating LLM-based workflow automation into SE practices, setting our work apart from existing reviews.

This survey examines how LLMs can be applied to construct and optimize ML workflows, with particular attention to their **implications for SE practice and theory**. From a practical standpoint, our analysis points to the need for robust verification pipelines, systematic quality management, context-aware deployment strategies, and risk mitigation for challenges such as hallucination and data leakage. From a theoretical standpoint, the findings suggest adapting development models, rethinking lifecycle boundaries, and formalizing uncertainty handling to accommodate the probabilistic and collaborative nature of LLM-assisted development.

To support this analysis, we introduce a taxonomy (Fig. 2) that organizes and synthesizes state-of-the-art methods across workflow stages, identifies strengths, weaknesses, and research gaps, and informs the following key contributions:

- To the best of our knowledge, this is the first survey to examine LLM-based ML workflow automation across all three stages, aligning them with SE concerns.
- We propose a hierarchical comparative framework for LLM-based ML workflow methods, encompassing both component-level categorization and workflow stage-level analysis.
- We synthesize key insights from each stage and cross-stage trends, deriving broader SE implications that cover both practical adaptations and theoretical extensions.
- We summarize major SE-oriented open challenges and outline future research directions for LLM-based ML workflows, grounded in our survey's findings.

The remainder of this paper is structured as follows. Section 2 reviews foundational concepts and recent advances in ML workflows and LLMs. Section 3 describes the survey methodology. Sections 4–6 examine the application of LLMs across key workflow stages: data preprocessing and feature engineering (Section 4), model selection and hyperparameter optimization (Section 5), and workflow evaluation (Section 6). Section 7 synthesizes

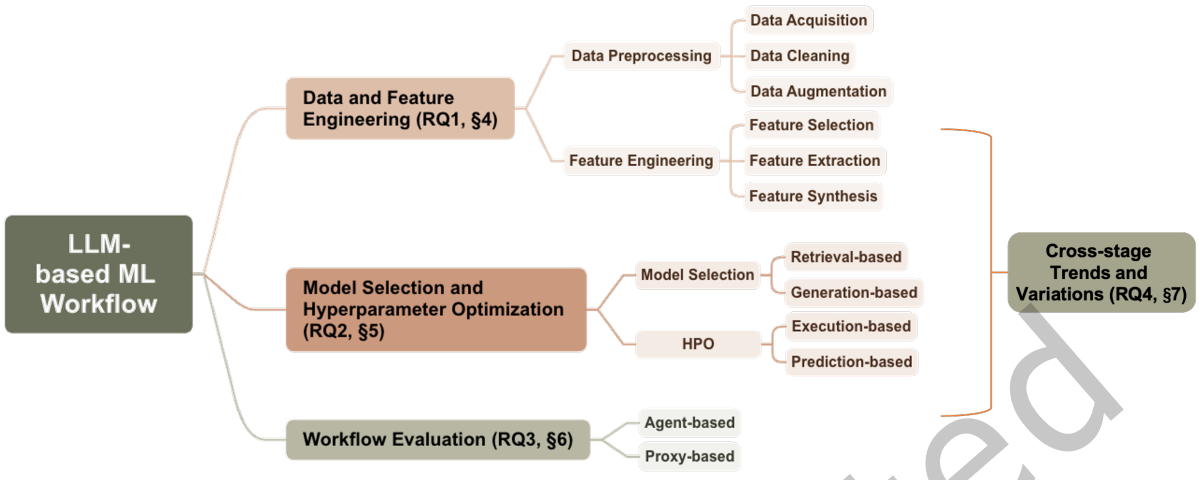


Fig. 2. The overall structure and taxonomy of this survey, organized around three core stages of the ML workflow (RQ1–RQ3), each with representative subcategories, along with a cross-stage analysis (RQ4) that synthesizes trends and variations observed across the workflow lifecycle.

cross-stage findings and discusses their broader implications for SE. Section 8 presents open challenges and future research directions, and Section 9 concludes the survey.

2 PRELIMINARIES

2.1 ML Task and Workflow

ML, a core subfield of AI, focuses on developing algorithms and statistical models that enable computers to perform specific tasks without explicit programming [4]. These tasks span classification, regression, clustering, and reinforcement learning, each aiming to discover patterns and make predictions from data [55, 78, 99, 211]. The overarching goal is to build models that generalize well from training data to unseen scenarios, thereby providing reliable and effective solutions [24].

An ML workflow is a structured sequence of interconnected components [128] that transforms a task specification into a functional ML solution in a systematic and repeatable manner. Such workflows resemble modular software systems: each stage encapsulates specific functionality, has well-defined interfaces, and should satisfy quality attributes such as usability, modularity, traceability, and performance. These attributes are critical to ensuring that workflows remain adaptable, maintainable, and auditable in real-world deployments.

The input to an ML workflow is typically a task specification, which includes a dataset definition and a task description. The dataset—comprising input features and, in supervised tasks, output labels—forms the foundation for model training [142]. For example, the HousingPrice dataset [59] may be stored in CSV format with 14 columns, where the first 13 represent predictive features and the final column indicates the target variable (housing price). The task description specifies the modeling objective and evaluation criteria [201], such as predicting housing prices with the R^2 metric, classifying images using the F_1-score, or detecting clusters using the Adjusted Rand Index (ARI).

In practice, ML tasks exhibit substantial diversity in structure, scale, and modality, encompassing numerical data, text, images, time series, and other complex formats [77]. This diversity reinforces the need for workflow

designs that are flexible, reusable, and quality-assured, aligning with SE principles to manage complexity and support long-term maintainability.

2.2 Background on LLMs

Large Language Models (LLMs) are advanced neural networks, typically built on Transformer architectures [167], that excel at processing and generating human-like text. Flagship examples such as OpenAI's GPT-4 [1], Google's PaLM-2 [7], and Meta AI's LLaMA-3 [164] have redefined natural language processing (NLP), achieving state-of-the-art performance in tasks like text generation, translation, summarization, and question answering [46, 53, 120]. Beyond traditional NLP, LLMs are increasingly applied in diverse domains, including SE [70] and ML workflow automation [18], due to their ability to generalize across modalities and interact with heterogeneous tools and systems [110, 182, 192].

LLMs often comprise tens to hundreds of billions of parameters [171, 207] and are trained on vast corpora, enabling them to capture complex patterns and generate coherent, context-aware outputs. Key capabilities include zero-shot and few-shot learning, where models adapt to new tasks with minimal task-specific data [143], and multi-step reasoning, enabled by prompting techniques like Chain-of-Thought (CoT) [176], Tree-of-Thought (ToT) [187], and Graph-of-Thought (GoT) [14]. LLMs can also be augmented with external knowledge bases [33], model context protocol (MCP) [71], and tool-use mechanisms [41, 212], enabling them to operate as autonomous agents capable of sensing, reasoning, and acting within complex environments [206]. In SE-oriented ML workflow construction, these capabilities allow LLMs to dynamically integrate heterogeneous resources, orchestrate modular workflow components, and adapt pipeline generation to evolving requirements—thereby improving maintainability, scalability, and collaboration between human developers and AI agents.

Despite these strengths, several limitations impact their integration into engineering workflows. Training and deployment are computationally intensive, demanding substantial hardware resources [9, 197]. LLMs may exhibit hallucinations—producing factually incorrect or unfaithful outputs [102, 103]—which threaten reliability, especially in safety-critical systems. Their stochastic nature means identical inputs can yield different outputs [170], complicating reproducibility and traceability. In addition, large-scale training corpora may contain sensitive or proprietary data, raising concerns over privacy, security, and compliance in regulated domains [186, 189]. These risks underscore the importance of quality assurance, verification pipelines, and risk mitigation strategies when deploying LLMs in SE contexts.

In this survey, we examine how these capabilities and limitations shape the use of LLMs in constructing and optimizing ML workflows across all stages. Our goal is to provide a software engineering–oriented synthesis of current methods, highlight practical and theoretical implications, and outline opportunities for developing LLM-assisted workflow automation.

3 SURVEY METHODOLOGY

3.1 Research Questions

This survey has been oriented and organized to answer the following research questions (RQs):

- **RQ1:** How are LLMs applied to support the Data and Feature Engineering stage in ML workflows?
- **RQ2:** How are LLMs utilized in the Model Selection and Hyperparameter Optimization stage of ML workflows?
- **RQ3:** How are LLMs employed to enhance the Workflow Evaluation stage in ML workflows?
- **RQ4:** What cross-stage trends and variations can be identified, and what are their implications for Software Engineering?

Through RQ1–RQ3, we analyze how LLMs are applied at different stages of the ML workflow. Building on these stage-specific findings, RQ4 integrates cross-stage trends and variations to reveal their broader implications for software engineering. This synthesis also surfaces key open challenges and outlines future directions aimed

at closing current gaps and enabling more robust, maintainable, and trustworthy ML workflow construction in SE contexts.

3.2 Paper Collection

To address the aforementioned research questions, we systematically collect relevant published articles on LLM-enhanced ML workflows. The criteria for paper selection are described as follows [155, 205].

Search Strategy. Following the methodology of [200, 205], we select three widely used scientific databases as primary sources for this survey: Google Scholar, DBLP, and arXiv. These repositories encompass a vast number of computer science publications, offering extensive search capabilities and openly accessible information.

To retrieve relevant papers from these repositories, we primarily rely on a predefined set of search keywords derived from prior studies. Our search query is designed to integrate two distinct sets of keywords: one focusing on ML workflows and the other on LLMs. An article is more likely to be relevant if it contains keywords from both categories. The complete list of search keywords is as follows:

- **Keywords related to ML Workflows:** *Machine Learning Workflow, Machine Learning Pipeline, Machine Learning Engineering, Deep Learning Workflow, Deep Learning Pipeline, Data Analysis Workflow, Data Analysis Pipeline, Data Science Workflow, Data Science Pipeline, Automated Machine Learning, AutoML, Data Preprocessing, Data Preparation, Data Engineering, Data Augmentation, Data Generation, Feature Engineering, FE, Feature Selection, Feature Extraction, Model Selection, Model Generation, Hyperparameter Optimization, HPO, Neural Architecture Search, NAS, Model Evaluation, Workflow Evaluation.*
- **Keywords related to LLMs:** *Large Language Model, LLM, Language Model, LM, Multimodal Large Language Model, MLLM, Foundation Model, Pre-trained, Pre-training, PLM, Natural Language Processing, NLP, ChatGPT, GPT, Agent, Multi-Agent.*

Inclusion and Exclusion Criteria. Following the keyword search, we initially retrieved a collection of articles potentially relevant to our research. To refine this selection, we conducted a relevance assessment based on our predefined inclusion and exclusion criteria [70], as outlined in Table 1. This step ensures that the chosen articles directly address our research questions (RQs). Subsequently, the first two authors conducted a thorough review of the articles to minimize the risk of incorrect inclusion or exclusion. They independently examined the abstracts and introductions of all retrieved articles, evaluated their relevance to LLM-enhanced ML workflows, and ensured the overall quality of the selected studies.

Table 1. Inclusion criteria and exclusion criteria.

Inclusion Criteria
(1) The paper must have more than 5 pages.
(2) The paper must have an accessible full text.
(3) The paper explicitly states that an LLM has been utilized.
(4) The paper claims that the study involves an ML workflow task.
Exclusion Criteria
(1) The paper is not written in English.
(2) The paper is a duplicate or a highly similar study by the same authors.
(3) The paper is a literature review or survey.
(4) The paper mentions the use of LLMs but does not provide details on the techniques employed.
(5) The paper primarily focuses on improving LLMs using ML workflow-related methods, rather than applying LLMs to ML workflow tasks.

Snowballing. To achieve comprehensive coverage and mitigate the risk of overlooking relevant studies, we extended our search beyond the initially selected papers. Specifically, we analyzed both the references cited within these papers and subsequent publications that have cited them. Additionally, we iteratively applied our selection criteria to refine the set of relevant studies. As a result, we finalized a collection of 34 papers for our survey and analysis, all published after 2022 (the release year of ChatGPT). The publication count has increased each year, indicating that this is a rapidly emerging research field attracting growing attention.

Table 2. Comparison of existing methods that employ LLMs for constructing and optimizing ML workflows in terms of specific workflow components.

Year	Method	Data Preprocessing	Feature Engineering	Model Selection	Hyperparameter Optimization	Workflow Evaluation
2022	LMPriors [23]		✓			
2022	ZAP [133]			✓		
2023	VIDS [60]	✓				
2023	JarviX [109]	✓				
2023	Aliro [22]	✓				
2023	AutoML-GPT [203]	✓		✓	✓	✓
2023	MLCopilot [201]			✓		
2023	GENIUS [208]			✓	✓	
2023	GPT-NAS [191]			✓	✓	
2024	AutoMMLab [186]	✓		✓	✓	
2024	AutoML-Agent [165]	✓		✓	✓	
2024	AutoM^{3}L [113]	✓	✓	✓	✓	
2024	Data Interpreter [69]	✓	✓	✓	✓	
2024	SELA [21]	✓	✓	✓	✓	
2024	AutoKaggle [105]	✓	✓	✓	✓	
2024	Agent K v1.0 [50]	✓	✓	✓	✓	
2024	Text-to-ML [184]	✓	✓	✓		✓
2024	LLM-Select [84]		✓			
2024	GL-Agent [177]		✓	✓		
2024	CAAFE [68]		✓			
2024	HuggingGPT [147]			✓		
2024	ModelGPT [159]			✓		
2024	VML [182]			✓		✓
2024	GE [121]				✓	
2024	AgentHPO [108]				✓	✓
2024	LLAMBO [110]				✓	
2025	Data Formulator 2 [169]	✓				
2025	Agent Laboratory [144]	✓		✓	✓	✓

* The symbol ✓ denotes that LLMs are applied to support a particular component within the method.

Additionally, we adopted a systematic analysis framework for our survey, in which each selected study was carefully coded and categorized according to the ML workflow stage(s) it addresses and the manner of LLM integration. This structured approach ensured rigorous comparison across methods—for example, grouping similar techniques and contrasting their outcomes—thereby enhancing the reproducibility of our review process.

Specifically, we categorized 28 papers based on the ML workflow stages where LLMs have been applied, as summarized in Table 2, which organizes the entries chronologically (from the earliest to the most recent) and aligns them with individual workflow components. The remaining 6 studies—ResearchAgent(2024) [77], CodeActAgent(2024) [174], DS-Agent(2024) [58], LAMBDA(2024) [153], AIDE(2025) [88], and AutoMind(2025) [132]—target end-to-end code generation for ML tasks rather than specific workflow components, and thus are not included in Table 2. Notably, upon careful investigation, all the surveyed methods have been practically validated through benchmark experiments or real-world use cases, rather than being purely conceptual.

4 ANSWERING RQ1: HOW ARE LLMS APPLIED TO SUPPORT THE DATA AND FEATURE ENGINEERING STAGE IN ML WORKFLOWS?

Upon receiving the task specification, the first stage in the ML workflow is data and feature engineering, which can be further divided into two key substeps: data preprocessing and feature engineering. Data preprocessing involves cleaning, transforming, and normalizing raw data to ensure consistency and quality for subsequent analysis. Feature engineering focuses on extracting informative and relevant features from the preprocessed data to enhance the performance of learning algorithms. The following subsections will explore how LLMs can support these two crucial processes.

4.1 Data Preprocessing

In many practical scenarios, the qualitative properties of raw data are not often consistent with the requirements of the target application or model [134, 195]. Consequently, data preprocessing has become an essential task in the machine learning application development process. Data preprocessing is typically divided into three key aspects: data acquisition, data cleaning, and data augmentation [62]. Each aspect plays a crucial role in ensuring the quality and usability of the data before it is fed into machine learning models.

- Data Acquisition - involves identifying and gathering suitable datasets.
- Data Cleaning - filters noisy or inconsistent data to preserve the integrity of model training.
- Data Augmentation - enhances model robustness by artificially increasing the size and diversity of the dataset.

4.1.1 Data Acquisition. Data acquisition forms the foundation of the machine learning workflow by ensuring that the right data is sourced for model training and validation. This stage is critical because the quality and relevance of the collected data directly impact the final model's performance [210]. The goal is to identify and gather datasets that align with the task at hand, which can be a time-consuming and labor-intensive process.

LLMs have emerged as powerful tools to streamline the data acquisition process by minimizing the need for extensive manual effort and domain-specific expertise. They are capable of interpreting task requirements, identifying pertinent data sources, and generating comprehensive dataset reports, thereby expediting the workflow. Approaches for LLM-assisted data acquisition generally fall into two main categories: dataset recommendation and dataset summarization.

Dataset Recommendation. In dataset recommendation, LLMs are used to align data selection with task requirements by combining external resource querying and internal knowledge. AutoMMLab [186] enables natural language interaction with users and retrieves domain-relevant datasets from a curated "dataset zoo," supporting customization for specialized ML tasks. AutoML-Agent [165] enhances this by deploying a Data Agent that queries public repositories (e.g., HuggingFace, Kaggle) based on task descriptions. When available, retrieved metadata is incorporated into the prompt to refine selection; otherwise, the agent relies on the LLM's prior knowledge to produce fallback suggestions.

However, LLM-based dataset acquisition can also introduce failure modes, particularly when relying on vague prompts, mismatched modalities, or incomplete metadata. These issues may lead to low-quality or irrelevant inputs that compromise downstream training stability and fairness. Agent Laboratory [144] addresses this by assigning

an ML Engineer agent to not only retrieve datasets but also iteratively generate, execute, and debug preprocessing code. By validating each step through real-time execution and compiler checks, the system ensures that selected datasets are usable and that the resulting pipelines are both correct and aligned with task requirements, enhancing the overall robustness of the workflow.

Dataset Summarization. LLMs are also increasingly used to summarize datasets, enhancing interpretability and guiding automated preprocessing. Early systems such as VIDS [60] and JarviX [109] focus on generating **human-readable summaries**, including schema analysis, value distributions, and column type detection. More recent systems emphasize **machine-actionable summaries**. SELA [21] constructs an insight pool from task descriptions and dataset metadata, enabling retrieval-augmented preprocessing suggestions. AutoKaggle [105] introduces a task-aware Reader Agent that parses file structure and objective descriptions to support code generation. Agent K v1.0 [50] integrates summarization directly into workflow planning, jointly analyzing task goals and dataset layout to guide end-to-end pipeline synthesis. Together, these methods reflect a shift from passive data profiling to active, context-aware summarization modules—enabling modular, reusable, and scalable preprocessing in LLM-driven ML workflows.

4.1.2 Data Cleaning. Data cleaning is a critical phase in ML workflows, ensuring data quality by addressing noise, inconsistencies, and missing values—issues that directly affect model reliability [57, 85]. Traditional AutoML systems often adopt rule-based heuristics for cleaning tasks such as error detection, imputation, and normalization [124]. However, recent advances show that integrating LLMs enables more intelligent, context-aware, and modular preprocessing strategies, especially in code generation settings where flexibility and automation are essential. These LLM-assisted methods generally fall into two interrelated categories: adaptive imputation and preparation, and context-enhanced transformation.

Adaptive Imputation and Preparation. In this paradigm, LLMs are used to dynamically fill in missing values and synthesize task-aware preprocessing code [21, 50, 69, 105]. For example, AutoM^{3}L [113] features a dedicated component, AFE-LLM$_{imputed}$, where the LLM actively interprets the surrounding data context, applies tailored prompt strategies, and performs semantic reasoning to infer and impute missing values. This LLM-driven contextual reasoning substantially improves data completeness, enabling downstream modules to operate on cleaner and more reliable inputs. Building upon this idea, Text-to-ML [184] adopts an iterative refinement loop where LLM-generated data preparation modules—covering loading, cleaning, and formatting—are progressively improved based on runtime feedback. This feedback-driven loop not only improves robustness but also mirrors SE practices such as test-and-refine cycles.

Context-enhanced Cleaning and Transformation. While adaptive approaches focus on basic value recovery and preparation, context-aware methods leverage task semantics and dataset characteristics to support richer, more informed transformations. These methods differ notably in their context sources and interaction styles.

For example, JarviX [109] uses pre-analyzed statistics and attribute correlations to guide the LLM in recommending operations such as removing outliers from an income column based on its correlation with education level. Although lightweight, this approach is constrained by the scope of precomputed insights. AutoML-GPT [203] instead draws on task descriptions and domain priors to recommend preprocessing steps like resizing images or tokenizing text—offering stronger modality-specific support but relying heavily on accurate task inputs and lacking interactivity.

In contrast, Aliro [22] and Data Formulator 2 (DF2) [169] incorporate interactive elements. Aliro allows users to engage with LLMs through a chat interface linked to live data visualizations, while DF2 interprets GUI operations and natural language intents to synthesize transformation code. These approaches offer improved usability and transparency but may require repeated user involvement and are less suited to fully automated pipelines.

As summarized in Table 3, these systems span a spectrum from static, prompt-based approaches to dynamic, interaction-driven frameworks.

Table 3. Comparison of context-enhanced cleaning and transformation methods across representative systems.

System	Context Source	Example Operation	Interaction Style
JarviX [109]	Pre-analyzed data statistics and correlations	Suggests removing outliers from an income column based on its correlation with education level	Prompt
AutoML-GPT [203]	Task descriptions and domain priors	Recommends resizing images to 224×224 and normalizing pixel values depending on the task description	Prompt
Aliro [22]	User interaction with visualizations	Generates code suggestions to remove selected outliers based on user interactions with PCA plots	Prompt + UI
DF2 [169]	GUI operations and natural language intents	Interprets user-defined chart configurations (e.g., dragging "Rank" to the Y-axis) and simple instructions (e.g., "sort by renewable energy percentage") to generate transformation code	Prompt + UI

Importantly, while some methods emphasize adaptive imputation and others prioritize context-aware transformation, these strategies are sometimes combined in practice to enable end-to-end data cleaning. Systems such as SELA [21], AutoKaggle [105], and Data Interpreter [69] integrate dataset summarization with transformation guidance, allowing LLMs to both interpret task metadata and retrieve or compose cleaning components (e.g., `FillMissingValues`, `ConvertDataTypes`, `FormatDatetime`, `MinMaxScale`). Rather than treating these stages in isolation, modern LLM-driven frameworks adopt a unified preprocessing flow that supports configurable, reusable, and task-adaptive cleaning operations. This reflects a shift toward workflow modularization and automation, aligning with SE principles such as separation of concerns [160], code reusability [118], and maintainability [95] in ML pipeline construction.

4.1.3 Data Augmentation. Data augmentation refers to techniques used to artificially increase the size of the training dataset by creating enhanced versions of existing data. This step is particularly useful for enhancing model robustness, improving generalization, and avoiding overfitting, especially in scenarios where data is scarce [150]. Recent systems have started to leverage LLMs not only to recommend augmentation strategies but also to synthesize data in task-specific and controllable ways.

Original Data Modification. A common approach is modifying existing inputs through LLM-generated transformation code. For instance, the Data Agent in AutoML-Agent [165] recommends operations such as flipping, rotation, and zooming based on the input modality and task context. These fine-grained transformations can be directly embedded into pipelines, offering a lightweight and adaptable solution. However, their effectiveness may be insufficient for abstract or imbalanced domains [8].

Synthetic Data Generation. Beyond modifying inputs, LLMs also enable synthetic data generation, producing entirely new examples that approximate the distribution of the original data. Early efforts rely on single PLMs to generate labeled synthetic samples for downstream training tasks [45, 193], demonstrating promising results in NLP and tabular domains. However, such methods may inherit biases from the base model and lack diversity. To mitigate this, FuseGen [215] introduces a collaborative generation approach by aggregating outputs from multiple PLMs, which improves data diversity and reduces overfitting risks—particularly important in SE contexts where pipeline-generated data must generalize well across tasks without incurring excessive computational costs. At the industrial scale, Nemotron-4 [2], a large generative model from NVIDIA, has been integrated into full-stack data generation pipelines across domains such as healthcare, retail, and manufacturing. Its strength lies not only in producing high-quality, large-scale datasets but also in adapting generation to domain-specific constraints—making it a suitable component for production-grade ML code generation systems.

In summary, original data modification offers lightweight integration and task adaptivity, while synthetic generation provides greater diversity and scale. Although LLM-based augmentation is still emerging, these methods show promise for bridging the gap between limited real-world data and the demands of robust, automated ML pipeline construction [47].

4.1.4 Summary. Analyzing the above, LLMs demonstrate considerable potential in automating and enhancing data preprocessing—spanning acquisition, cleaning, and augmentation—while also presenting practical constraints that require careful mitigation. Table 4 summarizes representative capabilities and limitations for each preprocessing type.

Table 4. Capabilities and limitations of LLMs for data preprocessing in ML workflows within the SE context.

Data Preprocessing Type	Example Systems	LLM Capabilities	LLM Limitations
Data Acquisition	AutoMMLab [186], AutoML-Agent [165], Agent Laboratory [144], VIDS [60], JarviX [109], SELA [21], AutoKaggle [105], Agent K v1.0 [50]	• Interpret task requirements and recommend relevant datasets • Reduce manual search effort and speed up pipeline setup via metadata filtering	• Retrieval errors when prompts/metadata are vague or incomplete • Difficulty integrating heterogeneous sources without schema alignment • Potential fairness and stability issues from mismatched datasets
Data Cleaning	AutoM^{3}L [113], Text-to-ML [184], JarviX [109], AutoML-GPT [203], Aliro [22], DF2 [169], SELA [21], AutoKaggle [105], Data Interpreter [69]	• Generate imputation and transformation logic dynamically • Support modular and reusable preprocessing code • Enable task-adaptive cleaning aligned with SE principles	• Misinterpret complex or domain-specific patterns • Generated code may lack correctness, testability, or robustness • Requires human-in-the-loop validation for production use
Data Augmentation	AutoML-Agent [165], FuseGen [215], Nemotron-4 [2]	• Recommend and generate transformations to improve robustness • Synthesize realistic, task-specific samples at scale • Adapt augmentation to domain constraints in industrial contexts	• Risk of bias amplification or irrelevant content • Traceability and privacy compliance challenges • Hard to ensure representativeness of synthetic data

In **Data Acquisition**, LLMs accelerate the identification and retrieval of relevant datasets by interpreting task intent, querying repositories, and leveraging metadata. However, these benefits are contingent on accurate task understanding and the availability of high-quality metadata; without validation mechanisms such as schema alignment checks, content filtering, or runtime verification, irrelevant or low-quality data can propagate through the pipeline, undermining downstream stability and fairness.

For **Data Cleaning**, LLMs enable dynamic imputation, transformation, and modular preprocessing components that align well with SE principles like maintainability, reusability, and separation of concerns. This modularity supports both automated workflows and human-in-the-loop refinement. Yet, the interpretive nature of LLM outputs means that complex or domain-specific anomalies may be misclassified, and generated code can suffer from correctness, testability, and robustness issues. Embedding automated unit testing, formal verification, or static analysis into LLM-driven cleaning modules could mitigate these risks in production-grade settings.

In **Data Augmentation**, LLMs can both recommend lightweight transformations and synthesize realistic, task-specific samples at scale, improving robustness in low-resource scenarios. These capabilities are particularly attractive in domains where acquiring labeled data is expensive or impractical. Nevertheless, augmentation carries unique SE challenges, including the amplification of bias, privacy concerns, and difficulty in ensuring the traceability and representativeness of synthetic data. Techniques such as provenance tracking, bias auditing, and domain-constrained generation are critical to integrating augmentation safely in SE-compliant workflows.

From a broader SE perspective, these findings highlight a recurring trade-off: while LLM-based automation enhances efficiency, adaptability, and modularity, it simultaneously raises the bar for quality control, transparency, and long-term maintainability. Addressing these concerns will require combining LLM capabilities with robust engineering safeguards, enabling ML code generation pipelines that are not only more intelligent but also reliable, reproducible, and aligned with established SE practices.

4.2 Feature Engineering

While ML models built on deep neural architectures can automatically learn useful features [11], certain application settings still require explicit feature processing before model training to ensure optimal performance. Feature engineering is the process of extracting and refining relevant features from raw data, a critical task that can significantly improve predictive accuracy [68]. This process typically involves three key subtopics: feature selection, feature extraction, and feature synthesis [123].

- Feature Selection - is the process of choosing the most significant features that reduce feature redundancy and improve model performance by focusing on the most relevant attributes.
- Feature Extraction - aims to create more robust, representative and compact features by applying specific mapping functions to the raw data.
- Feature Synthesis - involves generating new features from existing ones, creating richer representations that can better capture the underlying patterns in the data.

4.2.1 Feature Selection. Feature selection involves creating a subset of features from the original set by removing irrelevant or redundant ones [62]. This process simplifies the model, reduces overfitting risks, and improves overall performance [134]. The selected features are typically diverse and highly correlated with the target variables, ensuring they meaningfully contribute to the model's predictions.

LLMs are increasingly used to automate feature selection, leveraging their ability to understand the semantic context of datasets. In the LMPriors framework [23], LLMs are prompted to evaluate whether each candidate feature should contribute to predicting the target outcome. Features are selected based on the difference in log probabilities between generating a "Y" (Yes) or "N" (No) token, crossing a predefined threshold. On the other hand, LLM-Select [84] directly uses the generated text output instead of token probabilities to assess feature relevance. Here, LLMs analyze textual descriptions of features and their relationships to the target task, helping identify those most relevant for the model. This semantic-driven approach reduces model complexity by selecting features that are crucial for predictive performance. Similarly, AutoM^{3}L [113] incorporates contextual signals—such as column names, inferred modalities, and user instructions—into its AFE-LLM$_{filter}$ component for feature filtering. On multimodal datasets with masked values and injected noise, this automated feature engineering consistently improved performance by removing irrelevant attributes and imputing missing data, whereas AutoGluon [37] and AutoKeras [90] showed notable degradation.

However, LLM-based feature selection can also introduce undesirable bias. Since LLMs are trained on large-scale web corpora, they may internalize social or demographic stereotypes [43]. For example, when analyzing a dataset related to employment or health, biased priors might lead the model to overemphasize sensitive features such as gender or race, even if these attributes are only weakly correlated with the target variable. In the case of LMPriors, the prompt-based decision-making relies on the model's learned priors and may lack fairness-awareness when selecting features. Similarly, semantic reasoning in LLM-Select may amplify implicit associations from training data, resulting in skewed feature representations that favor majority groups.

To mitigate these risks, researchers have proposed integrating LLM-based methods with traditional data-driven metrics (e.g., mutual information, statistical correlation) or using human-in-the-loop validation to screen sensitive or potentially biased features [84]. Another avenue is fairness-aware prompting or group-sensitive evaluation to

explicitly assess the downstream impact of selected features. Nonetheless, challenges remain in achieving robust, fair, and explainable feature selection at scale.

4.2.2 Feature Extraction. Feature extraction is a dimensionality reduction technique that transforms the original features using mapping functions to extract informative, non-redundant features based on specific criteria. Unlike feature selection, which retains original features, feature extraction modifies them to produce new representations. Common methods for feature extraction include principal component analysis (PCA) [117], linear discriminant analysis (LDA) [181], and autoencoders [63].

LLMs have shown considerable promise in feature extraction, particularly for handling complex, multimodal datasets and generating semantically rich features. For example, in the LLM-based Versatile Graph Learning (GL-Agent) approach [177], LLMs are used to select appropriate feature engineering strategies that capture both the structural and semantic properties of graph data, tailoring them to the specific learning task and evaluation metrics. Similarly, the Text-to-ML method [184] automates the extraction and transformation of raw features across numerical, text, and image data, generating task-specific code that is subsequently validated using automatically generated unit tests and synthetic data. This process ensures consistency in the extracted features across different data types and tasks.

4.2.3 Feature Synthesis. Feature synthesis involves utilizing the statistical distribution of extracted features to generate new, complementary ones. This approach is especially valuable when existing features are insufficient to adequately represent the input data [123]. Traditionally, this process relied heavily on human expertise for tasks like standardization and feature discretization. However, manually exploring all possible feature combinations is impractical. As a result, automatic feature construction methods, including tree-based, genetic algorithm-based, and reinforcement learning-based approaches [166, 198], have been developed and have demonstrated performance comparable to or even surpassing human-designed features.

LLMs have proven particularly effective in generating semantically and contextually relevant synthetic features, enhancing the feature synthesis process through an interpretable, human-in-the-loop approach. In the Context-Aware Automated Feature Engineering (CAAFE) framework [68], LLMs iteratively generate additional meaningful features for tabular datasets based on the dataset description, as shown in Fig. 3. This approach not only generates Python code for feature creation but also provides explanations of the relevance and utility of the generated features. CAAFE represents a significant advancement in semi-automated data science tasks, focusing on context-aware solutions that move AutoML systems toward more interpretative and human-centered workflows [107].

Furthermore, the trend in LLM-assisted feature engineering is increasingly characterized by the integration of multiple subtopics, reflecting a more flexible, adaptive, and modular approach. For instance, systems like Data Interpreter [69] and AutoKaggle [105] utilize LLMs to dynamically select and combine multiple operators from pre-defined FE tools and libraries. These systems cover a broad spectrum of tasks, including Feature Selection (e.g., `GeneralSelection`, `VarianceBasedSelection`), Feature Extraction (e.g., `MinMaxScale`, `PCA`), and Feature Synthesis (e.g., `KFoldTargetMeanEncoder`, `CreateFeatureCombination`), enabling the construction of highly customizable and task-specific FE pipelines. Similarly, frameworks like SELA [21] and Agent K v1.0 [50] leverage evaluation score feedback to iteratively refine the generated FE code and parameter configurations. This capability highlights how LLMs are enabling a more holistic and efficient approach to feature engineering, seamlessly integrating automation with real-time optimization within the SE context. Nevertheless, the automation of feature selection and synthesis also carries the risk of propagating biases learned during pretraining [113].

4.2.4 Summary. Early studies highlight the potential of LLMs in automating feature engineering within ML workflows, spanning feature selection, extraction, and synthesis. Recent systems increasingly combine these subtopics, allowing LLMs to flexibly select and compose operators across stages to build more adaptive and configurable feature engineering pipelines. However, their adoption also raises concerns regarding the reliability,

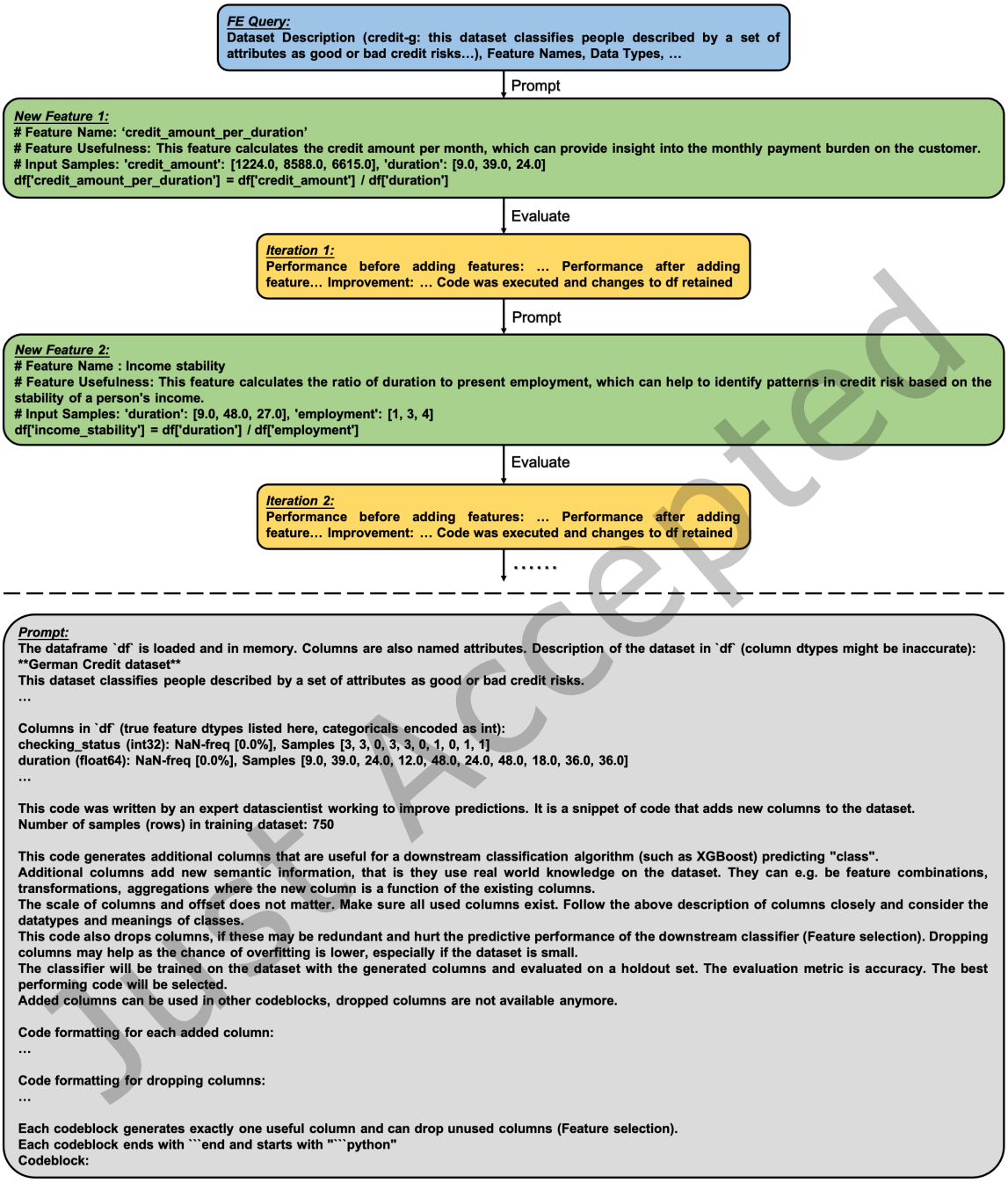


Fig. 3. An example of LLM-assisted feature engineering in CAAFE [68] method.

maintainability, and verifiability of generated code in SE-oriented workflows. Table 5 provides an overview of the capabilities and limitations of LLM-assisted feature engineering across various subtopics.

Table 5. Capabilities and limitations of LLMs for feature engineering in ML workflows within the SE context.

Feature Engineering Type	Example Systems	LLM Capabilities	LLM Limitations
Feature Selection	LMPriors [23], LLM-Select [84], AutoM^{3}L [113]	• Filter irrelevant/redundant attributes using metadata and task descriptions • Reduce manual tuning effort and improve pipeline efficiency	• Bias from pretraining data may skew feature selection • Semantically relevant features may lack statistical strength or generalizability
Feature Extraction	GL-Agent [177], Text-to-ML [184]	• Generate task-specific transformation code across modalities • Accelerate pipeline construction and improve scalability	• Extracted features may not capture data distribution or domain nuances • Generated code may lack robustness and require extra validation for maintainability
Feature Synthesis	CAAFE [68]	• Create new, contextually meaningful features with code snippets and explanations • Support iterative refinement via human-in-the-loop feedback	• High-dimensional data may lead to oversized prompts and inefficiency • Hallucinations may introduce invalid or irrelevant features without proper verification

* Data Interpreter [69], AutoKaggle [105], SELA [21], and Agent K v1.0 [50] span all three categories of FE.

For **Feature Selection**, LLMs can filter irrelevant or redundant attributes based on dataset metadata and task descriptions [23, 84, 113], reducing manual effort and improving pipeline efficiency. This semantic reasoning helps accelerate early-stage development while maintaining focus on relevant attributes. Nonetheless, LLMs may inherit biases from their pretraining corpora [43], leading to skewed feature prioritization or unfair performance across subpopulations. Moreover, semantically plausible features may lack statistical robustness, limiting their contribution to predictive performance. These risks highlight the importance of hybrid selection strategies—combining LLM reasoning with statistical metrics—and human-in-the-loop validation to meet SE standards for fairness, traceability, and reproducibility.

For **Feature Extraction**, LLMs can automatically generate transformation code to produce compact, representative features across different modalities [177, 184]. This capability reduces manual trial-and-error and scales to heterogeneous data sources, facilitating rapid prototyping. However, ensuring that extracted features accurately capture the underlying data distribution and domain-specific nuances remains challenging. Inadequate or overgeneralized transformations can silently degrade downstream model quality. From an SE perspective, extracted features require rigorous validation, automated testing, and robust documentation to ensure long-term maintainability and correctness.

For **Feature Synthesis**, LLMs are able to create new, contextually meaningful features with both executable code and human-readable explanations [68], enriching the representational space and improving interpretability. While this strengthens transparency and supports collaborative debugging, high-dimensional data can lead to overly large prompts and increased computational cost. Moreover, LLMs may hallucinate plausible yet invalid features, introducing noise or unnecessary complexity into the pipeline. Without strict verification and version control, these synthetic features can compromise both interpretability and maintainability in production-grade workflows.

Overall, LLM-assisted feature engineering extends the automation frontier beyond traditional AutoML approaches, offering unprecedented flexibility, modularity, and cross-modal adaptability. Yet, these benefits come with new SE-relevant risks, including bias propagation, statistical fragility, hallucinations, and lifecycle maintenance challenges. Addressing these issues through integrated verification pipelines, fairness-aware prompting,

and continuous quality monitoring will be essential for the safe, robust, and scalable adoption of LLM-based feature engineering in industrial ML workflows.

Summary for RQ1: Existing studies indicate that LLMs primarily support the early stages of ML workflows through two key dimensions: *data preprocessing* and *feature engineering*:

(1) For data preprocessing, three directions are frequently explored: data acquisition, cleaning, and augmentation. Among them, acquisition and cleaning receive the most attention, with recent work focusing on context-aware summarization, adaptive data preparation, and code generation for reusable cleaning components. Data augmentation is emerging as a complementary direction but remains less explored.

(2) For feature engineering, LLMs are employed across feature selection, extraction, and synthesis. Earlier work leveraged LLMs' semantic reasoning over dataset metadata to identify relevant attributes, while more recent methods generate transformation code, create novel features, and iteratively refine operator choices based on evaluation feedback. State-of-the-art frameworks integrate these capabilities into hybrid pipelines, enabling dynamic operator composition and cross-modal adaptability—properties that directly align with SE goals of configurability, modularity, and lifecycle maintainability.

From an **SE perspective**, LLM-driven data preprocessing and feature engineering can significantly reduce manual coding effort, accelerating prototyping and boosting productivity. However, production-grade adoption demands maintainability, interpretability, and traceability of the generated logic, supported by rigorous verification, automated testing, human-in-the-loop review, comprehensive documentation with audit trails, and version control for reproducibility. These measures are essential to ensure long-term software quality and reliability in AI-enhanced development processes.

5 ANSWERING RQ2: HOW ARE LLMS UTILIZED IN THE MODEL SELECTION AND HYPERPARAMETER OPTIMIZATION STAGE OF ML WORKFLOWS?

Model Selection (MS) and Hyperparameter Optimization (HPO) play a pivotal role in ML workflows, directly affecting model performance and generalizability. Model selection involves choosing the most suitable algorithm or architecture for a given task [185], while hyperparameter optimization fine-tunes key configuration settings such as learning rate, regularization strength, and the number of layers [168]. A related task, Combined Algorithm Selection and Hyper-parameter Optimization (CASH), jointly addresses MS and HPO by treating algorithm choice as a hyperparameter to be optimized [161].

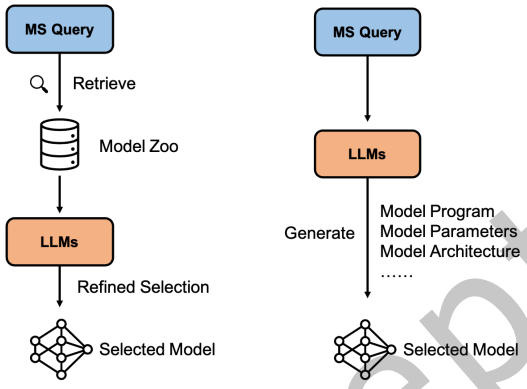
These two processes are critical for balancing model complexity, training efficiency, and predictive accuracy. Effective MS and HPO ensure that models generalize well to unseen data, mitigating the risks of overfitting or underfitting. Recently, LLMs have shown promise in enhancing MS and HPO by leveraging contextual understanding and task reasoning to recommend candidate models, propose initial hyperparameters, and generate optimization scripts that iteratively adjust configurations based on feedback [108, 182, 201]. In the following subsections, we discuss how LLMs are being applied to automate and improve these two crucial stages in ML workflow construction.

5.1 Model Selection

Model selection refers to the process of identifying the most appropriate algorithm or model architecture for a specific ML task. Candidate models, as shown in Fig. 1, are generally divided into two categories: traditional models, such as decision trees [138] and naive Bayes classifiers [140], and deep networks, such as convolutional neural networks (CNNs) [100] and recurrent neural networks (RNNs) [64], which are more frequently employed for tasks involving unstructured data like images or sequential inputs. Traditionally, model selection has relied heavily on trial-and-error experimentation, domain expertise, and substantial computational resources, requiring

human experts to manually test and compare multiple models [106]. The choice of model directly impacts predictive accuracy, interpretability, and the scalability of the full pipeline.

LLMs are increasingly being leveraged to streamline and partially automate model selection. By interpreting natural language descriptions of tasks and datasets, LLMs can retrieve candidate algorithms based on their pre-trained knowledge or generate code snippets for suitable model configurations, reducing the need for exhaustive manual experimentation. Current methods for LLM-assisted model selection can be broadly categorized into two approaches: **retrieval-based** and **generation-based**, as illustrated in Fig. 4.



(a) Retrieval-based (b) Generation-based

Fig. 4. Two main approaches for LLM-assisted model selection in ML workflow.

5.1.1 Retrieval-based Model Selection. Retrieval-based model selection generally follows a two-step paradigm: first, construct a repository of candidate models with detailed metadata; then, use LLMs to retrieve and select the most suitable ones based on task and model descriptions. While this approach emphasizes modularity, reusability, and explainability, frameworks differ in how they organize repositories, perform retrieval, and integrate results into downstream workflows.

(1) Model Zoo–driven Selection. In AutoMMLab [186], the authors construct a comprehensive model zoo, where each model is paired with a detailed model card and pre-trained weights, taking AI safety concerns into account. Each model card includes attributes such as the model's name, structure, parameters, floating point operations per second (FLOPs), inference speed, and performance metrics. An elaborate pipeline is then designed to automatically select the most appropriate model from the zoo, based on model performance and fuzzy matching scores between the model structure and the user's specified requirements. Similarly, AutoM^{3}L [113] constructs a multimodal model zoo but **addresses context length limitations** by using LLM-powered tools (e.g., ChatPaper [115]) to auto-generate concise model cards. It first retrieves the top-5 models per modality by embedding similarity between requirements and descriptions, and then applies the Model Selection-LLM (MS-LLM) to make the final choice. Beyond its multimodal use cases (Sec. 4.2.1), AutoM^{3}L also demonstrated strong scalability in single-modality settings: on the OpenML tabular benchmark, it achieved competitive or superior performance to popular AutoML frameworks—many of which use ensembles—despite evaluating only a single selected model across binary/multi-class classification and regression tasks.

AutoML-GPT [203] and HuggingGPT [147] employ an in-context task-model assignment mechanism to dynamically select models. This mechanism first filters models that align with the task type, then selects the top-k models based on the number of downloads, from which LLMs choose the most suitable model based on the user's

query and task information provided in the prompt. AutoKaggle [105] leverages LLMs to orchestrate pipeline execution, invoking a predefined tool, `TrainAndValidationAndSelectTheBestModel`, to perform validation across candidate models and return the best-performing option. Through this LLM-driven coordination, it achieves an average valid submission rate of 83% across 8 Kaggle tasks—28% higher than the AIDE framework [88].

(2) Experience-based Retrieval. Model repositories in above approaches rely on explicit model features [132], which is critical for solving ML tasks but can be challenging due to the **heterogeneous nature of the data** (e.g., code, configurations, logs). To address this, MLCopilot [201] introduces a two-stage knowledge-based reasoning approach that leverages LLMs to reason and solve tasks by drawing on **knowledge** from past experiences. In the offline stage, it standardizes historical data into a unified format, creating experience and knowledge pools with the aid of LLMs. During online execution, relevant past experiences are retrieved and combined with LLM reasoning to propose candidate solutions, enhancing reusability and explainability of model selection decisions. In evaluation, both text embedding of task descriptions and meta-feature similarities–based retrieval consistently outperformed random retrieval. Additionally, the Model Agent within the AutoML-Agent framework [165] performs model retrieval and hyperparameter suggestion, guided by **insights** provided by an Agent Manager regarding high-performing models and suitable hyperparameters for the specific ML task. Moreover, [133] presents a zero-shot AutoDL approach that meta-learns from a large pool of pre-trained models to predict the best-performing architecture given dataset meta-features, reducing the need for manual trial-and-error.

Retrieval-based frameworks vary in repository construction (manual vs. automated model cards), retrieval strategies (embedding similarity, meta-feature matching, validation-based ranking), and scope (single- vs. multi-modality, historical experience integration). Overall, they organize model knowledge systematically and pair it with LLM reasoning, enabling configurable and explainable model choice for SE-oriented ML workflow generation.

5.1.2 Generation-based Model Selection. Generation-based approaches leverage the generative capabilities of LLMs to synthesize model selection logic or even entire model architectures from task descriptions. These methods address the limitations of relying solely on predefined models or historical experience, which may not suit the diverse requirements of real-world ML tasks [68, 184].

(1) Code-level Generation. One line of work focuses on automatically generating the model selection component code within ML workflows. For instance, [184] proposes a Contextual Modular Generation framework that decomposes workflows into smaller modules, each independently generated by LLMs. Unit testing ensures compatibility between newly synthesized modules and less variable optimization components. Similarly, Data Interpreter [69] dynamically generates programmable model selection nodes by combining external functions, tool-based operators, and non-tool logic from libraries such as Pandas and NumPy. Frameworks like SELA [21] and Agent Laboratory [144] employ LLM agents to write executable model selection code from stepwise task instructions. Agent K v1.0 [50] extends this idea by adapting generated model designs to different modalities (e.g., tabular, CV, NLP), enabling flexible, task-specific model construction. Benchmarking against 5,856 human Kaggle competitors, it ranked in the top 38% by Elo-MMR—comparable to Expert-level users—and achieved 6 gold, 3 silver, and 7 bronze medals, meeting Kaggle's Grandmaster performance criteria. These methods can also be domain-specialized, as in GL-Agent [177], which configures both the search space and NAS algorithm for graph learning tasks. By referencing domain knowledge and PyG documentation, the agent selects candidate operations, designs hardware-aware search spaces, and recommends suitable NAS algorithms, achieving expert-level modularity and traceability.

(2) Parameter-level Generation. Another line of research focuses on directly generating and optimizing model parameters. The Verbalized Machine Learning (VML) framework [182] represents models and parameters in natural language prompts. An LLM-based optimizer iteratively updates parameters using task data and loss feedback, enabling fully verbalized, automated refinement. ModelGPT [159] translates user data and task descriptions into concise requirements, which are processed by a Model Customizer module to define architecture

and generate parameters via LoRA-assisted [73] hypernetworks, delivering tailored models up to 270× faster than previous paradigms such as full-parameter fine-tuning.

Overall, generation-based model selection methods aim to move beyond static repositories by dynamically generating architectures, parameters, or search strategies. These approaches improve flexibility, automation, and explainability in SE-oriented ML workflow generation, supporting scalable and testable pipeline code construction. Code-level generation allows fine-grained control over model logic and integration, while parameter-level generation accelerates adaptation to new tasks with minimal retraining.

5.1.3 Summary. LLMs provide promising capabilities for automating the model selection stage in ML workflows, improving efficiency, reducing manual experimentation, and supporting more traceable decision-making. Current studies mainly adopt two paradigms: retrieval-based and generation-based, each with distinct strengths and limitations, as summarized in Table 6.

Table 6. Capabilities and limitations of LLMs for model selection in ML workflows within the SE context.

Model Selection Type	Example Systems	LLM Capabilities	LLM Limitations
Retrieval-based	AutoMMLab [186], AutoM^{3}L [113], AutoML-GPT [203], HuggingGPT [147], AutoKaggle [105], MLCopilot [201], AutoML-Agent [165], ZAP [133]	• Leverage model repositories with enriched metadata to align recommendations with task descriptions • Enhance reproducibility and traceability by making selection criteria explicit and reusable • Support explainable decision-making via structured repositories and LLM reasoning	• Depend on repository coverage and metadata quality, costly to maintain • Long-context prompts may reduce retrieval accuracy • Limited adaptability to novel or highly specialized tasks without expert curation
Generation-based	Text-to-ML [184], Data Interpreter [69], SELA [21], Agent Laboratory [144], Agent K v1.0 [50], GL-Agent [177], VML [182], ModelGPT [159]	• Dynamically synthesize model selection logic, architectures, and parameters from natural language • Enable fine-grained, modular code generation for diverse modalities and tasks • Facilitate integration of generated components into broader ML workflows	• Stochastic inference may produce inconsistent or suboptimal choices • Numerical errors or incomplete reasoning can reduce reliability • Generated modules may lack compatibility or correctness without robust validation

For **retrieval-based approaches**, LLMs exploit curated model repositories and metadata-rich model cards to recommend algorithms that align with task descriptions [113, 201]. This explicit linking of model characteristics to selection decisions improves reproducibility, traceability, and reusability across workflows. However, their flexibility depends heavily on the coverage and quality of the repositories, which are costly to maintain and may quickly become outdated. In addition, long-context prompts can degrade retrieval accuracy, and novel or highly specialized tasks often require manual expert curation to supplement existing entries.

For **generation-based approaches**, LLMs directly synthesize model selection logic, architectures, or parameter settings from natural language task descriptions [182, 184], enabling fine-grained, task-specific code generation and seamless integration into larger workflows. While this improves adaptability, stochastic inference can yield inconsistent or suboptimal outputs, and numerical or logical errors may compromise reliability. Even when designs are sound, generated components can face compatibility issues with existing workflow modules, necessitating human review, rigorous validation, and testing—thus limiting full automation.

Across **both paradigms**, persistent challenges include incorporating domain-specific constraints, ensuring explainability of the chosen or generated models, and mitigating hallucinations that produce plausible but incorrect configurations. Addressing these issues will require stronger validation pipelines, standardized model

metadata, and robust quality assurance practices to ensure that LLM-driven model selection remains reliable and aligned with SE principles in ML workflow construction.

5.2 Hyperparameter Optimization

Hyperparameter optimization (HPO), or hyperparameter tuning, is the process of selecting the optimal hyperparameters that enhance a machine learning model's performance. Hyperparameters are configuration parameters set prior to training—such as the maximum depth in decision trees or learning rates in neural networks—and are not learned during training itself. The objective of HPO is to automatically identify the hyperparameter values that maximize a model's performance on a given task [12]. NAS is a specific instance of HPO, focusing on optimizing both the architecture and hyperparameters of neural networks simultaneously [83, 208].

Traditional HPO techniques like grid search, random search, and Bayesian optimization have proven effective [36]. However, these methods are computationally expensive, scale poorly to high-dimensional search spaces, and often require significant manual oversight—posing challenges for maintaining efficiency and reproducibility in ML workflow development. Recently, LLMs have emerged as promising tools to enhance the efficiency and effectiveness of hyperparameter optimization by leveraging their contextual reasoning and prior knowledge to propose optimized hyperparameter configurations. Based on whether actual training and testing are performed for hyperparameter settings, LLM-assisted HPO methods can be broadly categorized into two types: **execution-based** and **prediction-based** HPO, as shown in Fig. 5.

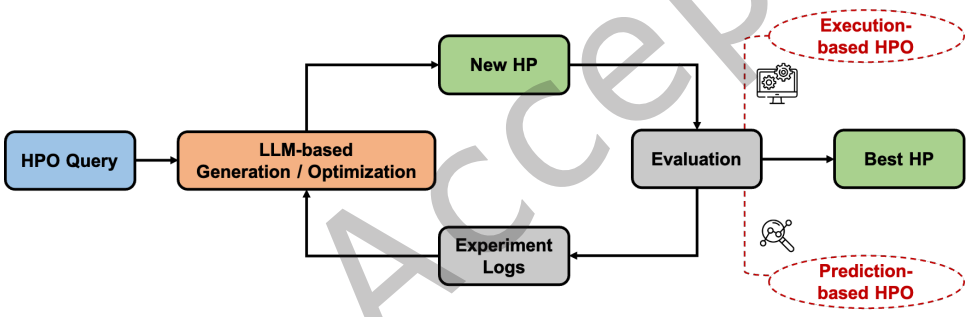


Fig. 5. Two main approaches for LLM-assisted hyperparameter optimization in ML workflow.

5.2.1 Execution-based HPO. Execution-based HPO refers to approaches where hyperparameter settings generated or optimized by LLMs are empirically validated through real model training and evaluation, with iterative refinement driven by observed performance metrics. These methods aim to reduce manual trial-and-error while increasing the automation, auditability, and maintainability of hyperparameter tuning processes in ML workflow code generation.

(1) Direct Hyperparameter Generation. Some approaches focus on generating hyperparameter configurations directly from task descriptions. For instance, AutoMMLab [186] introduces HPO-LLaMA, a fine-tuned LLaMA-7B model that generates and iteratively refines hyperparameters based on dataset and model specifications. Compared to an important random sampling baseline (averaged over 1,000 runs) in the realm of HPO, HPO-LLaMA significantly outperforms in all four representative CV tasks, achieving effective configurations even with a single iteration. Agent Laboratory [144] explores a Combined Algorithm Selection and Hyperparameter Optimization (CASH) paradigm, where the mle-solver agent repeatedly applies `REPLACE` and `EDIT` operations on training code to improve performance scores. Data Interpreter [69] also falls in this category, leveraging binary success/failure signals to adapt hyperparameters and regenerate faulty nodes. While lacking fine-grained numeric

feedback, it demonstrates robust recovery from pipeline failures across 8 Kaggle tabular datasets with minimal user input.

(2) HPO Code Generation. Other systems focus on producing executable HPO routines as part of the ML workflow code. SELA [21] introduces an Experiment Executor agent that generates hyperparameter tuning scripts (e.g., grid search, random search) and iteratively refines the pipeline through Monte Carlo Tree Search (MCTS) guided by final execution scores. Experiments on 20 datasets from the AMLB benchmark [48] show that SELA basically outperforms two traditional AutoML systems and two agent-based baselines. AutoKaggle [105] takes a similar approach, leveraging the `TrainAndValidationAndSelectTheBestModel` tool to programmatically search for and select the best-performing configuration.

(3) Integration with External Optimization Tools. Some frameworks couple LLM reasoning with established search libraries to enhance automation and reduce manual setup. AutoM^{3}L [113] integrates LLM-based interpretation of configuration files with `ray.tune` to automate hyperparameter search. Compared to AutoKeras [90] and AutoGluon [37], AutoM^{3}L achieves comparable or superior accuracy, with a 2.7% improvement on the CH-SIMS dataset. Its key advantage lies in fully automating the HPO process—eliminating manual setup required by other frameworks—while benefiting from community-driven pretrained model hubs (e.g., HuggingFace, Timm). Agent K v1.0 [50] follows a similar design, pairing agent-based reasoning with the HEBO Bayesian optimizer [26] to adjust key hyperparameters (e.g., learning rate, optimizer settings) and select the best-performing model based on validation loss.

(4) NAS-oriented Optimization. Another line of research targets neural architecture search (NAS), combining architecture and hyperparameter optimization in one loop. GENIUS [208] uses GPT-4 as a "black-box" optimizer to iteratively refine architectures and hyperparameters for improved accuracy. GPT-NAS [191] reduces search complexity by letting an LLM generate candidate architecture components, which evolutionary algorithms (EAs) then evaluate and filter efficiently. Guided Evolution (GE) [121] extends this idea by introducing "Evolution of Thought" (EoT) operations, where LLMs iteratively revise code segments representing architectures and hyperparameters, mimicking expert-like genetic refactoring. Applied to the ExquisiteNetV2 model [209], GE autonomously evolved variants that improved accuracy from 92.52% to 93.34% without increasing model size—demonstrating its ability to enhance both performance and compactness. This highlights the potential of LLM-driven NAS to accelerate and automate model design pipelines with minimal human intervention.

5.2.2 Prediction-based HPO. Prediction-based HPO refers to approaches where LLMs design hyperparameter configurations based on predicted model performance, bypassing the need for exhaustive training and evaluation runs [165]. This paradigm offers faster, more cost-efficient optimization while still supporting automation and reasoning transparency in ML workflow code generation.

Recent work demonstrates multiple ways to leverage LLMs as predictive or surrogate optimizers. AgentHPO [108] employs a two-agent framework in which a Creator agent generates candidate hyperparameters from natural language task descriptions, while an Executor agent partially simulates training and provides feedback for iterative refinement. Evaluated on 12 representative ML tasks, AgentHPO consistently outperforms random search and, in many cases, surpasses the best human-designed configurations. Beyond performance, it also offers interpretable outputs, enabling more transparent and explainable tuning processes. Similarly, AutoML-GPT [203] predicts model performance by generating synthetic training logs for given dataset and model specifications, allowing hyperparameters to be tuned through analysis of the simulated metrics without full execution.

Building on established optimization strategies, LLAMBO [110] reformulates Bayesian optimization in natural language, enabling LLMs to propose promising candidate settings based on prior evaluations and task context. Surrogate models assess these candidates to better balance exploration and exploitation. Empirical results show that LLAMBO consistently outperforms four widely used production baselines across diverse benchmarks—including proprietary and synthetic tasks—demonstrating strong practical utility and adaptability in real-world

HPO scenarios. However, LLAMBO's reliance on LLM inference introduces higher computational overhead compared to traditional BO methods. This trade-off yields improved sample efficiency, particularly valuable in black-box settings, and suggests potential for hybrid strategies—e.g., applying LLAMBO in early-stage search or integrating it as a component within lightweight BO frameworks.

5.2.3 Summary. LLMs are increasingly applied to hyperparameter optimization (HPO) in ML workflows to reduce manual tuning effort, accelerate prototyping, and enhance the traceability and explainability of design choices in automated code generation. Existing work generally follows two paradigms—execution-based and prediction-based—each offering distinct strengths and facing specific challenges, as summarized in Table 7.

Table 7. Capabilities and limitations of LLMs for hyperparameter optimization in ML workflows within the SE context.

HPO Type	Example Systems	LLM Capabilities	LLM Limitations
Execution-based	AutoMMLab [186], Agent Laboratory [144], Data Interpreter [69], SELA [21], AutoKaggle [105], AutoM^{3}L [113], Agent K v1.0 [50], GENIUS [208], GPT-NAS [191], GE [121]	• Generate hyperparameters, tuning scripts, or architectures from task descriptions • Refine configurations iteratively using real training feedback, and integrate with external search libraries • Improve auditability and traceability of tuning logic in workflow code	• High computational cost and scalability issues in large or high-dimensional search spaces • Require manual debugging for compatibility issues
Prediction-based	AgentHPO [108], AutoML-GPT [203], LLAMBO [110]	• Predict optimal hyperparameters from prior evaluations and task context without full training • Enable rapid prototyping and early-stage pipeline design • Support explainable tuning via interpretable outputs or surrogate models	• Inaccurate or biased predictions, including hallucinated metrics, can lead to poor tuning choices • Require additional validation to ensure reliability in deployment • LLM inference adds computational overhead compared to lightweight baselines

For **execution-based methods**, LLMs act as intelligent optimizers or code generators, producing configurations that are validated through actual training and refined based on observed metrics [50, 113, 186, 208]. These approaches offer strong reliability and transparent decision traces—key for SE principles such as auditability and maintainability—but they are resource-intensive and harder to scale to very large or complex search spaces.

For **prediction-based methods**, LLMs bypass full training by predicting performance from task and model descriptions [108, 110, 203]. This greatly reduces tuning time and computational demands, making it ideal for rapid prototyping. However, inaccurate or biased predictions can undermine reproducibility and result in suboptimal configurations once executed in production environments.

Overall, LLM-assisted HPO enhances traditional AutoML by combining semantic reasoning, natural language interfaces, and simulation-based prediction, effectively bridging human-intelligible task specifications with automated optimization. The main SE challenge lies in balancing efficiency, reliability, and maintainability, ensuring that generated tuning logic remains robust, verifiable, and sustainable in long-term ML workflow development.

Summary for RQ2: Existing studies show that LLMs support the mid-stage of ML workflows primarily through two dimensions: *model selection* and *hyperparameter optimization*.

(1) For model selection, two paradigms prevail. Retrieval-based methods leverage curated model repositories and metadata-enriched model cards to recommend candidate algorithms, improving traceability and reusability of decisions but depending heavily on repository coverage and quality, which limits adaptability to novel or specialized tasks. Generation-based methods synthesize selection logic, architectures, or parameter settings directly from natural language descriptions, enabling greater flexibility across modalities but risking inconsistencies, numerical errors, or component incompatibilities that require post-hoc validation. Hybrid designs are emerging, combining retrieval for reliable baselines with generative components for task-specific adaptation.

(2) For hyperparameter optimization, research follows two main approaches. Execution-based methods iteratively refine hyperparameters through real training feedback, providing empirically grounded reliability and transparent tuning traces, but at the cost of high computation and poor scalability in large search spaces. Prediction-based methods accelerate tuning by simulating performance outcomes or reformulating optimization steps, reducing resource demands but risking inaccurate or biased predictions that lead to suboptimal configurations. While execution-based HPO remains dominant, advances such as agentic feedback loops, integration with mature optimizers, and CASH-style joint decisions aim to improve efficiency, verifiability, and maintainability of tuning code in SE contexts.

From an **SE standpoint**, LLM-augmented model selection and HPO enable rapid design space exploration, accelerating prototyping and reducing manual effort in configuration tasks. However, automated choices may overlook domain-specific constraints, quality attributes, and non-functional requirements essential for robust, maintainable, and compliant software systems. To ensure sustainable adoption, practitioners should embed validation, interpretability, and governance mechanisms into these processes, aligning LLM-generated configurations with system goals and long-term engineering standards.

6 ANSWERING RQ3: HOW ARE LLMS EMPLOYED TO ENHANCE THE WORKFLOW EVALUATION STAGE IN ML WORKFLOWS?

Evaluating workflow performance is a crucial step after generating ML models, as it determines whether a constructed pipeline meets the required standards for accuracy, efficiency, and robustness. Traditionally, evaluation involves fully training the model and assessing its performance on a validation dataset. While reliable, this process is computationally expensive and time-consuming, particularly for large datasets and complex workflows [98, 202]. For example, NASNet required 500 GPUs over four days, consuming roughly 2,000 GPU hours to complete a search for the CIFAR-10 task [214]. These costs highlight the need for more efficient evaluation strategies in automated ML workflow generation. LLM-driven methods have emerged as promising solutions to reduce resource demands while maintaining acceptable accuracy. Moreover, evaluation is closely tied to hyperparameter optimization (HPO), as tuning often depends on reliable performance metrics, leading to overlaps between methods used in both stages.

Current approaches can be broadly categorized into two types: **agent-based evaluation** and **proxy-based evaluation**. These approaches aim to estimate or approximate workflow performance with reduced computation, providing faster feedback for iterative pipeline design and tuning.

6.1 Agent-based Evaluation

This category leverages LLM-powered agents to simulate or approximate workflow evaluation results without running full training cycles. These agents predict performance metrics or assess the quality of generated ML

code based on descriptive task information, intermediate outputs, or partial executions, effectively acting as an agent-as-a-judge to provide early feedback on pipeline quality.

For example, AutoML-GPT [203] generates simulated training logs that estimate metrics such as loss and accuracy based on provided hyperparameters, data cards, and model cards, offering coarse-grained predictions without costly execution. The VML framework [182] adopts a more granular approach: a learner agent, parameterized by LLM prompts, predicts output values for individual inputs, which are compared to ground-truth labels by an optimizer agent to estimate performance. Similarly, Agent Laboratory [144] employs an LLM-based reward model to score generated ML programs (0–1 scale) based on their alignment with the research plan, code quality, and observed outputs. These approaches accelerate early-stage evaluation and help filter low-quality pipelines, but they may suffer from predictive inaccuracies, hallucinations, or overconfident estimates.

6.2 Proxy-based Evaluation

Another line of research leverages surrogate models or zero-cost (ZC) proxies to deliver lightweight, computation-efficient predictions of workflow performance. These approaches exploit pre-trained knowledge or analytical heuristics to approximate key metrics such as accuracy and generalization potential, avoiding the need for full training and evaluation cycles.

For instance, Text-to-ML [184] integrates ZC proxies [119] during pipeline optimization to assess candidate ML programs using only a single forward/backward pass, significantly reducing evaluation overhead. TabPFN [67] offers another example: a pre-trained Transformer approximates Bayesian inference on synthetic datasets, enabling small tabular classification problems to be solved in seconds via a single forward pass. These methods greatly cut computational costs and enable rapid search over large design spaces. However, their reliability depends heavily on the coverage and representativeness of the pretraining data or proxy metrics.

6.3 Summary

LLMs are increasingly applied to support workflow evaluation in ML pipelines, aiming to reduce evaluation time and computational cost while maintaining the reliability of assessment. Existing studies mainly follow two paradigms—agent-based and proxy-based evaluation—each offering distinct benefits and facing specific limitations, as summarized in Table 8.

Table 8. Capabilities and limitations of LLMs for ML workflow evaluation within the SE context.

Workflow Evaluation Type	Example Systems	LLM Capabilities	LLM Limitations
Agent-based	AutoML-GPT [203], VML [182], Agent Laboratory [144]	• Simulate training or score ML programs from task descriptions, intermediate outputs, and partial runs • Provide early, low-cost feedback to filter out weak pipelines and avoid full execution	• Predictions may be inaccurate or hallucinated without ground-truth validation • Limited reproducibility and reliability across runs or domains • Bias risk if training/evaluation context is under-specified
Proxy-based	Text-to-ML [184], TabPFN [67]	• Use surrogate models or zero-cost proxies to approximate metrics with minimal computation • Enable rapid, scalable exploration of large or high-dimensional search spaces	• Depend on coverage and representativeness of prior data or heuristics • Less effective or misleading for novel or highly specialized tasks without close proxies

For **agent-based methods**, LLMs act as reasoning agents that simulate training behavior or score generated ML programs based on descriptive task information and partial results [144, 182, 203]. These methods offer

fast, coarse-to-fine feedback, enabling early elimination of weak pipeline candidates and reducing unnecessary computation. However, their reliance on generated predictions makes them vulnerable to inaccuracies and overconfident estimates, which can compromise decisions and reproducibility in the entire workflows.

For **proxy-based methods**, surrogate models or zero-cost proxies approximate performance metrics using pre-learned priors or heuristic evaluations [67, 184]. This approach significantly accelerates evaluation and scales well to large search spaces, supporting rapid iteration in ML pipeline design. Yet, its effectiveness is bounded by the representativeness of prior knowledge and the reliability of proxy signals. When pipelines deviate from known patterns or operate in novel domains, proxy-based predictions may mislead model selection and hyperparameter optimization, reducing the overall robustness of automated code generation workflows.

Summary for RQ3: Recent studies reveal a growing adoption of LLMs to accelerate and enhance the final *workflow evaluation* stage in ML workflows, aiming to reduce computational overhead while enabling faster, more informed design iterations.

Two main strategies have emerged: agent-based evaluation, where LLM-powered agents predict performance from task descriptions, intermediate outputs, or partial executions—providing early, low-cost feedback without full training; and proxy-based evaluation, which employs surrogate models or zero-cost heuristics to approximate key metrics efficiently. Both approaches improve feedback loops and help prioritize candidate workflows, but remain vulnerable to predictive inaccuracies, hallucinations, and limited generalizability in novel domains. To mitigate these risks, hybrid approaches are increasingly explored, combining LLM-based predictions with partial or real executions, alongside stronger traceability and explainability measures to enhance trust and reproducibility.

From an **SE perspective**, LLM-assisted evaluation functions as an automated testing layer that accelerates iteration cycles and supports early defect detection. However, over-reliance on opaque proxy metrics can undermine critical engineering decisions. Therefore, LLM-based evaluation should complement—rather than replace—traditional validation practices, ensuring reliability and maintainability across the software lifecycle.

7 ANSWERING RQ4: WHAT CROSS-STAGE TRENDS AND VARIATIONS CAN BE IDENTIFIED, AND WHAT ARE THEIR IMPLICATIONS FOR SOFTWARE ENGINEERING?

This section synthesizes key insights emerging from RQ1–RQ3, focusing on the recurring patterns and divergent behaviors exhibited by LLM-driven techniques across different stages of ML workflow construction. We examine how such methods enhance or complicate the workflow development process, depending on the task context and degree of automation involved. These findings are then used to derive broader implications for software engineering, encompassing both practical adaptations and theoretical extensions to existing SE methodologies.

7.1 Cross-Stage Trends and Variations

7.1.1 Common Strengths and Limitations. Table 9 summarizes the cross-stage strengths and limitations of LLM-driven techniques, providing a consolidated software engineering perspective on where such methods offer concrete benefits and where further design interventions may be required.

Across all three workflow stages—data and feature engineering, model selection and HPO, and workflow evaluation—a consistent theme is the prominent role of LLMs in improving development productivity. Despite the varying technical goals of each stage, LLM-based approaches uniformly enhance automation and provide intelligent assistance throughout the workflow. Several recurring patterns can be observed:

(1) **General-purpose task adaptation:** LLMs serve as flexible adapters, enabling the transfer of pre-trained knowledge across diverse tasks without the need for extensive re-engineering [40]. (2) **Natural language interfaces:** By allowing users to specify complex workflow logic via high-level prompts, LLMs significantly

reduce the barrier to entry for non-expert developers. (3) **Generative prototyping:** LLMs facilitate rapid exploration of the solution space by generating multiple viable workflow candidates in parallel, accelerating early-stage design [25]. (4) **Interactive refinement:** Many methods support iterative improvement through conversational or programmatic feedback loops, enabling faster convergence toward satisfactory solutions [101]. (5) **Emergent context awareness:** Some approaches demonstrate an implicit understanding of inter-stage dependencies or domain-specific heuristics, even without explicit task decomposition or supervision.

Taken together, these patterns lead to shorter development cycles, reduced manual effort, and broader accessibility for a wider range of users. They contribute to a long-standing goal: automating routine development tasks to increase productivity and improve the scalability of workflow construction [112]. However, several persistent limitations are also evident across the workflow lifecycle, raising important concerns for SE:

(1) **Limited transparency and traceability:** Many LLM-based techniques generate workflow components—such as feature transformations, model configurations, or evaluation heuristics—without exposing the underlying rationale or decision logic. This opaqueness hinders explainability, makes failure diagnosis difficult, and limits reproducibility, all of which are critical for engineering trustworthy data-driven systems [172]. (2) **Uncertain correctness and brittle robustness:** Across stages, LLMs may produce syntactically plausible yet semantically flawed artifacts—e.g., mismatched feature encodings or ill-posed model parameters—especially when workflow constraints are underspecified. Moreover, the performance of LLM-driven workflows often deteriorates under distribution shifts or noisy inputs, raising concerns about their reliability in production environments [79]. (3) **Integration and lifecycle maintenance challenges:** As workflows evolve, the non-deterministic and implicit nature of LLM generation poses challenges for systematic testing, version control, and change management. In the absence of standardized interface contracts or formal behavioral specifications, it becomes difficult to safely embed LLM-generated components into long-lived, mission-critical SE systems [104, 190].

These limitations highlight the need for architecture-aware design, robust verification pipelines, and lifecycle-conscious engineering strategies to support the responsible integration of LLMs into ML workflow automation systems.

Table 9. Cross-stage comparison of LLM-driven workflow methods from an SE perspective.

Workflow Stage	Main Benefits	Key Challenges
Data and Feature Engineering (RQ1)	• Automates low-level data tasks (e.g., schema transformation, missing value handling) • Reduces manual coding effort • Enhances early-stage prototyping speed	• Limited robustness to structural and semantic data mismatches • Poor generalization across heterogeneous task contexts • Lack of explicit error tracing mechanisms
Model Selection and HPO (RQ2)	• Enables flexible pipeline design and adaptation • Supports human-in-the-loop AutoML via promptable agents • Promotes modular reuse and recomposition of ML components	• Stochastic generation undermines reproducibility and debugging • Incomplete capture of problem constraints • Lacks principled search guidance under resource constraints
Workflow Evaluation (RQ3)	• Provides low-cost performance estimation (e.g., via surrogate models or LLM prediction) • Supports rapid feedback for iterative workflow refinement	• Unreliable approximations due to hallucination and overconfidence • Vulnerable to distribution shift and unseen task patterns • Difficult to validate or certify predictions in high-stakes applications

7.1.2 Stage-Specific Differences. While many strengths and limitations of LLM-based methods remain consistent across workflow stages, our analysis of Table 9 and the corresponding techniques reveals stage-specific variations in their behavior and effectiveness throughout the ML workflow.

First, the degree of **structural regularity** differs markedly between stages. Data preparation tasks are often schema-driven, making them well-suited to in-context learning via example-driven prompting. In contrast, model selection and configuration involve more complex dependency structures—such as hierarchical hyperparameters or conditional pipeline paths—which can challenge an LLM's ability to maintain global coherence and enforce architectural constraints [105, 165].

Second, the **difficulty of verification** increases as one moves downstream in the workflow. In earlier stages, generated artifacts like feature encodings or data transformations can be validated against known schemas or input–output consistency. However, evaluation-stage methods typically rely on predictive approximations (e.g., proxy metrics, zero-cost heuristics), which lack direct ground truth and are inherently harder to validate—particularly under domain shift or limited feedback settings [108, 203].

Third, we observe differences in the **role and granularity of user guidance**. During data and model stages, LLMs can be effectively steered through explicit prompt constraints (e.g., indicating model preferences or preprocessing logic) [186]. Evaluation, by contrast, often involves autonomous agent behaviors or surrogate models with minimal opportunity for user-specified adjustments, resulting in lower controllability and less predictable outcomes [96].

These stage-specific differences imply that SE practices for integrating LLMs should be context-sensitive. In particular, downstream stages characterized by higher uncertainty and automation may benefit from additional safeguards—such as execution checkpoints, human-in-the-loop validation, or hybrid mechanisms that blend learned surrogates with empirical evaluations—to mitigate the risks introduced by opaque or unverifiable model behaviors.

7.2 Implications for SE Practice and Theory

The cross-stage analysis of LLMs-driven ML workflow methods highlights both the opportunities and challenges associated with their integration into SE processes. These findings carry important implications for both practical engineering practices and foundational SE theories.

7.2.1 Implications for SE Practice. From a practical standpoint, LLMs promise to accelerate and enhance the construction of ML workflows by automating low-level tasks, supporting rapid prototyping, and enabling more accessible interfaces for non-experts. However, realizing these benefits in production environments requires addressing several key concerns:

- **Verification and validation mechanisms:** The opaque and probabilistic nature of LLM-generated artifacts—particularly in model selection and workflow evaluation—necessitates the development of robust verification pipelines. These may include hybrid testing strategies that combine static checks (e.g., schema conformance, constraint satisfaction) with dynamic validation (e.g., partial execution, simulation traces) to assess correctness and reliability [27, 116].
- **Managing quality attributes:** As LLM-generated components become part of SE workflows, it becomes increasingly important to consider core software quality attributes such as maintainability, traceability, and interpretability [72, 75]. To this end, developers may benefit from adopting practices such as defining interface contracts for generated modules, embedding provenance tracking, and leveraging explainability tools to support auditing and debugging across the workflow lifecycle.
- **Context-sensitive deployment strategies:** Given the stage-specific variations in LLM controllability and robustness, SE practitioners should adopt differentiated integration strategies. For instance, more autonomous

LLM behavior in evaluation stages may require runtime monitoring, fallback mechanisms, or human-in-the-loop controls [137], whereas earlier stages may benefit from tight prompt templating and user steering [136].

Collectively, these practices suggest a shift toward architecture-aware and lifecycle-conscious engineering approaches that treat LLMs not merely as tools, but as semi-autonomous collaborators whose outputs must be constrained, inspected, and governed.

7.2.2 Implications for SE Theory. Beyond tooling and deployment concerns, the integration of LLMs into ML workflow development also opens up opportunities to revisit some foundational assumptions in SE theory:

- **Revisiting development models:** Traditional SE methodologies (e.g., waterfall, agile, DevOps) [5, 141] typically assume a deterministic and developer-authored codebase. However, as ML workflows increasingly involve LLM-in-the-loop processes—such as prompt-based code generation, dynamic agent collaboration, and human–AI co-creation—there is growing motivation to explore alternative development models that better capture iterative prompting, evolving generative intents, and shared agency between human and machine actors [204].
- **Redefining lifecycle boundaries:** When LLMs are used to synthesize or modify components across multiple workflow stages, the traditional separations between design, implementation, testing, and maintenance begin to blur. This suggests the potential value of lifecycle models that emphasize tighter integration between generation, verification, and feedback—possibly in a more continuous and adaptive fashion than conventional phase-based paradigms allow [92, 125].
- **Formalizing uncertainty and feedback:** Since LLM-generated artifacts are inherently probabilistic [145], SE theory may benefit from frameworks that help characterize generation uncertainty, support specification of confidence thresholds, and enable uncertainty-aware decision-making during development. This is particularly relevant in ML workflows, where LLM-generated pipelines or evaluation strategies may vary across runs or rely on incomplete user instructions [18, 88].

Taken together, these considerations point toward the need to gradually expand SE theory to accommodate emerging patterns of generative and collaborative software development—especially in contexts where code is no longer entirely authored by humans, but shaped through interaction with powerful yet imperfect language models.

Summary for RQ4: Our cross-stage analysis reveals that LLM-driven methods consistently enhance productivity and automation throughout the ML workflow lifecycle, enabling more accessible and efficient development via general-purpose prompting, rapid prototyping, and context-aware assistance. However, these benefits are accompanied by persistent concerns around transparency, correctness, and robustness, which cut across all workflow stages. In addition, we observe stage-specific differences: early stages (e.g., data and feature engineering) offer more opportunities for user steering and verification, while later stages (e.g., evaluation) rely more on autonomous model behavior and are harder to validate.

These findings carry important implications for SE. Practically, there is a need for quality-aware and stage-sensitive engineering strategies—including modular validation, provenance tracking, and lifecycle monitoring—to ensure reliability when integrating LLMs into real-world workflows. Theoretically, our results suggest that SE models may need to evolve to accommodate the probabilistic, continuous, and collaborative nature of LLM-assisted development, motivating new lifecycle models that treat prompt refinement, uncertainty handling, and human–AI co-creation as first-class concerns in workflow engineering.

8 DISCUSSIONS

Building on the findings from RQ1 to RQ4, this section shifts focus to two forward-looking aspects. First, we identify several open challenges that need to be tackled to advance the practical use of LLMs in ML workflow development. Second, we outline key future directions for research, with an emphasis on advancing LLM-driven workflows in ways that align with long-term SE goals.

8.1 Open Challenges

8.1.1 Mitigating Potential Data Leakage in LLM-Based ML Workflows. A major challenge in integrating LLMs into ML workflows is the risk of data leakage, or data snooping. Because LLMs are pre-trained on vast amounts of publicly available text, including many common ML benchmarks, there is a significant risk of overlap with evaluation datasets. Such overlap can lead to partial or complete "memorization" of training or test data, inflating performance estimates during decision and evaluation [68, 84]. Our review identifies three workflow stages where data leakage risks are most prevalent, along with representative mitigation strategies (Table 10).

In the Data and Feature Engineering stage, one risk is that LLMs may incorporate external sources containing target-related information. This can be mitigated by vetting and filtering external sources before integration to remove any data that overlaps with the evaluation target. Another risk is that LLMs may generate features embedding implicit target signals, which can be mitigated by restricting unsafe code operations and applying automated leakage checks to detect and block target-related information in generated features [113].

In Model Selection and HPO, leakage may occur when prompts contain evaluation data cues—such as task examples or distributional statistics—leading to overfitting. This can be mitigated by removing or obfuscating such cues and validating configurations on hidden or shifted datasets, and refining prompts to emphasize fairness and diversity [18, 68].

During Workflow Evaluation, contamination may arise if benchmarks or test sets appear in the LLM's pre-training corpus. A common mitigation strategy is to use private or post-cutoff datasets to reduce data leakage and ensure a fair evaluation [201].

Table 10. Representative data leakage risks and mitigation strategies across LLM-driven ML workflow stages.

Workflow Stage	# Papers Involved	# Papers Referencing Risk	Representative Risk Type	Mitigation Strategy
Data and Feature Engineering	18	5	• External data sources contain target-related information • Generated features implicitly encode target signals	• Vet and filter external sources before integration • Restrict unsafe code operations and apply automated leakage checks
Model Selection and HPO	21	8	• Prompts include evaluation data cues	• Remove or obfuscate such cues in prompts and validate on hidden or shifted datasets
Workflow Evaluation	5	2	• Evaluation benchmarks or test sets appear in pretraining corpus	• Use private or post-cutoff datasets for evaluation

* (1) A paper is considered as referencing data leakage risk if it explicitly mentions data leakage or implicitly describes scenarios that could lead to it. (2) Each *Representative Risk Type* is sequentially mapped to its corresponding *Mitigation Strategy*.

8.1.2 Reducing Hallucination Risks in LLM-Based ML Workflows. Hallucination in LLMs—categorized as factuality hallucination (incorrect factual content) or faithfulness hallucination (deviation from user intent) [76]—poses unique risks to ML workflow construction, where generated artifacts (e.g., code, features, metrics) directly influence downstream stages. As summarized in Table 11, such risks emerge in diverse forms: from misinterpreted

requirements and spurious feature correlations during Data and Feature Engineering, to fabricated logs or non-functional architectures in Model Selection and HPO, and unsupported claims or misread metrics in Workflow Evaluation. While the specific manifestations vary by stage, a common challenge is that hallucinations, once introduced, may propagate silently through subsequent stages, making them harder to detect and rectify.

Existing mitigation strategies reveal three recurring patterns. First, **prevention through structured control**—such as ReAct-style reasoning [188]-action loops, schema-validated parsing, and constrained prompt templates—aims to reduce the likelihood of hallucination at generation time. Second, **in-process verification**—including performance-based validation, cross-model agreement, and automated unit or meta-unit tests—seeks to detect inconsistencies before outputs propagate. Third, **post-hoc correction** leverages human-in-the-loop review, empirical benchmarking, and iterative refinement to repair outputs that deviate from requirements.

Integrating these stage-specific safeguards into a unified, traceable workflow validation framework is an open challenge for LLM-based ML systems. Beyond reducing isolated hallucinations, future solutions should ensure that all workflow artifacts—whether code, configuration, or evaluation reports—are both faithful to the user's intent and verifiable against empirical evidence, enabling more robust and reproducible SE outcomes.

Table 11. Representative hallucination risks and mitigation strategies across LLM-driven ML workflow stages.

Workflow Stage	# Papers Involved	# Papers Referencing Risk	Representative Risk Type	Mitigation Strategy
Data and Feature Engineering	18	14	• Hallucinated internal logic or intermediate representations • Generation of syntactically correct but semantically invalid feature code • Misinterpretation of user requirements in feature specification • Ingestion of unverified or outdated external knowledge • Reinforcement of spurious correlations from pretrained biases	• ReAct-style reasoning with verifiable execution [188] • Structured prompts and automated feature validation pipelines • Requirement clarification dialogs prior to code synthesis • Source verification and domain knowledge vetting • Bias mitigation via fine-tuning or targeted data augmentation
Model Selection and HPO	21	20	• Generation of non-existent or invalid model artifacts • Fabrication of evaluation logs, metrics, or experimental details • Hallucinated hyperparameter suggestions such as unreasonable value ranges • Unrealistic architecture changes without empirical verification	• Schema-constrained generation, repository validation, and unit/meta-unit tests • Human-in-the-loop validation of reported results and artifacts • Validation against trusted baselines and held-out datasets • Mandatory empirical benchmarking before code integration
Workflow Evaluation	5	5	• Fabrication or misinterpretation of evaluation metrics • Unsupported performance claims beyond empirical evidence • Divergence between generated summaries and actual execution logs • Misinterpretation of automated tool outputs or evaluation criteria	• Schema-constrained parsing of logs and output artifacts • Cross-run consistency checks and majority voting mechanisms • Human review with defined discrepancy thresholds • Re-execution or cross-model validation prior to reporting

* (1) A paper is considered as referencing hallucination risk if it explicitly mentions hallucination or implicitly describes scenarios that could lead to it. (2) Each *Representative Risk Type* is sequentially mapped to its corresponding *Mitigation Strategy*.

To synthesize the findings from Table 10 and Table 11, we further compare the prevalence of data leakage and hallucination risks across different workflow stages, as shown in Figure 6.

- Across all workflow stages, the proportion of studies referencing data leakage risks remains consistently lower than that for hallucination, with values below 40%. Data leakage concerns are slightly more prominent in later stages—Model Selection and HPO (38%) and Workflow Evaluation (40%)—than in Data and Feature

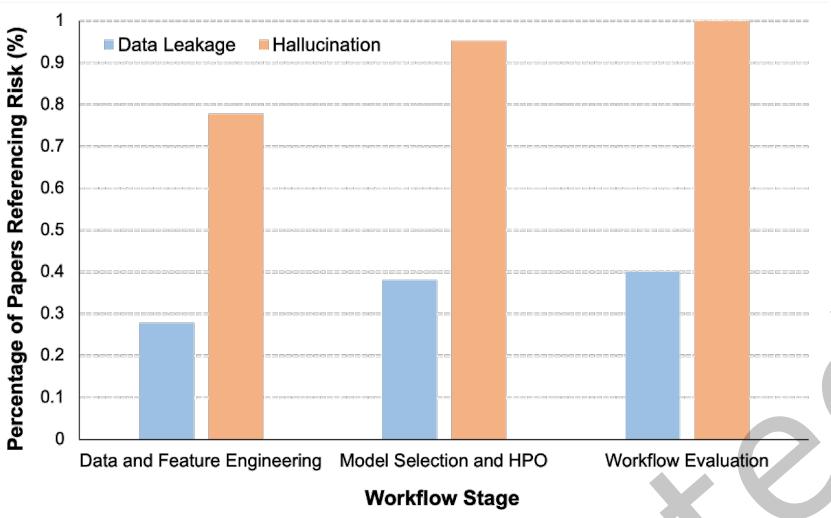


Fig. 6. Proportion of papers referencing data leakage and hallucination risks by workflow stage.

Engineering (28%), suggesting growing recognition of subtle leakage pathways introduced during model configuration and evaluation.

- Hallucination risks are referenced in a substantially higher proportion of papers, exceeding 75% in all stages and reaching 100% in Workflow Evaluation. This reflects the perception of hallucination as a pervasive and immediate threat to the reliability of LLM-driven ML workflows, particularly in stages involving automated decision-making and evaluation.
- The consistent disparity between hallucination and data leakage mentions suggests that the former is regarded as the more urgent and widespread challenge. However, the relatively lower attention to data leakage should not be interpreted as lower severity; rather, it may reflect underestimation or insufficient detection methodologies. In practice, data leakage remains equally critical, with the potential to silently compromise model validity and generalization if left unaddressed.

8.1.3 Enhancing Interaction and Explainability in LLM-Based ML Workflows. The effectiveness of LLM-driven ML workflows depends on two interrelated capabilities: (1) accurately capturing and formalizing user intent through human–LLM interaction, and (2) ensuring that the decision-making process is transparent and auditable throughout the workflow lifecycle. From an SE perspective, these capabilities align with requirements specification, design rationale capture, and traceability—all of which are critical for building maintainable, trustworthy workflow systems.

On the **interaction** side, prompt engineering serves as an ad-hoc form of requirements specification, encoding task goals, data constraints, and model configuration details for the LLM [19]. In ML workflows, this process is complicated by context length limitations, which constrain the integration of multi-stage pipeline specifications, large datasets, and fine-grained configuration parameters into a single interaction [147]. While long-context architectures [30, 82, 175] can partially mitigate this, workflow construction also demands structured encoding of heterogeneous requirements—ranging from preprocessing operators to hyperparameter search strategies—while maintaining interface compatibility between automatically generated components [68, 201]. Iterative refinement via user feedback, coupled with detailed interaction logs, supports better requirements validation and facilitates post-hoc debugging [177].

On the **explainability** side, LLM-generated workflows must provide not only domain-specific interpretability but also design rationale documentation that can be inspected, reproduced, and audited [34]. Current systems such as Text-to-ML [184] and AgentHPO [108] improve transparency through logging and workflow tracing, but over-reliance on textual summaries can obscure hidden assumptions, creating a false sense of reliability. Furthermore, the opaque reasoning processes of LLMs make it difficult to justify the selection of specific models, features, or hyperparameters—hindering maintainability and change impact analysis in production systems [184].

Addressing these challenges requires workflow-specific explainability mechanisms that integrate with established SE practices. Examples include visual analytics for pipeline architectures, causal diagrams linking data transformations to performance changes, and interactive feature attribution tools that enable design-time validation [65]. Embedding human-in-the-loop review [68, 107] at key decision points ensures that generated components remain aligned with stakeholder objectives, regulatory requirements, and maintainability constraints. By framing interaction and explainability as SE concerns—encompassing requirements capture, rationale documentation, and traceability—future LLM-driven ML workflows can achieve not only functional correctness but also long-term robustness and trustworthiness.

8.1.4 Addressing Practical Deployment Challenges of LLM-Based ML Workflows. Deploying ML workflows synthesized or configured with the assistance of LLMs in production environments presents a range of technical, organizational, and compliance-related challenges. **Scalability** remains a recurring issue: workflows that perform well in controlled experiments often fail to meet stringent latency, throughput, and fault-tolerance requirements at production scale without substantial re-engineering, performance tuning, and careful resource management [35]. **Integration with heterogeneous infrastructure** is similarly complex, as automatically generated components must interoperate with existing data pipelines, legacy systems, API endpoints, cloud platforms, and MLOps toolchains [3, 194]. This often requires non-trivial adaptation, interface refactoring, and backward compatibility measures.

Ensuring **operational robustness**—including monitoring, logging, error handling, and rollback mechanisms—can also be challenging, as automatically synthesized code may lack production-grade safeguards, increasing the risk of cascading failures [10, 114]. **Security and compliance** introduce additional constraints: workflows may inadvertently violate data protection regulations or industry-specific standards if not explicitly validated against such requirements, leading to risks such as data leakage or non-compliance with audit controls [151]. **Cross-team coordination** can further complicate deployment, as integrating generated components often requires collaboration across data science, software engineering, DevOps, and compliance teams [81].

Addressing these challenges calls for embedding **rigorous software engineering validation** (e.g., automated testing [162], static and dynamic code analysis [38]), **DevOps best practices** (e.g., CI/CD pipelines [135], infrastructure-as-code [148]), and **compliance-aware development workflows** that incorporate security scanning, legal review, and continuous audit mechanisms [13, 148]. In practice, success depends on combining automation-driven workflow synthesis with disciplined engineering processes to ensure that deployed pipelines are not only functional, but also scalable, robust, secure, and regulation-compliant.

8.1.5 Advancing Benchmarking and Evaluation for LLM-Based ML Workflows. Benchmarking is fundamental for evaluating the performance, reliability, and generalization capability of LLM-driven ML workflows. Yet, existing benchmarks vary considerably in scope, task coverage, and evaluation criteria, resulting in fragmented assessments that hinder fair comparison and consistent progress tracking.

Two primary benchmark categories have emerged. The first targets **end-to-end ML workflow automation**, covering all stages from data preprocessing to model evaluation. Examples include MLE-Bench [18], which simulates real-world ML competitions using 75 curated Kaggle challenges, and MLAgentBench [77], which evaluates LLM-powered agents in model development and evaluation. These primarily measure workflow quality via final task submission scores. The second focuses on **data science tasks** with narrower but varied scopes. Benchmarks

such as Spider2-V [16], DataSciBench [196], and DS-1000 [97] assess realistic, multi-modal data preparation, analysis, and visualization in real or simulated settings. Others, including SWE-bench [89], BigCodeBench [213], and DSBench [91], emphasize fine-grained programming skills such as code generation and local reasoning. While overlapping with certain workflow stages, they do not comprehensively cover the full ML engineering lifecycle [18] and are thus excluded from Table 12.

Despite these efforts, critical gaps remain. **First**, no widely adopted benchmark holistically covers all explicit ML workflow stages—across heterogeneous data types and deployment contexts—for both component-level and end-to-end evaluation. **Second**, most rely on task-specific leaderboards or performance metrics but lack standardized, software engineering-oriented quality measures (e.g., robustness, testability, explainability, code quality) that are essential for engineering-grade systems. **Third**, existing datasets are heavily skewed toward public platforms like Kaggle, with limited coverage of data from industrial production, scientific computing, and other real-world domains [54]—restricting the ability to evaluate system in complex, dynamic deployment environments.

Addressing these limitations will require the development of **comprehensive, unified benchmarking frameworks** that span the entire ML workflow pipeline while supporting diverse modalities, task types, and deployment scenarios. In parallel, adopting **standardized evaluation metrics grounded in SE principles**—beyond task success—will be key to assessing reliability, transparency, and maintainability. Expanding benchmarks to incorporate datasets from **industrial and scientific domains** will further ensure that evaluations reflect the challenges of real-world, large-scale deployments. Such advances will enable rigorous, reproducible, and meaningful comparisons, ultimately accelerating progress toward trustworthy and effective automation of ML workflows.

Table 12. Representative benchmarks for evaluating LLM-driven ML workflows.

Benchmark	Data Source	Modalities	Tasks	Main Evaluation Metrics
MLE-Bench [18]	75 Kaggle competitions including datasets, descriptions, and public leaderboards	Tabular, Text, Image, Video, Audio, Time-series	Classification, Regression, Time-series forecasting, Image generation, Text generation, ...	ML task performance (medal rate), Efficiency (runtime)
MLAgentBench [77]	13 datasets including canonical tasks, classic Kaggle, Kaggle challenges, recent research, and code improvement	Tabular, Text, Image, Time-series, Graph	Classification, Regression, Language modeling, Segmentation, ...	ML task performance (success rate), Efficiency (wall-clock time)
Agent K's Benchmark [50]	79 Kaggle competitions including datasets, descriptions, and leaderboards	Tabular, Text, Image	Classification, Regression, Time-series Forecasting	ML task performance (Elo-MMR ranking, medal counts)

8.2 Future Directions

8.2.1 Towards Fully Transparent End-to-End ML Workflow Generation. A compelling frontier is the development of **fully transparent, end-to-end ML workflow generation**, where LLMs autonomously orchestrate all stages—from preprocessing to evaluation—while preserving traceability and user oversight. The goal is not just to minimize human intervention, but to transform workflows into **self-explaining artifacts**: auditable, reproducible, and continuously maintainable.

Current agent-based systems such as AIDE [88], ResearchAgent [77], and CodeActAgent [173, 174] provide automation in fragments, often without explicit, interpretable structures. Future systems should instead generate workflows that are executable yet readable blueprints, combining code with structured rationales. This would elevate LLM-generated workflows from opaque automation to **living design documents**, bridging AI outputs and engineering practice.

8.2.2 Interactive and Auditable Human–LLM Workflow Co-Design. Full automation alone cannot ensure accountability; equally important are **human–LLM co-design frameworks** that treat workflows as collaborative products. Beyond prompt iteration, visual editors could allow users to reshape components while LLMs propagate changes across stages, reconciling dependencies in real time.

Future platforms should log interventions as version-controlled histories, provide explanations with traceable rationales, and embed compliance checks for regulated domains. With adaptive feedback loops, LLMs would learn from human refinements, shifting co-design from error correction to **trustworthy collaboration**—as if working alongside a skilled engineer rather than debugging a machine output.

8.2.3 Hybrid Architectures with Specialized Models and Domain-Tuned LLMs. General-purpose LLMs are versatile, but efficiency and domain alignment remain challenges. A promising path is **hybrid orchestration**, where LLMs act as architects—decomposing tasks, designing pipelines, and reasoning about constraints—while specialized ML engines (HPO optimizers, NAS, ensembling modules) serve as builders executing domain-optimized tasks.

Simultaneously, **domain-tuned LLMs** trained on curated biomedical, financial, or industrial corpora can encode context-sensitive rules and compliance constraints [93]. Advances in parameter-efficient tuning, federated adaptation, and governance-aware fine-tuning will be essential. This **division of labor between reasoning and execution** marks a step toward scalable, domain-grounded workflow intelligence.

8.2.4 Continuously Evolving ML Workflow and Component Repositories. Static workflow and component libraries cannot keep pace with the rapid evolution of ML ecosystems. A transformative alternative is to establish **dynamic, continuously evolving repositories** that ingest new algorithms, APIs, and design patterns as they emerge—keeping workflow generators perpetually up-to-date.

Such repositories could function like **"workflow app stores"**: living knowledge bases enriched via retrieval-augmented generation [44], in-context learning [31], or plugin-driven discovery. They should track version history, compatibility, and validation metadata to ensure safe reuse. This would enable LLMs to generate workflows that are not just current but **evergreen, explainable, and resilient to ecosystem drift**.

8.2.5 Advancing Self-Improving Agentic Workflows for ML Pipeline Generation. Agentic workflows—where autonomous agents collaborate through decomposition, tool use, and feedback [127, 158, 199]—offer a powerful paradigm for ML pipeline generation. Their strength lies in mimicking expert behavior: adapting to evolving constraints while optimizing long-term outcomes [142].

Future research should embed key SE principles: (1) **self-planning**, dynamically translating requirements into modular subtasks; (2) **persistent memory**, recording experiments and configurations for traceability and reuse; and (3) **self-improvement loops**, diagnosing errors and refining heuristics over time. Building such auditable, maintainable orchestration frameworks would turn agentic workflows into **self-improving collaborators**—capable of continuous optimization in complex, real-world settings.

8.2.6 Embedding ML Workflow Generation into Full-Stack Data Applications. A final horizon is embedding workflow generation within **full-stack data applications**, integrating upstream data governance with downstream decision support. Upstream, workflows should directly leverage data lakes, catalogs, and schema managers to guarantee provenance and compliance. Downstream, they should seamlessly connect to dashboards, visualization layers, and application-specific front-ends—allowing insights to flow into enterprise decision-making without friction.

Achieving this vision demands unified representations spanning data, workflow logic, and user interfaces. By plugging directly into enterprise data and software stacks [94], LLM-generated workflows could evolve from research prototypes into **production-ready infrastructure**, positioning them as first-class citizens in intelligent, adaptive software ecosystems.

Taken together, these directions chart a coherent trajectory for the future of ML workflow generation. **From vision to foundational capability**, the field is moving toward fully transparent, end-to-end generation while embracing human–LLM co-design as a necessary complement. **From architecture to resource evolution**, hybrid designs that separate reasoning from execution, coupled with continuously evolving repositories, ensure that workflow systems remain both efficient and up-to-date. **From system-level design to ecosystem-level deployment**, self-improving agentic workflows and full-stack integration extend the scope from research prototypes to production-ready infrastructures.

In sum, ML workflows are poised to evolve from **opaque automation** into systems that are **transparent**, **collaborative**, **self-improving**, and **enterprise-integrated**. Such a transformation not only advances workflow generation as a technical challenge but also repositions it as a cornerstone of intelligent, trustworthy, and adaptive software engineering practice.

9 CONCLUSION

This survey presented the first SE-oriented, stage-wise review of how LLMs are applied to construct and optimize ML workflows, covering data preprocessing and feature engineering, model selection and hyperparameter optimization, and workflow evaluation. Through a systematic taxonomy and comparative analysis, we synthesized stage-specific findings and cross-stage trends, revealing both opportunities and gaps in current approaches. Specifically, our findings point to the need for rigorous quality assurance, systematic risk mitigation, and adaptation of SE practices to the probabilistic and collaborative nature of LLM-assisted workflows. Addressing these challenges will require tighter integration of AI innovation with established engineering principles.

While LLM-driven workflow automation shows remarkable promise, the field remains in its early stages. Continued research—spanning algorithmic advances, tooling support, and empirical evaluation—will be essential for ensuring the reliability, transparency, and scalability of such systems. We hope this work provides a foundation for researchers and practitioners, fostering closer collaboration between the SE and ML communities to shape the next generation of intelligent and trustworthy ML workflows.

ACKNOWLEDGMENTS

This work is supported by the Interdisciplinary Program of Shanghai Jiao Tong University (project number YG2024QNB05). This work is also supported by the Program of Technology Innovation of the Science and Technology Commission of Shanghai Municipality (Granted No. 21511104700).

REFERENCES

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[2] Bo Adler, Niket Agarwal, Ashwath Aithal, Dong H Anh, Pallab Bhattacharya, Annika Brundyn, Jared Casper, Bryan Catanzaro, Sharon Clay, Jonathan Cohen, et al. 2024. Nemotron-4 340B Technical Report. *arXiv preprint arXiv:2406.11704* (2024).

[3] Md Abdullah Al Alamin and Gias Uddin. 2024. How far are we with automated machine learning? characterization and challenges of AutoML toolkits. *Empirical Software Engineering* 29, 4 (2024), 91.

[4] Greg Allen. 2020. Understanding AI technology. *Joint Artificial Intelligence Center (JAIC) The Pentagon United States* 2, 1 (2020), 24–32.

[5] Fernando Almeida, Jorge Simões, and Sérgio Lopes. 2022. Exploring the benefits of combining devops and agile. *Future Internet* 14, 2 (2022), 63.

[6] Sohyun An, Hayeon Lee, Jaehyeong Jo, Seanie Lee, and Sung Ju Hwang. 2023. DiffusionNAG: Predictor-guided Neural Architecture Generation with Diffusion Models. In *The Twelfth International Conference on Learning Representations*.

[7] Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. 2023. Palm 2 technical report. *arXiv preprint arXiv:2305.10403* (2023).

[8] Gabriel O Assunção, Rafael Izbicki, and Marcos O Prates. 2024. Is Augmentation Effective in Improving Prediction in Imbalanced Datasets? *Journal of Data Science* (2024), 1–16.

[9] Guangji Bai, Zheng Chai, Chen Ling, Shiyu Wang, Jiaying Lu, Nan Zhang, Tingwei Shi, Ziyang Yu, Mengdan Zhu, Yifei Zhang, et al. 2024. Beyond efficiency: A systematic survey of resource-efficient large language models. *arXiv preprint arXiv:2401.00625* (2024).

[10] Firas Bayram and Bestoun S Ahmed. 2025. Towards trustworthy machine learning in production: An overview of the robustness in mlops approach. *Comput. Surveys* 57, 5 (2025), 1–35.

[11] Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence* 35, 8 (2013), 1798–1828.

[12] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of machine learning research* 13, 2 (2012).

[13] Samuel Bernardo, Pablo Orviz, Mario David, Jorge Gomes, David Arce, Diana Naranjo, Ignacio Blanquer, Isabel Campos, Germán Moltó, and Joao Pina. 2024. Software Quality Assurance as a Service: Encompassing the quality assessment of software and services. *Future Generation Computer Systems* 156 (2024), 254–268.

[14] Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 17682–17690.

[15] Fabio Calefato, Luigi Quaranta, Filippo Lanubile, and Marcos Kalinowski. 2023. Assessing the use of automl for data-driven software engineering. In *2023 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 1–12.

[16] Ruisheng Cao, Fangyu Lei, Haoyuan Wu, Jixuan Chen, Yeqiao Fu, Hongcheng Gao, Xinzhuang Xiong, Hanchong Zhang, Yuchen Mao, Wenjing Hu, et al. [n. d.]. How far are multimodal agents from automating data science and engineering workflows?, 2024. *URL https://arxiv. org/abs/2407.10956* ([n. d.]).

[17] Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q Feldman, et al. 2023. MultiPL-E: a scalable and polyglot approach to benchmarking neural code generation. *IEEE Transactions on Software Engineering* 49, 7 (2023), 3675–3691.

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

[19] Banghao Chen, Zhaofeng Zhang, Nicolas Langrené, and Shengxin Zhu. 2023. Unleashing the potential of prompt engineering in Large Language Models: a comprehensive review. *arXiv preprint arXiv:2310.14735* (2023).

[20] Huaming Chen and M Ali Babar. 2024. Security for Machine Learning-based Software Systems: A Survey of Threats, Practices, and Challenges. *Comput. Surveys* 56, 6 (2024), 1–38.

[21] Yizhou Chi, Yizhang Lin, Sirui Hong, Duyi Pan, Yaying Fei, Guanghao Mei, Bangbang Liu, Tianqi Pang, Jacky Kwok, Ceyao Zhang, et al. 2024. SELA: Tree-Search Enhanced LLM Agents for Automated Machine Learning. *arXiv preprint arXiv:2410.17238* (2024).

[22] Hyunjun Choi, Jay Moran, Nicholas Matsumoto, Miguel E Hernandez, and Jason H Moore. 2023. Aliro: an automated machine learning tool leveraging large language models. *Bioinformatics* 39, 10 (2023), btad606.

[23] Kristy Choi, Chris Cundy, Sanjari Srivastava, and Stefano Ermon. 2022. LMPriors: Pre-Trained Language Models as Task-Specific Priors. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*.

[24] Yeounoh Chung, Peter J Haas, Eli Upfal, and Tim Kraska. 2018. Unknown examples & machine learning model generalization. *arXiv preprint arXiv:1808.08294* (2018).

[25] Claudionor N Coelho Jr, Hanchen Xiong, Tushar Karayil, Sree Koratala, Rex Shang, Jacob Bollinger, Mohamed Shabar, and Syam Nair. 2024. Effort and Size Estimation in Software Projects with Large Language Model-based Intelligent Interfaces. *arXiv preprint arXiv:2402.07158* (2024).

[26] Alexander I Cowen-Rivers, Wenlong Lyu, Rasul Tutunov, Zhi Wang, Antoine Grosnit, Ryan Rhys Griffiths, Alexandre Max Maraval, Hao Jianye, Jun Wang, Jan Peters, et al. 2022. Hebo: Pushing the limits of sample-efficient hyper-parameter optimisation. *Journal of Artificial Intelligence Research* 74 (2022), 1269–1349.

[27] Marcos Cramer and Lucian McIntyre. 2025. Verifying LLM-Generated Code in the Context of Software Verification with Ada/SPARK. *arXiv preprint arXiv:2502.07728* (2025).

[28] Tijl De Bie, Luc De Raedt, José Hernández-Orallo, Holger H Hoos, Padhraic Smyth, and Christopher KI Williams. 2022. Automating data science. *Commun. ACM* 65, 3 (2022), 76–87.

[29] DeepSeek. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. https://api.semanticscholar.org/CorpusID:275789950

[30] Yiran Ding, Li Lyna Zhang, Chengruidong Zhang, Yuanyuan Xu, Ning Shang, Jiahang Xu, Fan Yang, and Mao Yang. 2024. Longrope: Extending llm context window beyond 2 million tokens. *arXiv preprint arXiv:2402.13753* (2024).

[31] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Tianyu Liu, et al. 2022. A survey on in-context learning. *arXiv preprint arXiv:2301.00234* (2022).

[32] Yihong Dong, Ge Li, and Zhi Jin. 2023. CODEP: grammatical seq2seq model for general-purpose code generation. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 188–198.

[33] José Cassio dos Santos Junior, Rachel Hu, Richard Song, and Yunfei Bai. 2024. Domain-driven LLM development: insights into rag and fine-tuning practices. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 6416–6417.
[34] Jaimie Drozdal, Justin Weisz, Dakuo Wang, Gaurav Dass, Bingsheng Yao, Changruo Zhao, Michael Muller, Lin Ju, and Hui Su. 2020. Trust in AutoML: exploring information needs for establishing trust in automated machine learning systems. In *Proceedings of the 25th international conference on intelligent user interfaces*. 297–307.
[35] Vijayan Naveen Edapurath. 2024. Building Scalable MLOps: Optimizing Machine Learning Deployment and Operations. (2024).
[36] Katharina Eggensperger, Philipp Müller, Neeratyoy Mallik, Matthias Feurer, René Sass, Aaron Klein, Noor Awad, Marius Lindauer, and Frank Hutter. 2021. HPOBench: A collection of reproducible multi-fidelity benchmark problems for HPO. *arXiv preprint arXiv:2109.06716* (2021).
[37] Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. 2020. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv preprint arXiv:2003.06505* (2020).
[38] N Valentine Eziamaka, T Narku Odonkor, and A Adewale Akinsulire. 2024. Advanced strategies for achieving comprehensive code quality and ensuring software reliability. *Computer Science & IT Research Journal* 5, 8 (2024), 1751–1779.
[39] Sarah Fakhoury, Aaditya Naik, Georgios Sakkas, Saikat Chakraborty, and Shuvendu K Lahiri. 2024. Llm-based test-driven interactive code generation: User study and empirical evaluation. *IEEE Transactions on Software Engineering* (2024).
[40] Shengda Fan, Xin Cong, Yuepeng Fu, Zhong Zhang, Shuyan Zhang, Yuanwei Liu, Yesai Wu, Yankai Lin, Zhiyuan Liu, and Maosong Sun. [n. d.]. WorkflowLLM: Enhancing Workflow Orchestration Capability of Large Language Models. In *The Thirteenth International Conference on Learning Representations*.
[41] Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. 2024. A Survey on RAG Meeting LLMs: Towards Retrieval-Augmented Large Language Models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 6491–6501.
[42] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. 2022. Auto-sklearn 2.0: Hands-free automl via meta-learning. *Journal of Machine Learning Research* 23, 261 (2022), 1–61.
[43] Isabel O Gallegos, Ryan A Rossi, Joe Barrow, Md Mehrab Tanjim, Sungchul Kim, Franck Dernoncourt, Tong Yu, Ruiyi Zhang, and Nesreen K Ahmed. 2024. Bias and fairness in large language models: A survey. *Computational Linguistics* (2024), 1–79.
[44] Aoran Gan, Hao Yu, Kai Zhang, Qi Liu, Wenyu Yan, Zhenya Huang, Shiwei Tong, and Guoping Hu. 2025. Retrieval Augmented Generation Evaluation in the Era of Large Language Models: A Comprehensive Survey. *arXiv preprint arXiv:2504.14891* (2025).
[45] Jiahui Gao, Renjie Pi, Lin Yong, Hang Xu, Jiacheng Ye, Zhiyong Wu, Weizhong Zhang, Xiaodan Liang, Zhenguo Li, and Lingpeng Kong. 2023. Self-Guided Noise-Free Data Generation for Efficient Zero-Shot Learning. In *International Conference on Learning Representations (ICLR 2023)*.
[46] Jianqi Gao, Hao Wu, Yiu-ming Cheung, Jian Cao, Hang Yu, and Yonggang Zhang. 2025. Mitigating Forgetting in Adapting Pre-trained Language Models to Text Processing Tasks via Consistency Alignment. In *Proceedings of the ACM on Web Conference 2025*. 3492–3504.
[47] Zijie Geng, Xijun Li, Jie Wang, Xiao Li, Yongdong Zhang, and Feng Wu. 2023. A deep instance generative framework for milp solvers under limited data availability. *Advances in Neural Information Processing Systems* 36 (2023), 26025–26047.
[48] Pieter Gijsbers, Marcos LP Bueno, Stefan Coors, Erin LeDell, Sébastien Poirier, Janek Thomas, Bernd Bischl, and Joaquin Vanschoren. 2024. Amlb: an automl benchmark. *Journal of Machine Learning Research* 25, 101 (2024), 1–65.
[49] Görkem Giray. 2021. A software engineering perspective on engineering machine learning systems: State of the art and challenges. *Journal of Systems and Software* 180 (2021), 111031.
[50] Antoine Grosnit, Alexandre Maraval, James Doran, Giuseppe Paolo, Albert Thomas, Refinath Shahul Hameed Nabeezath Beevi, Jonas Gonzalez, Khyati Khandelwal, Ignacio Iacobacci, Abdelhakim Benechehab, et al. 2024. Large language models orchestrating structured reasoning achieve kaggle grandmaster level. *arXiv preprint arXiv:2411.03562* (2024).
[51] Xiaodong Gu, Meng Chen, Yalan Lin, Yuhan Hu, Hongyu Zhang, Chengcheng Wan, Zhao Wei, Yong Xu, and Juhong Wang. 2025. On the effectiveness of large language models in domain-specific code generation. *ACM Transactions on Software Engineering and Methodology* 34, 3 (2025), 1–22.
[52] Xiaodong Gu, Hongyu Zhang, Dongmei Zhang, and Sunghun Kim. 2017. DeepAM: Migrate APIs with multi-modal sequence to sequence learning. *arXiv preprint arXiv:1704.07734* (2017).
[53] Yang Gu, Jian Cao, Yuan Guo, Shiyou Qian, and Wei Guan. 2023. Plan, Generate and Match: Scientific Workflow Recommendation with Large Language Models. In *International Conference on Service-Oriented Computing*. Springer, 86–102.
[54] Yang Gu, Jian Cao, Shiyou Qian, and Wei Guan. 2023. SWORTS: a scientific workflow retrieval approach by learning textual and structural semantics. *IEEE Transactions on Services Computing* 16, 6 (2023), 4205–4219.
[55] Yang Gu, Jian Cao, Shiyou Qian, Nengjun Zhu, and Wei Guan. 2024. ProSwats: A Proxy-based Scientific Workflow Retrieval Approach by Bridging the Gap between Textual and Structural Semantics. In *2024 IEEE International Conference on Web Services (ICWS)*. IEEE, 932–943.
[56] Yang Gu, Jian Cao, Hengyu You, Nengjun Zhu, and Shiyou Qian. 2025. ADELA: Accelerating Evolutionary Design of Machine Learning Pipelines with the Accompanying Surrogate Model. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 39.

16915–16923.
[57] Shubha Guha, Falaah Arif Khan, Julia Stoyanovich, and Sebastian Schelter. 2024. Automated data cleaning can hurt fairness in machine learning-based decision making. *IEEE Transactions on Knowledge and Data Engineering* (2024).
[58] Siyuan Guo, Cheng Deng, Ying Wen, Hechang Chen, Yi Chang, and Jun Wang. 2024. DS-Agent: Automated Data Science by Empowering Large Language Models with Case-Based Reasoning. *arXiv preprint arXiv:2402.17453* (2024).
[59] David Harrison Jr and Daniel L Rubinfeld. 1978. Hedonic housing prices and the demand for clean air. *Journal of environmental economics and management* 5, 1 (1978), 81–102.
[60] Md Mahadi Hassan, Alex Knipper, and Shubhra Kanti Karmaker Santu. 2023. Chatgpt as your personal data scientist. *arXiv preprint arXiv:2305.13657* (2023).
[61] Junda He, Christoph Treude, and David Lo. 2024. LLM-Based Multi-Agent Systems for Software Engineering: Literature Review, Vision and the Road Ahead. *ACM Transactions on Software Engineering and Methodology* (2024).
[62] Xin He, Kaiyong Zhao, and Xiaowen Chu. 2021. AutoML: A survey of the state-of-the-art. *Knowledge-based systems* 212 (2021), 106622.
[63] Geoffrey E Hinton and Ruslan R Salakhutdinov. 2006. Reducing the dimensionality of data with neural networks. *science* 313, 5786 (2006), 504–507.
[64] S Hochreiter. 1997. Long Short-term Memory. *Neural Computation MIT-Press* (1997).
[65] Robert R Hoffman, Shane T Mueller, Gary Klein, and Jordan Litman. 2018. Metrics for explainable AI: Challenges and prospects. *arXiv preprint arXiv:1812.04608* (2018).
[66] Markus Hofmann and Ralf Klinkenberg. 2016. *RapidMiner: Data mining use cases and business analytics applications*. CRC Press.
[67] Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. 2022. Tabpfn: A transformer that solves small tabular classification problems in a second. *arXiv preprint arXiv:2207.01848* (2022).
[68] Noah Hollmann, Samuel Müller, and Frank Hutter. 2024. Large language models for automated data science: Introducing caafe for context-aware automated feature engineering. *Advances in Neural Information Processing Systems* 36 (2024).
[69] Sirui Hong, Yizhang Lin, Bangbang Liu, Binhao Wu, Danyang Li, Jiaqi Chen, Jiayi Zhang, Jinlin Wang, Lingyao Zhang, Mingchen Zhuge, et al. 2024. Data interpreter: An LLM agent for data science. *arXiv preprint arXiv:2402.18679* (2024).
[70] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology* 33, 8 (2024), 1–79.
[71] Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. 2025. Model context protocol (mcp): Landscape, security threats, and future research directions. *arXiv preprint arXiv:2503.23278* (2025).
[72] Chao Hu, Yitian Chai, Hao Zhou, Fandong Meng, Jie Zhou, and Xiaodong Gu. 2024. How Effectively Do Code Language Models Understand Poor-Readability Code?. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. 795–806.
[73] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685* (2021).
[74] Wenbo Hu, Yifan Xu, Yi Li, Weiyue Li, Zeyuan Chen, and Zhuowen Tu. 2024. Bliva: A simple multimodal llm for better handling of text-rich visual questions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 2256–2264.
[75] Xing Hu, Feifei Niu, Junkai Chen, Xin Zhou, Junwei Zhang, Junda He, Xin Xia, and David Lo. 2025. Assessing and Advancing Benchmarks for Evaluating Large Language Models in Software Engineering Tasks. *arXiv preprint arXiv:2505.08903* (2025).
[76] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. 2023. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *arXiv preprint arXiv:2311.05232* (2023).
[77] Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. 2024. MLAgentBench: Evaluating Language Agents on Machine Learning Experimentation. In *Forty-first International Conference on Machine Learning*.
[78] Ziyi Huang, Yang Li, Dushuai Li, Yao Mu, Hongmao Qin, and Nan Zheng. 2025. Post-interactive Multimodal Trajectory Prediction for Autonomous Driving. *arXiv preprint arXiv:2503.09366* (2025).
[79] Sinclair Hudson, Sophia Jit, Boyue Caroline Hu, and Marsha Chechik. 2024. A software engineering perspective on testing large language models: Research, practice, tools and benchmarks. *arXiv preprint arXiv:2406.08216* (2024).
[80] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. 2019. *Automated machine learning: methods, systems, challenges*. Springer Nature.
[81] Samuel Idowu, Osman Osman, Daniel Strüber, and Thorsten Berger. 2024. Machine learning experiment management tools: a mixed-methods empirical study. *Empirical Software Engineering* 29, 4 (2024), 74.
[82] Raisa Islam and Owana Marzia Moushi. 2024. GPT-4o: The Cutting-Edge Advancement in Multimodal LLM. *Authorea Preprints* (2024).
[83] Ganesh Jawahar, Muhammad Abdul-Mageed, Laks VS Lakshmanan, and Dujian Ding. 2023. LLM Performance Predictors are good initializers for Architecture Search. *arXiv preprint arXiv:2310.16712* (2023).
[84] Daniel P Jeong, Zachary C Lipton, and Pradeep Ravikumar. 2024. LLM-Select: Feature Selection with Large Language Models. *arXiv preprint arXiv:2407.02694* (2024).

[85] MZH Jesmeen, J Hossen, S Sayeed, CK Ho, K Tawsif, Armanur Rahman, and E Arif. 2018. A survey on cleaning dirty data using machine learning paradigm for big data analytics. *Indones. J. Electr. Eng. Comput. Sci* 10, 3 (2018), 1234–1243.

[86] Ling Jiang, Junwen An, Huihui Huang, Qiyi Tang, Sen Nie, Shi Wu, and Yuqun Zhang. 2024. Binaryai: binary software composition analysis via intelligent binary source code matching. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13.

[87] Xue Jiang, Yihong Dong, Lecheng Wang, Zheng Fang, Qiwei Shang, Ge Li, Zhi Jin, and Wenpin Jiao. 2024. Self-planning code generation with large language models. *ACM Transactions on Software Engineering and Methodology* 33, 7 (2024), 1–30.

[88] Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. 2025. AIDE: AI-Driven Exploration in the Space of Code. *arXiv preprint arXiv:2502.13138* (2025).

[89] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770* (2023).

[90] Haifeng Jin, Qingquan Song, and Xia Hu. 2019. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 1946–1956.

[91] Liqiang Jing, Zhehui Huang, Xiaoyang Wang, Wenlin Yao, Wenhao Yu, Kaixin Ma, Hongming Zhang, Xinya Du, and Dong Yu. 2024. DSBench: How Far Are Data Science Agents from Becoming Data Science Experts? *arXiv preprint arXiv:2409.07703* (2024).

[92] Styliani I Kampezidou, Archana Tikayat Ray, Anirudh Prabhakara Bhat, Olivia J Pinon Fischer, and Dimitri N Mavris. 2024. Fundamental components and principles of supervised machine learning workflows with numerical and categorical data. *Eng* 5, 1 (2024), 384–416.

[93] Noriko Kanemaru, Koichiro Yasaka, Naomasa Okimoto, Mai Sato, Takuto Nomura, Yuichi Morita, Akira Katayama, Shigeru Kiryu, and Osamu Abe. 2025. Efficacy of Fine-Tuned Large Language Model in CT Protocol Assignment as Clinical Decision-Supporting System. *Journal of Imaging Informatics in Medicine* (2025), 1–13.

[94] Rafiq Ahmad Khan, Siffat Ullah Khan, Muhammad Azeem Akbar, and Musaad Alzahrani. 2024. Security risks of global software development life cycle: Industry practitioner's perspective. *Journal of Software: Evolution and Process* 36, 3 (2024), e2521.

[95] Dong Jae Kim and Tse-Hsun Chen. 2024. Exploring the Impact of Inheritance on Test Code Maintainability. In *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*. 382–383.

[96] Martin Klissarov, R Devon Hjelm, Alexander T Toshev, and Bogdan Mazoure. [n. d.]. On the Modeling Capabilities of Large Language Models for Sequential Decision Making. In *The Thirteenth International Conference on Learning Representations*.

[97] Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. 2023. DS-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*. PMLR, 18319–18345.

[98] Teddy Lazebnik, Amit Somech, and Abraham Itzhak Weinberg. 2022. Substrat: A subset-based optimization strategy for faster automl. *Proceedings of the VLDB Endowment* 16, 4 (2022), 772–780.

[99] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436–444.

[100] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[101] Chao Lei, Yanchuan Chang, Nir Lipovetzky, and Krista A Ehinger. 2024. Planning-driven programming: A large language model programming workflow. *arXiv preprint arXiv:2411.14503* (2024).

[102] Florian Leiser, Sven Eckhardt, Valentin Leuthe, Merlin Knaeble, Alexander Maedche, Gerhard Schwabe, and Ali Sunyaev. 2024. Hill: A hallucination identifier for large language models. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. 1–13.

[103] Junyi Li, Jie Chen, Ruiyang Ren, Xiaoxue Cheng, Wayne Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. 2024. The dawn after the dark: An empirical study on factuality hallucination in large language models. *arXiv preprint arXiv:2401.03205* (2024).

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

[105] Ziming Li, Qianbo Zang, David Ma, Jiawei Guo, Tuney Zheng, Minghao Liu, Xinyao Niu, Yue Wang, Jian Yang, Jiaheng Liu, et al. 2024. Autokaggle: A multi-agent framework for autonomous data science competitions. *arXiv preprint arXiv:2410.20424* (2024).

[106] Marius Lindauer, Holger H Hoos, Frank Hutter, and Torsten Schaub. 2015. Autofolio: Algorithm configuration for algorithm selection. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*.

[107] Marius Lindauer, Florian Karl, Anne Klier, Julia Moosbauer, Alexander Tornede, Andreas C Mueller, Frank Hutter, Matthias Feurer, and Bernd Bischl. 2024. Position: A Call to Action for a Human-Centered AutoML Paradigm. In *Proceedings of the 41st International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 235)*, Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (Eds.). PMLR, 30566–30584. https://proceedings.mlr.press/v235/lindauer24a.html

[108] Siyi Liu, Chen Gao, and Yong Li. 2024. Large Language Model Agent for Hyper-Parameter Optimization. *arXiv preprint arXiv:2402.01881* (2024).

[109] Shang-Ching Liu, ShengKun Wang, Tsungyao Chang, Wenqi Lin, Chung-Wei Hsiung, Yi-Chen Hsieh, Yu-Ping Cheng, Sian-Hong Luo, and Jianwei Zhang. 2023. JarviX: A LLM no code platform for tabular data analysis and optimization. In *Proceedings of the 2023*

Conference on Empirical Methods in Natural Language Processing: Industry Track. 622–630.
[110] Tennison Liu, Nicolás Astorga, Nabeel Seedat, and Mihaela van der Schaar. 2024. Large Language Models to Enhance Bayesian Optimization. arXiv:2402.03921 [cs.LG] https://arxiv.org/abs/2402.03921
[111] Zhijie Liu, Yutian Tang, Xiapu Luo, Yuming Zhou, and Liang Feng Zhang. 2024. No need to lift a finger anymore? assessing the quality of code generation by chatgpt. *IEEE Transactions on Software Engineering* (2024).
[112] Andrea Lops, Fedelucio Narducci, Azzurra Ragone, Michelantonio Trizio, and Claudio Bartolini. 2025. A system for automated unit test generation using large language models and assessment of generated test suites. In *2025 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 29–36.
[113] Daqin Luo, Chengjian Feng, Yuxuan Nong, and Yiqing Shen. 2024. AutoM3L: An Automated Multimodal Machine Learning Framework with Large Language Models. *arXiv preprint arXiv:2408.00665* (2024).
[114] Yumo Luo, Mikko Raatikainen, and Jukka K Nurminen. 2023. Autonomously adaptive machine learning systems: Experimentation-driven open-source pipeline. In *2023 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 44–52.
[115] Yongle Luo, Rongsheng Wang, Peter Gam, Jiaxi Cui, circlestarzero, Shiwen Ni, Jaseon Quanta, Qingxu Fu, and Siyuan Hou. 2023. ChatPaper: Use LLM to summarize papers. https://github.com/kaixindelele/ChatPaper.
[116] Zihan Ma, Taolin Zhang, Maosong Cao, Wenwei Zhang, Minnan Luo, Songyang Zhang, and Kai Chen. 2025. Rethinking Verification for LLM Code Generation: From Generation to Testing. *arXiv preprint arXiv:2507.06920* (2025).
[117] Andrzej Maćkiewicz and Waldemar Ratajczak. 1993. Principal components analysis (PCA). *Computers & Geosciences* 19, 3 (1993), 303–342.
[118] Bilal Mehboob, Chun Yong Chong, Sai Peck Lee, and Joanne Mun Yee Lim. 2021. Reusability affecting factors and software metrics for reusability: A systematic literature review. *Software: Practice and Experience* 51, 6 (2021), 1416–1458.
[119] Joe Mellor, Jack Turner, Amos Storkey, and Elliot J Crowley. 2021. Neural architecture search without training. In *International conference on machine learning*. PMLR, 7588–7598.
[120] Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. 2024. Large language models: A survey. *arXiv preprint arXiv:2402.06196* (2024).
[121] Clint Morris, Michael Jurado, and Jason Zutty. 2024. Llm guided evolution-the automation of models advancing models. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 377–384.
[122] Rohan Mukherjee, Yeming Wen, Dipak Chaudhari, Thomas Reps, Swarat Chaudhuri, and Christopher Jermaine. 2021. Neural program generation modulo static analysis. *Advances in Neural Information Processing Systems* 34 (2021), 18984–18996.
[123] Alhassan Mumuni and Fuseini Mumuni. 2024. Automated data processing and feature engineering for deep learning and big data applications: a survey. *Journal of Information and Intelligence* (2024).
[124] Felix Neutatz, Binger Chen, Yazan Alkhatib, Jingwen Ye, and Ziawasch Abedjan. 2022. Data Cleaning and AutoML: Would an optimizer choose to clean? *Datenbank-Spektrum* 22, 2 (2022), 121–130.
[125] Anh Nguyen-Duc, Pekka Abrahamsson, and Foutse Khomh. 2024. *Generative AI for effective software development*. Springer.
[126] Nikolay O Nikitin, Pavel Vychuzhanin, Mikhail Sarafanov, Iana S Polonskaia, Ilia Revin, Irina V Barabanova, Gleb Maximov, Anna V Kalyuzhnaya, and Alexander Boukhanovsky. 2022. Automated evolutionary approach for the design of composite machine learning pipelines. *Future Generation Computer Systems* 127 (2022), 109–125.
[127] Boye Niu, Yiliao Song, Kai Lian, Yifan Shen, Yu Yao, Kun Zhang, and Tongliang Liu. [n. d.]. Flow: Modularized Agentic Workflow Automation. In *The Thirteenth International Conference on Learning Representations*.
[128] Bentley James Oakes, Michalis Famelis, and Houari Sahraoui. 2024. Building Domain-Specific Machine Learning Workflows: A Conceptual Framework for the State of the Practice. *ACM Transactions on Software Engineering and Methodology* 33, 4 (2024), 1–50.
[129] Randal S Olson and Jason H Moore. 2016. TPOT: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on automatic machine learning*. PMLR, 66–74.
[130] OpenAI. 2024. Learning to Reason with LLMs. https://openai.com/index/learning-to-reason-with-llms.
[131] Francisco Villarroel Ordenes and Rosaria Silipo. 2021. Machine learning for marketing on the KNIME Hub: The development of a live repository for marketing applications. *Journal of Business Research* 137 (2021), 393–410.
[132] Yixin Ou, Yujie Luo, Jingsheng Zheng, Lanning Wei, Shuofei Qiao, Jintian Zhang, Da Zheng, Huajun Chen, and Ningyu Zhang. 2025. AutoMind: Adaptive Knowledgeable Agent for Automated Data Science. *arXiv preprint arXiv:2506.10974* (2025).
[133] Ekrem Öztürk, Fabio Ferreira, Hadi Jomaa, Lars Schmidt-Thieme, Josif Grabocka, and Frank Hutter. 2022. Zero-shot AutoML with pretrained models. In *International Conference on Machine Learning*. PMLR, 17138–17155.
[134] Anubha Parashar, Apoorva Parashar, Weiping Ding, Mohammad Shabaz, and Imad Rida. 2023. Data preprocessing and feature selection techniques in gait recognition: A comparative study of machine learning and deep learning approaches. *Pattern Recognition Letters* 172 (2023), 65–73.
[135] Akshit Raj Patel and Sulabh Tyagi. 2025. Enhancing software quality of CI/CD pipeline through continuous testing: a DevOps-driven maintainable hybrid automation testing framework. *International Journal of Information Technology* (2025), 1–15.

[136] Chanathip Pornprasit and Chakkrit Tantithamthavorn. 2024. Fine-tuning and prompt engineering for large language models-based code review automation. *Information and Software Technology* 175 (2024), 107523.
[137] Jianing Qiu, Kyle Lam, Guohao Li, Amish Acharya, Tien Yin Wong, Ara Darzi, Wu Yuan, and Eric J Topol. 2024. LLM-based agentic systems in medicine and healthcare. *Nature Machine Intelligence* 6, 12 (2024), 1418–1420.
[138] J. Ross Quinlan. 1986. Induction of decision trees. *Machine learning* 1 (1986), 81–106.
[139] Maxim Rabinovich, Mitchell Stern, and Dan Klein. 2017. Abstract syntax networks for code generation and semantic parsing. *arXiv preprint arXiv:1704.07535* (2017).
[140] Irina Rish et al. 2001. An empirical study of the naive Bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, Vol. 3. Citeseer, 41–46.
[141] Nayan B Ruparelia. 2010. Software development lifecycle models. *ACM SIGSOFT Software Engineering Notes* 35, 3 (2010), 8–13.
[142] Ripon K Saha, Akira Ura, Sonal Mahajan, Chenguang Zhu, Linyi Li, Yang Hu, Hiroaki Yoshida, Sarfraz Khurshid, and Mukul R Prasad. 2022. SapientML: synthesizing machine learning pipelines by learning from human-writen solutions. In *Proceedings of the 44th International Conference on Software Engineering*. 1932–1944.
[143] Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. 2024. A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv preprint arXiv:2402.07927* (2024).
[144] Samuel Schmidgall, Yusheng Su, Ze Wang, Ximeng Sun, Jialian Wu, Xiaodong Yu, Jiang Liu, Zicheng Liu, and Emad Barsoum. 2025. Agent laboratory: Using llm agents as research assistants. *arXiv preprint arXiv:2501.04227* (2025).
[145] Johannes Schneider. 2024. Explainable Generative AI (GenXAI): a survey, conceptualization, and research agenda. *Artificial Intelligence Review* 57, 11 (2024), 289.
[146] Vraj Shah, Jonathan Lacanlale, Premanand Kumar, Kevin Yang, and Arun Kumar. 2021. Towards benchmarking feature type inference for automl platforms. In *Proceedings of the 2021 international conference on management of data*. 1584–1596.
[147] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2024. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems* 36 (2024).
[148] Ryo Shimizu, Yuna Nunomura, and Hideyuki Kanuka. 2024. Test-suite-guided discovery of least privilege for cloud infrastructure as code. *Automated Software Engineering* 31, 1 (2024), 25.
[149] Jiho Shin, Moshi Wei, Junjie Wang, Lin Shi, and Song Wang. 2023. The good, the bad, and the missing: Neural code generation for machine learning tasks. *ACM Transactions on Software Engineering and Methodology* 33, 2 (2023), 1–24.
[150] Connor Shorten and Taghi M Khoshgoftaar. 2019. A survey on image data augmentation for deep learning. *Journal of big data* 6, 1 (2019), 1–48.
[151] Sudhi Sinha and Young M Lee. 2024. Challenges with developing and deploying AI models and applications in industrial systems. *Discover Artificial Intelligence* 4, 1 (2024), 55.
[152] Yanqi Su, Zhenchang Xing, Chong Wang, Chunyang Chen, Xiwei Xu, Qinghua Lu, and Liming Zhu. 2024. Automated Soap Opera Testing Directed by LLMs and Scenario Knowledge: Feasibility, Challenges, and Road Ahead. *arXiv preprint arXiv:2412.08581* (2024).
[153] Maojun Sun, Ruijian Han, Binyan Jiang, Houduo Qi, Defeng Sun, Yancheng Yuan, and Jian Huang. 2024. LAMBDA: A Large Model Based Data Agent. *arXiv preprint arXiv:2407.17535* (2024).
[154] Maojun Sun, Ruijian Han, Binyan Jiang, Houduo Qi, Defeng Sun, Yancheng Yuan, and Jian Huang. 2024. A Survey on Large Language Model-based Agents for Statistics and Data Science. *arXiv preprint arXiv:2412.14222* (2024).
[155] Weisong Sun, Chunrong Fang, Yifei Ge, Yuling Hu, Yuchen Chen, Quanjun Zhang, Xiuting Ge, Yang Liu, and Zhenyu Chen. 2024. A survey of source code search: A 3-dimensional perspective. *ACM Transactions on Software Engineering and Methodology* 33, 6 (2024), 1–51.
[156] Zeyu Sun, Qihao Zhu, Lili Mou, Yingfei Xiong, Ge Li, and Lu Zhang. 2019. A grammar-based structural cnn decoder for code generation. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 7055–7062.
[157] Yan Tai, Weichen Fan, Zhao Zhang, and Ziwei Liu. 2024. Link-context learning for multimodal llms. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 27176–27185.
[158] Xiaoyu Tan, Bin Li, Xihe Qiu, Chao Qu, Wei Chu, Yinghui Xu, and Yuan Qi. 2025. Meta-Agent-Workflow: Streamlining Tool Usage in LLMs through Workflow Construction, Retrieval, and Refinement. In *Companion Proceedings of the ACM on Web Conference 2025*. 458–467.
[159] Zihao Tang, Zheqi Lv, Shengyu Zhang, Fei Wu, and Kun Kuang. 2024. ModelGPT: Unleashing LLM's Capabilities for Tailored Model Generation. *arXiv preprint arXiv:2402.12408* (2024).
[160] Peri Tarr, Harold Ossher, William Harrison, and Stanley M Sutton Jr. 1999. N degrees of separation: Multi-dimensional separation of concerns. In *Proceedings of the 21st international conference on Software engineering*. 107–119.
[161] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2013. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 847–855.

[162] Suresh Thummalapenta, Saurabh Sinha, Nimit Singhania, and Satish Chandra. 2012. Automating test automation. In *2012 34th international conference on software engineering (ICSE)*. IEEE, 881–891.
[163] Alexander Tornede, Difan Deng, Theresa Eimer, Joseph Giovanelli, Aditya Mohan, Tim Ruhkopf, Sarah Segel, Daphne Theodorakopoulos, Tanja Tornede, Henning Wachsmuth, et al. 2023. Automl in the age of large language models: Current challenges, future opportunities and risks. *arXiv preprint arXiv:2306.08107* (2023).
[164] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).
[165] Patara Trirat, Wonyong Jeong, and Sung Ju Hwang. 2024. AutoML-Agent: A Multi-Agent LLM Framework for Full-Pipeline AutoML. *arXiv preprint arXiv:2410.02958* (2024).
[166] Haleh Vafaie and Kenneth De Jong. 1998. Evolutionary feature space transformation. In *Feature Extraction, Construction and Selection: a data mining perspective*. Springer, 307–323.
[167] A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems* (2017).
[168] Amala Mary Vincent and P Jidesh. 2023. An improved hyperparameter optimization framework for AutoML systems using evolutionary algorithms. *Scientific Reports* 13, 1 (2023), 4737.
[169] Chenglong Wang, Bongshin Lee, Steven Drucker, Dan Marshall, and Jianfeng Gao. 2025. Data Formulator 2: Iterative Creation of Data Visualizations, with AI Transforming Data Along the Way. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. 1–17.
[170] Jiaye Wang. 2024. Guiding Large Language Models to Generate Computer-Parsable Content. *CoRR* (2024).
[171] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science* 18, 6 (2024), 186345.
[172] Wei Wang, Huilong Ning, Gaowei Zhang, Libo Liu, and Yi Wang. 2024. Rocks coding, not development: A human-centric, experimental evaluation of LLM-supported SE tasks. *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 699–721.
[173] Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024. Executable code actions elicit better llm agents. *arXiv preprint arXiv:2402.01030* (2024).
[174] Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. 2024. OpenHands: An Open Platform for AI Software Developers as Generalist Agents. *arXiv preprint arXiv:2407.16741* (2024).
[175] Xindi Wang, Mahsa Salmani, Parsa Omidi, Xiangyu Ren, Mehdi Rezagholizadeh, and Armaghan Eshaghi. 2024. Beyond the limits: A survey of techniques to extend the context length in large language models. *arXiv preprint arXiv:2402.02244* (2024).
[176] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903* (2022).
[177] Lanning Wei, Huan Zhao, Xiaohan Zheng, Zhiqiang He, and Quanming Yao. 2024. A Versatile Graph Learning Approach through LLM-based Agent. arXiv:2309.04565 [cs.LG] https://arxiv.org/abs/2309.04565
[178] Shizheng Wen, Arsh Kumbhat, Levi Lingsch, Sepehr Mousavi, Praveen Chandrashekar, and Siddhartha Mishra. 2025. Geometry Aware Operator Transformer as an Efficient and Accurate Neural Surrogate for PDEs on Arbitrary Domains. *arXiv preprint arXiv:2505.18781* (2025).
[179] Lorenz Wendlinger, Emanuel Berndl, and Michael Granitzer. 2021. Methods for Automatic Machine-Learning Workflow Analysis. In *Machine Learning and Knowledge Discovery in Databases. Applied Data Science Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part V 21*. Springer, 52–67.
[180] Yongji Wu, Matthew Lentz, Danyang Zhuo, and Yao Lu. 2022. Serving and optimizing machine learning workflows on heterogeneous infrastructures. *arXiv preprint arXiv:2205.04713* (2022).
[181] Petros Xanthopoulos, Panos M Pardalos, Theodore B Trafalis, Petros Xanthopoulos, Panos M Pardalos, and Theodore B Trafalis. 2013. Linear discriminant analysis. *Robust data mining* (2013), 27–33.
[182] Tim Z Xiao, Robert Bamler, Bernhard Schölkopf, and Weiyang Liu. 2024. Verbalized Machine Learning: Revisiting Machine Learning with Language Models. *arXiv preprint arXiv:2406.04344* (2024).
[183] Doris Xin, Eva Yiwei Wu, Doris Jung-Lin Lee, Niloufar Salehi, and Aditya Parameswaran. 2021. Whither automl? understanding the role of automation in machine learning workflows. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–16.
[184] Jinglue Xu, Zhen Liu, Nagar Anthel Venkatesh Suryanarayanan, and Hitoshi Iba. 2024. Large Language Models Synergize with Automated Machine Learning. *arXiv preprint arXiv:2405.03727* (2024).
[185] Chengrun Yang, Yuji Akimoto, Dae Won Kim, and Madeleine Udell. 2019. OBOE: Collaborative filtering for AutoML model selection. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 1173–1183.
[186] Zekang Yang, Wang Zeng, Sheng Jin, Chen Qian, Ping Luo, and Wentao Liu. 2024. AutoMMLab: Automatically Generating Deployable Models from Language Instructions for Computer Vision Tasks. *arXiv preprint arXiv:2402.15351* (2024).

[187] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems* 36 (2024).
[188] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *International Conference on Learning Representations (ICLR)*.
[189] Yifan Yao, Jinhao Duan, Kaidi Xu, Yuanfang Cai, Zhibo Sun, and Yue Zhang. 2024. A survey on large language model (llm) security and privacy: The good, the bad, and the ugly. *High-Confidence Computing* (2024), 100211.
[190] Ziyang Ye, Triet Huynh Minh Le, and M Ali Babar. 2025. LLMSecConfig: An LLM-Based Approach for Fixing Software Container Misconfigurations. In *2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR)*. IEEE, 629–641.
[191] Caiyang Yu, Xianggen Liu, Wentao Feng, Chenwei Tang, and Jiancheng Lv. 2023. GPT-NAS: Evolutionary neural architecture search with the generative pre-trained model. *arXiv preprint arXiv:2305.05351* (2023).
[192] Shuang Yu, Tao Huang, Mingyi Liu, and Zhongjie Wang. 2023. Bear: Revolutionizing service domain knowledge graph construction with llm. In *International Conference on Service-Oriented Computing*. Springer, 339–346.
[193] Yue Yu, Yuchen Zhuang, Jieyu Zhang, Yu Meng, Alexander J Ratner, Ranjay Krishna, Jiaming Shen, and Chao Zhang. 2024. Large language model as attributed training data generator: A tale of diversity and bias. *Advances in Neural Information Processing Systems* 36 (2024).
[194] Mohammad Zarour, Hamza Alzabut, and Khalid T Al-Sarayreh. 2025. MLOps best practices, challenges and maturity models: A systematic literature review. *Information and Software Technology* 183 (2025), 107733.
[195] Carlos Vladimiro González Zelaya. 2019. Towards explaining the effects of data preprocessing on machine learning. In *2019 IEEE 35th international conference on data engineering (ICDE)*. IEEE, 2086–2090.
[196] Dan Zhang, Sining Zhoubian, Min Cai, Fengzu Li, Lekang Yang, Wei Wang, Tianjiao Dong, Ziniu Hu, Jie Tang, and Yisong Yue. 2025. DataSciBench: An LLM Agent Benchmark for Data Science. arXiv:2502.13897 [cs.CL] https://arxiv.org/abs/2502.13897
[197] Hengrui Zhang, August Ning, Rohan Baskar Prabhakar, and David Wentzlaff. 2024. LLMCompass: Enabling Efficient Hardware Design for Large Language Model Inference. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 1080–1096.
[198] Jianyu Zhang, Jianye Hao, Françoise Fogelman-Soulié, and Zan Wang. 2019. Automatic feature engineering by deep reinforcement learning. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. 2312–2314.
[199] Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. [n. d.]. AFlow: Automating Agentic Workflow Generation. In *The Thirteenth International Conference on Learning Representations*.
[200] Jie M Zhang, Mark Harman, Lei Ma, and Yang Liu. 2020. Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering* 48, 1 (2020), 1–36.
[201] Lei Zhang, Yuge Zhang, Kan Ren, Dongsheng Li, and Yuqing Yang. 2023. Mlcopilot: Unleashing the power of large language models in solving machine learning tasks. *arXiv preprint arXiv:2304.14979* (2023).
[202] Michael R Zhang, Nishkrit Desai, Juhan Bae, Jonathan Lorraine, and Jimmy Ba. 2023. Using large language models for hyperparameter optimization. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*.
[203] Shujian Zhang, Chengyue Gong, Lemeng Wu, Xingchao Liu, and Mingyuan Zhou. 2023. Automl-gpt: Automatic machine learning with gpt. *arXiv preprint arXiv:2305.02499* (2023).
[204] Sai Zhang, Zhenchang Xing, Ronghui Guo, Fangzhou Xu, Lei Chen, Zhaoyuan Zhang, Xiaowang Zhang, Zhiyong Feng, and Zhiqiang Zhuang. 2025. Empowering agile-based generative software development through human-ai teamwork. *ACM Transactions on Software Engineering and Methodology* (2025).
[205] Yixuan Zhang, Mugeng Liu, Haoyu Wang, Yun Ma, Gang Huang, and Xuanzhe Liu. 2024. Research on WebAssembly Runtimes: A Survey. *ACM Transactions on Software Engineering and Methodology* (2024).
[206] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 19632–19642.
[207] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223* (2023).
[208] Mingkai Zheng, Xiu Su, Shan You, Fei Wang, Chen Qian, Chang Xu, and Samuel Albanie. 2023. Can gpt-4 perform neural architecture search? *arXiv preprint arXiv:2304.10970* (2023).
[209] Shi-Yao Zhou and Chung-Yen Su. 2021. A novel lightweight convolutional neural network, ExquisiteNetV2. *arXiv preprint arXiv:2105.09008* (2021).
[210] Xuanhe Zhou, Xinyang Zhao, and Guoliang Li. 2024. LLM-Enhanced Data Management. *arXiv preprint arXiv:2402.02643* (2024).
[211] Zhi-Hua Zhou. 2021. *Machine learning*. Springer nature.
[212] Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. 2024. Toolqa: A dataset for llm question answering with external tools. *Advances in Neural Information Processing Systems* 36 (2024).

[213] Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. 2024. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv preprint arXiv:2406.15877* (2024).

[214] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 8697–8710.

[215] Tianyuan Zou, Yang Liu, Peng Li, Jianqing Zhang, Jingjing Liu, and Ya-Qin Zhang. 2024. FuseGen: PLM Fusion for Data-generation based Zero-shot Learning. *arXiv preprint arXiv:2406.12527* (2024).