ESP IJAST
ESP International Journal of Advancements in Science & Technology
ISSN: 2583-9233 / Volume 4 Issue 1 February 2026 / Page No: 27–42
Paper Id: IJAST-V4I1P104 / Doi: 10.56472/25839233/IJAST-V4I1P104

Original Article

AI-Augmented Data Engineering: Paradigms, Patterns, and Future Directions

Amol Bhatnagar
Director of Cloud Architecture, Sogeti-Capgemini USA.

Received Date: 29 December 2025 Revised Date: 20 January 2026 Accepted Date: 08 February 2026

Abstract: *The rapid development arena of Artificial Intelligence (AI), especially from Large Language Models (LLMs), has prompted a drastic change in the way data engineering is practised. This paper provides an in-depth overview of AI-powdered data engineering, and discusses how modern AI techniques are re-shaping the conventional development, profiling/optimisation, and maintenance process of the data pipeline. We then study recent paradigms in so-called pipeline automation with LLMs, the growing use of machine learning to optimise database query execution, and ponder over risks and governance that should be considered when allowing AI-driven data flows. By taking a systematic approach to analysing state-of-the-art solutions, we distil commonalities from existing solutions- prompt-based pipeline construction, smart schema generation, automatic code generation and autotuning database architectures. What we have found is that there are tremendous opportunities and challenges associated with this burgeoning area, such as opening the door for hallucination and security attacks, spreading biases and requiring a governance framework to be in place. We suggest a layered governance model with technical and organisational controls complemented by ongoing oversight. Finally, we discuss areas for future research, including the creation of dedicated data engineering LLMs, more powerful human-AI collaboration paradigms, and the direction in which autonomous self-healing data infrastructure is evolving. This paper contributes to the emerging and increasingly important field of AI-enhanced software engineering through its emphasis on applying such a methodology to data engineering fields.*

Keywords: *AI-Augmented Data Engineering, Large Language Models, Automated Pipeline Generation, Data Workflow Optimization, AI Governance, Mlops, Dataops.*

I. INTRODUCTION

Data engineering has become an important discipline in the contemporary data-driven organisation, concerned with creating, constructing and maintaining systems for collecting, storing, transforming and serving large datasets [1]. The manual labour required to set up complex data pipelines and the steep learning curve make it difficult for many organisations to afford an engineering team that can maintain such a setup. These challenges have become more exacerbated with the emergence of the Big Data era, while the volume, velocity, and variety of data continue to grow [2].

The rise of Large Language Models (LLMs) like GPT-4, Claude, and specialised models for coding has set up ground-breaking possibilities in the enhancement and automation of data engineering tasks [3]. These models exhibit impressive skills in code writing, natural language understanding, and pattern recognition, leading to their great suitability for solving repetitive, complex and knowledge-work-intensive parts of the data engineering work [4].

AI-enhanced data engineering is the next-gen evolution of traditional human (or script-based) automation to intelligent systems that can understand requirements, expressed in even natural language, and produce corresponding data processing code, optimise pipeline performance and even self-diagnose and repair failures [5]. This paradigm shift has the potential of making data engineering accessible to the masses by removing technical barriers, as well as speeding up development cycles due to automated code generation and making systems more reliable with intelligent monitoring and optimisation [6].

But along with this technological progress also comes new difficulties and dangers. Using AI in essential data infrastructure raises issues including robustness, security, bias (or fairness), explainability, and governance [7]. LLMs may emit incorrect or insecure code ("hallucination"), can unintentionally embed biases from training data, and introduce opaque and unwieldy dependencies [8].

A. Motivation and Scope

The work presented in this paper was motivated by the growing interest from industry in applying AI-assisted development tools and their desire to bring the same systematic understanding about using such tools to the data engineering domain as well. Though general-purpose coding assistants, like GitHub Copilot, have been widely adopted, Data engineering requires special challenges such as handling complex state, managing data at the best quality level and tuning performance at scale for regulators' compliance [9].In this paper, there are three main focal points: (1) the use of LLMs in for automating pipeline generation and optimizing, with an emphasis on current approaches as well as capabilities and limitations; (2) risk factors that emerge from AI-driven data workflows which should be taken into account when balancing technical, operational, and ethical concerns; and (3) governance structures to ensure a responsible implementation of AI-enhanced data engineering practices.

B. Contributions

The main contributions of this work are:

- A new taxonomy for current paradigms and patterns of AI-augmented data engineering
- Systematic LLM applications in automated pipeline generation and optimisation.
- Identification and classification of risks in AI-based dataflows
- Governing AI-enabled data engineering at multiple levels
- Future and emerging trends discussion

C. Paper Organization

The rest of this paper is organised as follows: Section II gives a brief overview of data engineering and LLM features. We discuss the current state of AI empowered data engineering in Section III. In IV, we describe the uses of LLMs in automatic pipeline design. Section V discusses optimisation techniques. Risks and challenges are discussed in Section VI. Section VII presents governance frameworks. Section VIII discusses future work , and Section IX concludes.

II. BACKGROUND AND RELATED WORK

A. Data Engineering the Old-Fashioned Way

Data engineering refers to the designing and building of systems that collect, store, and process data at scale [10]. Data Engineering Processes. The traditional data engineering process will typically involve multiple key activities, including but not limited to the following: i) Data Ingestion from various sources; ii) Data Transformation using ETL (Extract-Transform-Load) or ELT (Extract-Load-Transform); iii) Data quality validation; iv)Pipeline orchestration and v) Monitoring [11]. These are commonly implemented using tools like Apache Spark, Apache Airflow, data build tool (dbt) and different cloud services [12].Modern data engineering poses several challenges: heterogeneity of data sources and formats, different requirements in latency, both batch and streaming, needing to scale to large volumes of data, governance and compliance with regulatory rules, provisions for quality control over the data ingested, as well as lineage tracking [13]. Addressing these challenges historically demanded the mastery of distributed systems, database technologies and programming in Python, SQL, Scala, among others [14].

B. Large Language Models for Code Generation

Recently, Large Language Models have shown impressive successes in code generation and understanding [15]. Models such as OpenAI's Codex, Google's PaLM-Coder, and Meta's Code Llama have been pre-trained on large corpora of source code mined from public repositories, allowing them to synthesise syntactically(cor) rect and often semantically( ) dignified code from natural language descriptions [16].It has been reported that LLMs can achieve competitive performance on coding benchmarks like HumanEval and MBPP [17]. These approaches use transformer architectures with attention to long-term code dependencies: they can learn the context, patterns of API usage and typical coding idioms [18]. Newer approaches have been fine-tuned for code-related tasks, considering execution feedback, leveraging retrieval-augmented generation to ground responses in relevant documentation [19].

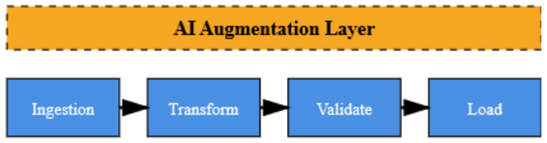

Figure 1 : AI Augmentation Layer Overlaying Traditional Data Pipeline Stages

C. Related Work in AI-Assisted Software Engineering

Use of AI in SE AI has been investigated in terms of many dimensions with regards to the domain of software engineering. Chen et al. [20] introduced AI pair programming and benchmarked Codex for functional correctness. Barke et al. [21] examined the interaction between developers and code generation tools with a focus on how iterative update and validation occur. Hou et al. [22] studied LLMs in the AV domain for bug fixing and program repair.

More specific to data-centric applications, there has even been work on LLMs for SQL generation [23], data transformation script synthesis [24], and automatic data cleaning [25]. Yet `full life cycle' AI augmentation for data engineering is still under-explored. This paper fills this gap, presenting an integrated perspective on AI-augmented data engineering practices, risks and governance needs.

III. CURRENT PARADIGMS IN AI-AUGMENTED DATA ENGINEERING

Data Engineering with AI has materialised in a number of paradigms, each indicative of a particular approach to using artificial intelligence for the development and management of data workflows. This chapter describes these paradigms and then focuses on common patterns which have emerged in practice.

A. Paradigm 1: Conversational Flow Design

The conversational approach considers data pipeline development as a dialogue between a data engineer and an AI assistant [26]. Engineers state requirements in natural language, and the AI system synthesises pipeline components that satisfy the requirements while improving these components by iteration according to feedback. 133This work builds on principles of such few-shot learning, with examples of similar pipelines or transformations being supplied as context in order to steer generation [27].

In practice, this philosophy results in tools that let engineers describe the transformations they want as "aggregate sales data by region and calculate month-over-month growth", and get back SQL/Pandas code or domain-specific languages like dbt [28] that actually perform those operations. The informal style lowers the entry level for inferring programmers, and at the same time supports quick prototyping as well as alternate speculations [29].

B. Paradigm 2: Template-Based Generation

Template-based generation pre-packages specific pipeline patterns with AI-driven customisation [30]. This model has a repository of pre-evaluated pipeline templates for common use-cases such as CDC replication, dimension modelling, event stream processing and leverages LLMs to customise these templates for specific needs. The benefit of this method is that it bounds the space in which to generate, lowering the risk for mistakes and exploitable code, since there is still room for creativity available [31].

Organisations that have embraced this paradigm often develop internal template libraries that capture institutional knowledge, best practices, and compliance requirements. It determines suitable templates of which parameters, schema mappings and business logic are customised based on natural language descriptions [32].

C. Paradigm 3: Retrieval-Augmented Generation

Retrieval-augmented generation (RAG) is a way to equip LLMs with the ability to utilise external knowledge sources in the course of generating generations [33]. For data engineering, this could be documentation, codebases, data catalogues, and schema definitions. The system then pulls relevant context, based on the user's query, and delivers this information to the LLM, which grounds organisation-specific knowledge. [34].

Table 1 : Comparison of AI-Augmented Data Engineering Paradigms

Paradigm	Flexibility	Control	Learning Curve	Risk Level
Conversational	High	Medium	Low	Medium-High
Template-Based	Medium	High	Low-Medium	Low-Medium
RAG-Enhanced	High	Medium-High	Medium	Medium
Agent-Based	Very High	Low-Medium	Medium-High	High

Data Engineering For data engineering use cases, RAG systems commonly connect with existing or vertical-specific data catalogues (eg, DataHub, Amundsen) to fetch schema information and lineage graphs as well as metrics on data quality. This context information will aid the LLM in producing code that accurately refers to tables, complies with the data types and reflects established patterns [35].

D. Paradigm 4 Agent-Based Autonomous Systems

The most developed paradigm consists of autonomous agents who are able to plan, execute and supervise complex data workflows with little human intervention [36]. These frameworks use reinforcement learning and multi-agent systems methods to break the large objectives into smaller running tasks, handling errors, performance optimisations, etc.[37].

They can automatically crawl for new data sources, infer schemas, create transformation logic, verify generated results and deploy the pipeline to production [38]. Although providing the best opportunity for automation, this model is riskiest and demands more advanced governance [39].

E. Common Patterns Across Paradigms

There are some general trends across these paradigms:

- Schema-First Design: The most successful AIa approaches start with explicit schema specification or induction, and use the learned structure to guide and constrain further generation [40].
- Iterative Refinement: Instead of immediately producing complete and correct solutions in one go, effective systems produce a first stab at design that is tested, validated and refined against feedback [41].
- Hybrid Automation: Real-world deployments pair Automated Generation with human inspection check-points at key junctures, a trade-off between the efficiency and robustness of systems [42].
- Prompt Engineering: AI augmentation is only as good as the quality of prompts, which need to be made such that the model can well understand them, and it demonstrates great naturalness even as they are computer-written [43].

IV. LLMS FOR AUTOMATED PIPELINE GENERATION

One of the most promising applications of LLMs in data engineering is automatic pipeline generation. This section discusses specific methods and capabilities (limitations) in automatically generating the data processing code using LLMs.

A. NL-to-SQL/Code Conversion

One of the most advanced applications of LLMs in data engineering is translation from natural language to SQL or other code for processing data [44]. LLMs in a more recent style can understand complicated business questions and produce precise SQL queries which involve joins, aggregations, window functions and CTEs (Common Table Expressions) [45].

Research by Rajkumar et al. [46] reported that fine-tuned LLMs can reach over 80% accuracy on text-to-SQL benchmarks with schema context. These challenges include resolving mentions, disambiguating queries, and generating a query respecting both data availability and performance constraints [47].

In more evolved forms, it includes execution feedback: executing generated code combines querying the system with evaluating results or errors and refining the code/enquiry until correct output is generated [48]. While this approach has greatly improved success rates, it relies on the ability to execute code in highly controlled and isolated environments [49].

B. ETL/ELT Pipeline Synthesis

LLMs can derive full ETL/ELT pipelines directly from high-level specifications [50]. This includes developing code to extract data from disparate data sources (databases, APIs, files), transforming logic (involving cleaning of raw data and type conversion, as well as applying/generating business rules), and loading it into target systems [51].

For example, given a requirement of Extract customer transactions from PostgreSQL, calculate their lifetime value and load them into Snowflake data warehouse with daily refresh, an LLM would create the necessary extraction queries (to read data out), transformation logic in Python or SQL(Error Handling) and orchestration code for frameworks such as Apache Airflow [52].

C. Schema Inference and Mapping

LLMs show good performance in automatically inferring data schemas of unstructured or semi-structured data and generating mappings between different schema representations [53]. Through sample data or schema documents, LLMs could generate suitable data types, discover relationships between entities and propose normalisation actions [54].

D. Schema Inference and Mapping

LLMs show good performance in automatically inferring data schemas of unstructured or semi-structured data and generating mappings between different schema representations [53]. Through sample data or schema documents, LLMs could generate suitable data types, discover relationships between entities and propose normalisation actions [54].

This feature is especially useful when integrating data from different systems is required. LLMs are capable of analysing source and targetschemas, matching the corresponding fields even when they have different names and generating transformation code for type conversion and structural differences [55].

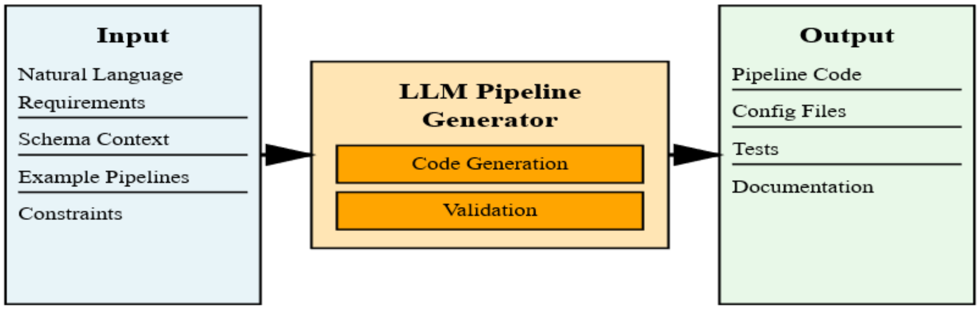

Figure 2 : LLM-Based Automated Pipeline Generation Workflow

E. Schema Inference and Mapping

LLMs show good performance in automatically inferring data schemas of unstructured or semi-structured data and generating mappings between different schema representations [53]. Through sample data or schema documents, LLMs could generate suitable data types, discover relationships between entities and propose normalisation actions [54].

This feature is especially useful when integrating data from different systems is required. LLMs are capable of analysing source and targetschemas, matching the corresponding fields even when they have different names and generating transformation code for type conversion and structural differences [55].

F. Generating Code with dbt and Modern Data Stack

The modern data stack, specifically tools like dbt (data build tool), lends itself nicely to LLM-assisted development [56]. DBT is declarative and SQL-focused, which makes it a great fit for LLM's skills in writing SQL. LLMs can produce dbt models, tests, documentation, and even sophisticated Jinja templating from natural language descriptions [57].

For example, an engineer might ask a question such as “Create a dbt model that calculates customer cohort retention rates on a monthly granularity,” and the LLM outputs concrete files in accordance with the request— including SQL logic, schema definition, tests for data quality checks, and documentation [58]. This speeds up development, while keeping the modularity and testability, dbt adds [59].

G. Handling Complex Transformations

Basic transformations are easily automatable, but complex business logic requiring domain expertise is considerably more challenging [60]. LLMs have difficulty with transformations that rely on a deep understanding of business context, multi-step reasoning, or knowledge that is not well represented in their training data [61].

Strategies that are beneficial for complex transformations include detailed context and examples in prompts, creating a decomposition of the complex tasks into smaller sub-tasks to be solved incrementally, or adding domain-specific knowledge with RAG or fine-tuning [62]. Human intervention, however, is still necessary to validate complex transformation logic [63].

H. Limitations and Failure Modes

A number of limitations impact the trustworthiness of LLM described pipelines [64]:

- Hallucination: LLMs have the potential to produce syntactically valid code, but semantically invalid, such as references to non-existing functions or incorrect usage of an API, and valid logic that does not satisfy the requirements [65].
- Performance in the Dark: Generated code may appear correct, but could perform poorly at scale due to missing optimisations such as proper indexing, partitioning, or query structure [66].

- Security: LLMs might produce code that is vulnerable to SQL injection, bad error processing, or insufficient access controls [67].
- Context Limitations: Although state-of-the-art models used in our experiments are able to see a long code context, LLMs still have trouble being consistent within very big codebases or intricate systems [68].

V. OPTIMIZATION IN AI-DRIVEN PIPELINES

Beyond generating pipelines to begin with, AI systems are able to iterate and refine data workflows over time for performance, cost, and reliability optimisation. In this section, we review methods for AI-based optimisation in data engineering.

A. Performance 1) Query and Pipeline Optimisation

LLMs may be used to examine the available data pipeline and propose performance enhancements [69]. This can be as simple as rewriting the query into a more efficient form, but could also include advising on an appropriate indexing strategy, suggestions for partitioning or finding opportunities for materialisation or caching [70].

For instance, your LLM could look at a slow query that's doing lots of subqueries and rewrite it to make use of CTEs or window functions for speed. Research by Wang et al. [71], LLMs trained on query execution plans can recommend optimisations that yield an improvement of 30-50% in average speed.

State-of-the-art optimisation systems integrate static code analysis with dynamic performance metrics. Through the extraction of execution plans, resource consumption, and bottlenecks, those systems are able to reveal individual problems and derive focused optimisations [72]. LLMs can use cost-based analysis and integration with query optimisers to make decisions based on their data instead of by using system heuristics [73].

B. Intelligent Resource Allocation

For example, systems based on AI can flexibly allocate resources on the data pipeline according to the workload and performance requirements [74]. Machine learning models that have been trained on historical execution data can predict resource demand, determine the best configuration of a cluster, and schedule them to minimise costs and meet SLAs [75].

Intelligent resource allocation is especially beneficial for cloud data platforms. Systems can automatically scale compute resources, change the limits of concurrency and provision instance types depending on workload characteristics [76]. For example, the Netflix data platform in production utilises ML models to determine Spark job resource demand and a 40% reduction in over-provisioning is observed while delivering sustainable performance was reported [77].

C. Automated Data Quality Optimisation

LLMs can perform thorough data quality checks and even recommend optimisations in order to improve the reliability of new data [78]. Through examining data schemas, business rules and known quality problems, these systems suggest validation rules, anomaly detection logic and data profiling queries [79].

Smart DQ systems adapt and learns from quality incidents, spotting the patterns amongst data anomalies and their causes. It also allows quality control in advance of the dissemination, by spotting and repairing issues before they cascade through the chain [80]. The existing approaches, like Great Expectations and others, are getting injected with AI to automatically generate test suites based on data characteristics [81].

D. Cost Optimisation

Cloud data warehouse costs can quickly grow out of control if pursued without due diligence [82]. Query patterns, storage usage, and compute consumption are analysed by AI systems in order to identify potential cost-saving opportunities [83]. This involves advising on how to cluster and partition tables, remove unused or redundant data, rewrite queries en masse to reduce scan volumes, as well as materialised view manipulations [84].

LLMs can also examine DAGs of data pipelines to find places for parallelisation, combining similar jobs, and reducing redundant computation [85]. These systems, by interpreting data lineage and dependencies, ensure that optimizations do not sacrifice the freshness or correctness of the data [86].

E. Self-Healing Pipelines

AI-enhanced pipelines that can identify failure, diagnose root causes, and apply fixes automatically [87]. This includes watching pipeline runs, looking at error logs and stack traces, coming up with a patch candidate and testing it, then applying the fix to the codebase as needed [88].

Self-healing systems borrow from the world of automated program repair, in which LLMs have been used to comprehend error messages and automatically infer what the likely causes are, so we can generate a patch [89]. When a pipeline fails because of schema changes happening in the source system, for example, the system could report a mismatch so new transformation logic can be identified and applied [90]. Nevertheless, the self-healing feature should be strictly controlled to avert potential adverse effects [91].

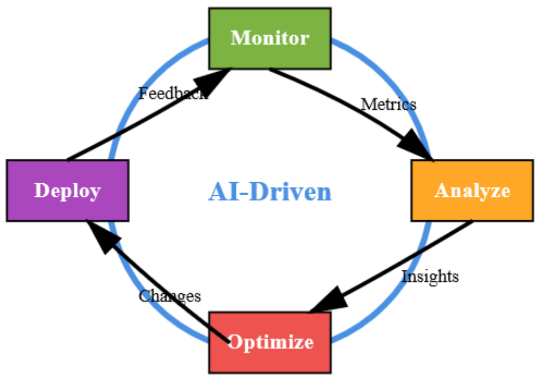

Figure 3 : Continuous Optimization Cycle in AI-Augmented Data Pipelines

VI. RISKS AND GOVERNANCE OF AI-DRIVEN DATA ENGINEERING

However, AI augmentation also has significant associated risks that organisations need to manage using a robust governance model. This section discusses technical, operational and ethical issues in AI-based data engineering.

A. Technical Risks

a) Code Correctness and Hallucination

The closest technical threat is that LLM-generated code could have the shape of Glitch-sites generation, as in L MCs (Section 6), but filled with, say, logical errors [92]. Hallucination \{ where model outputs synthesise realistic, but impossible data \} may appear as nonexistent API endpoints, incorrect algorithms, or accurate business logic which is inconsistent with what is needed [93].

Studies by Liu et al. [94] even state that state-of-the-art code generation models generate functionally incorrect code in 20-30% of cases on complex problems. In data engineering, silent data corruption can occur because of these errors; the pipelines run successfully but generate slightly erroneous results which flow through downstream systems [95].

Mitigation strategies consist of comprehensive testing suites that run on the generated code, formal verification methods for key transformations and human inspection points for high-risk transformations [96]. Organisations should deploy differential testing, comparing the outputs from AI-produced pipelines with existing baselines [97].

b) Security Vulnerabilities

LLMs trained on public code repositories might have learned insecure patterns from their training data [98]. Code that is generated automatically can introduce SQL injection, insufficient input validation, hardcoded passwords, and limited access control [99].

Pearce et al. [100] analysed the security of GitHub Copilot and observed that 40% of code examples autogenerated by it were exploitable. In data engineering, we can restate the risks of security vulnerabilities as being those due to exfiltration of data, unauthorised access to sensitive information and compliance violations [101].

Autonomous AI scanners should be added to organisational CI/CD pipelines along with whitelists of accepted libraries and patterns and mandates for AI code security reviews involving sensitive data access [102]. Generated code common vulnerabilities can be identified by integrations with tooling such as Snyk, SonarQube or Bandit [103].

c) Performance and Scalability Issues

LLMs have shallow configuration settings and no knowledge of performance characteristics of distributed systems, so it is easy to generate code that runs on a small dataset but blows up in production [104]. Identification issues often found include: inefficient join strategies, lack of appropriate partitioning, too much data shuffling and unbounded operations requiring larger-than-necessary memory buffers [69].Performance testing using production-size data volumes is necessary before deploying AI-generated pipelines [106]. Various tests, including load testing, profiling and query plan analysis, should be included in the pipeline validation [107].

d) Technical Debt and Maintainability

Fast AI-empowered coding can exacerbate technical debt buildup if not well-maintained [108]. At times, AI developed code can be....undocumented ... "inventing its own style"... establish unfounded dependencies and create an architecture that no one understands how to maintain [109].Research by Sandoval et al. [110] observed that AI-supported development may degrade the software quality metrics if developers blindly accept the suggestions without understanding them. Companies need to weigh such productivity gains against code quality norms (and introduce documentation and review of AI-generated components) [111].

B. Operational Risks

a) Reliability of Pipelines and Propagation of Failures

The combination of AI-driven automation can lead to increasingly complex systems which are tightly coupled and propagate failures quickly [112]. If an AI system creates a poor workflow that taints important data, such damage can spread amongst many downstream systems before it is noticed [113].Strong monitoring, circuit breakers and validation of data at the system boundary are key [114]. Organisations should use staged rollouts of AI made changes, gradual production traffic exposure, automation for rollback and rich alerting on data quality metrics [115].

b) Knowledge Transfer and Skill Depreciation

Over-dependence on AI-authored code can degrade the skills of data engineers to comprehend, debug, or modify complex pipelines [116]. This introduces an organisational risk in case the AI system is unavailable to provide answers or in case it provides erroneous outputs that need manual intervention [117].Striking a balance between automation and skill development demands that it is done intentionally: by making engineers review and understand auto-generated code; by rotating who gets to boss the Learning so multiple others in a team can still "work" without assistance from AI; or by providing documentation of rules that show how they were implemented, ideally expressed as short stories [118].

c) Vendor Lock-in and Dependent Risks

In building critical infrastructure on expensive proprietary AI services, organisations risk being locked in [119]. When an AI service endows capability, or changes the price, availability and so on, the dependent systems are hardly maintained [120].

Ways to reduce this risk include not building provider-specific abstractions, storing generated code in version control and developing internal expertise that reduces dependence on external services [121].

Table 2 : Risk Assessment Matrix for AI-Augmented Data Engineering

Risk Category	Specific Risk	Likelihood	Impact	Priority
Technical	Code Hallucination	High	High	Critical
Technical	Security Vulnerabilities	Medium	Very High	Critical
Technical	Performance Issues	Medium	Medium	High
Operational	Failure Propagation	Medium	High	High
Operational	Skill Degradation	High	Medium	Medium
Ethical	Bias Propagation	Medium	High	High
Compliance	Regulatory Violations	Low	Very High	Critical
Compliance	Audit Trail Gaps	High	Medium	High

C. Ethical and Bias Risks

a) Bias in the Processing Logic for Data

LLMs trained on data with bias can create pipeline logic that reflects, or even amplifies, the biases present in society [122]. This is a great concern if pipelines are processing data used to make decisions about people, such as credit scoring, hiring or resource allocation [123].

For example, an LLM in use might induce segmentation logic that reproduces discriminatory groups or aggregate metrics which harm protected groups [124]. Should AI-generated pipelines for processing PII or providing an output used in decision-making be bias tested [125]?

b) Privacy and Data Protection

Pipeline generation with LLMs might also lead to accidental sharing of sensitive information [126]. When engineers supply sample data or schema to AI systems, that data can be captured in logs, used for model training or included in the model output [127].

Organisations should have clear guidelines on what they can share with AI systems, anonymise data related to examples and context, use either on-premises or private AI deployments for sensitive workloads, and regularly audit the access logs of their AI system [128].

c) Transparency and Explainability

AI-produced pipelines can be "black boxes" whereby the motivation for particular implementation decisions is not known [129]. This makes debugging difficult, harder to determine compliance with business rules and reduces trust in data outputs [130].

Practices and tools for good practice include: forcing AI systems to produce explanatory remarks with code produced and capture our documentation of prompts/contexts used to generate itg in training lineages that record when and how AI is involved in pipeline development [131].

D. Compliance and Regulatory Risks

a) Regulatory Compliance

Regulated enterprises (such as financial, healthcare, and government) are required to satisfy strict standards on data management [132]. AI models must comply with regulations such as GDPR, HIPAA, SOC 2 and industry standards [133].

Obstacles include ensuring that code generated by AI enforces the necessary access controls and encryption, maintains sufficient audit trails, complies with data retention and deletion mandates and navigates cross-border restrictions on data transfer [134]. Validation for compliance needs to be built into the pipeline approval process [135].

b) Auditability and Documentation

Regulatory inspections demand full documentation of data processing events [136]. Automated pipelines produced through AI will need to be well-documented so that they can demonstrate compliance, justify processing logic, and provide an ability to reproduce results [137].

Automate documentation generation for AI-generated pipelines, version all pipeline components, including generation prompts, change logs which highlight when and where AI modification took place, and unquestionable testing records showing how generated code was verified [138].

E. Governance Framework

Responsive AI-Augmented Data Engineering governance. Effective governance of the AI-augmented data engineering should be multi-layer, including technical controls, organisational policies, as well as cultural practices [139].

a) Technical Governance Controls

12.2 Technical safeguards are the starting point for AI governance [140]:

- Automated Assurance: All AI-gene rated code must continuously pass automated test cases at all levels, such as unit testing, integration testing, security scanning and performance benchmarking before deployment [141].
- Sandboxed execution: The pipelines generated by the AI should be evaluated in an isolated environment that is similar to production, but in which it cannot access or modify production data [142].

- Version Control: All AI conversations, prompts and outputs generated must be versioned as traditional code so that they can be reproduced and rolled back [143].
- Monitoring and Observability. Improved Monitoring for AI-Generated Pipelines should monitor data quality, execution patterns, and outliers that could signal generation errors [144].

b) Organisational Policies

Policies can be actively controlled in this process for the right kind of AI use within data engineering [145]:

- Use Case Classification: Specify what types of pipelines should be built by AI generation (low-risk, well-understood patterns) and what type of pipeline will take traditional development (high-risk, novel logic) [146].
- Review Requests: Ensure reviews by humans are compulsory at certain risk thresholds, with critical pipelines reviewed by senior engineers and non-critical changes being handled through an automated system [147].
- Data Sharing Policies: Define the necessary data that can be shared with AI systems, and more rigorously manage sensitive information [148].
- Responsibility Boundary: Affirm ownership of AI-driven code by holding engineers responsible for deployed pipelines, irrespective of how they were generated [149].

c) Training and Culture

Organisational culture has to change, and how it will evolve with AI augmentation, and quality standards [150]:

- Engineer Training: Data explainers require training on agile engineering, limitations of AI and effective human-AI patterns for collaboration [151].
- Culture of Quality: AI is a productivity enhancer and not a usurper for critical judgment and quality assurance [152].
- Iterative improvement: Create feedback loops that enable engineers to report AI-generated errors, so that generating processes can iteratively improve [153].

VII. FUTURE OF AI-AUGMENTED DATA ENGINEERING

The landscape of AI-empowered data engineering is changing fast. In this section, we highlight new trends, advances in technology and future research directions to drive the next wave of data infrastructure.

A. Specialised Data Engineering LLMs

Though they are impressive, today's general-purpose LLMs may be supplemented by task-specific models trained specifically for data engineering purposes [154]. Such models could then be fine-tuned on domain-specific corpora like data pipeline code, data modelling documentation, query optimisation literature and best practices in data engineering[155].

Specialized models may know about distributed systems performance, data storage formats with their trade-offs and implications on physical storage (s3, hdfs), there can be an understanding of how to talk about quality \(and would punch a hole in Little's box \) or think through the completeness of data, it could also generate code optimized for particular platforms such as Spark/Snowflake/BigQuery [156].

Developing benchmark datasets that are specifically designed for data engineering tasks; Defining evaluation metrics to measure the functional correctness, performance and maintainability of pipelines; As we could not find any work that has experimented with (c)(d) and multi-modal classification strategies [157].

B. Enhanced Human-AI Collaboration

Upcoming systems will outgrow mere code generation and advance into a stage of intelligent cooperation where AI acts as a smart classmate in the whole development lifetime [158]. This includes interactive design sessions where engineers and AI systems together explore solution spaces, real-time feedback where AI proposes enhancements while engineers are authoring code, and collaborative debugging where AI helps with root cause analysis and fix generation [159].

Next-generation interfaces may offer conversational guidance, where users can ask questions about the process being developed [14], while doing so in visual programming environments where AI assists in creating pipelines using graphical manipulation [121], or even code-writing modes that use reinforcement learning to observe user actions and provide proactive suggestions [125].

Primary research challenges include building successful mental models for AI-human collaboration, crafting interfaces that foster trust and scepticism, and designing feedback loops to enable the AI to get better at offering assistance over time [161].

C. Autonomous Data Infrastructure

The future mindset being nurtured is that of an autonomous data infrastructure that self-configures, behaves and heals with little human involvement [162]. In such a case, systems could be built with multi-agent architectures where agents could specialise on different data management tasks: ingestion agents that would be responsible for discovering and connecting to new sources of data, transformation agents that can learn intuitive business logic required in queries, and optimisation agents that would allow us to continually improve performance [163].

These would experience and learn from real operations, recognising patterns of failure and success, adapting to a changing data profile (a workload) and optimising for objectives and constraints at an organisation-specific level [164].

However, fully autonomous systems greatly raise concerns about control, accountability, and failure modes [165]. There are extensive research challenges around what principled means of preserving human oversight can exist concurrent with autonomy, how to ensure safe exploration and experimentation, and fail-safe systems that prevent catastrophic failures [166].

D. Integration with DataOps/MLOps

There will be more and even tighter integration between AI-accelerated data engineering, the wider practices of DataOps, and MLOps [167]. This convergence essentially involves end-to-end automation from raw data to deployed models, including automatic data pipeline generation, feature engineering automation (automated search of predictive features), model training and optimisation, and deployment together with AI systems that provide insight into these processes as they are executing [168].

Systems of the future might offer environments where data scientists can express desired analyses in natural language, and then the system automatically produces necessary data pipelines, feature transformations, and model training code [169]. It could be a way to massively speed up the business question to the production model timeline [170].

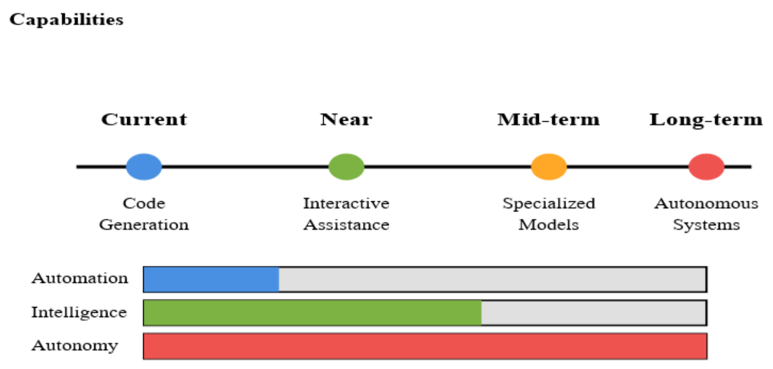

Figure 4 : Evolution Timeline and Capability Progression in AI-Augmented Data Engineering

E. Trustworthy and Explainable AI Systems

As AI assumes increasingly important roles in data infrastructure, interpretability and trustworthiness are of very high importance [171]. Future works will need to devise mechanisms for explaining AI-based decisions for the pipeline code that should be generated, assigning confidence scores to the produced code and assisting users in human verification of complex logic [172].

Emerging approaches include creating natural language explanations accompanying code, visualisations of the reasoning applied to make design decisions, and access to provenance on different ways in which patterns were learned or selected [173]. These capabilities help data engineers understand, trust and in concert with AI systems [174].

F. Federated and Privacy-Preserving AI

Data privacy is a growing concern for businesses applying AI services [175]. For future systems, we anticipate that federated learning (train or fine-tune a gene's model on-premises without the need to see sensitive data), differential privacy (preventing the recovery of training data), and secure multi-party computation for AI collaboration development [176].

These devices would allow companies to leverage AI augmentation while preventing sensitive information from falling out of their control in order to comply with data privacy regulations [177].

G. Standardisation and Interoperability

With the rise of AI-augmented data engineering, best practices will be adopted in the industry for expressing pipelines and documenting AI participation, as well as being able to port between platforms [178]. Standardisation works may identify common AI-assisted development tool interfaces, metadata models for documenting AI-generated code and good practices for governance and quality assurance [179].

Open source ecosystems will play an influential part in fostering collaborative tools, benchmarks and frameworks that avoid vendor lock-in and promote innovation [180].

H. AI and Responsible Innovation with Ethics

The data engineering fraternity should lead the way to ethical considerations as augmentation with AI becomes pervasive [181]. This would include the development of humane use guidelines for AI and data infrastructure; professional codes of ethics for AI-augmented development; audit frameworks to assess the impact of AI technologies [182]; and interdisciplinary dialogue between technologists, ethicists, and policymakers.

Standards bodies, industry organisations and academic institutions should work together to develop these ethical frameworks tailored toward AI-accelerated data engineering - driving innovation while being responsibly aware [183].

VIII. CONCLUSION

AI-assisted data engineering is a game-changing innovation in how data infrastructure has been architected, implemented and operated. The use of Large Language Models and similar AI-driven tech provides the first potential to automate the creation of pipelines, with optimisations for pipeline performance and self-healing systems. Contemporary paradigms—conversational design, template-based generation, retrieval-augmented approaches and autonomous agents—exemplify the potential as well as challenges of this technology.

LLMs are particularly good at compiling natural language requirements down to executable code, deducing schemas and translating complex transformations. But there are still major limitations, like code hallucination, security holes, performance blindspots and maintenance nightmares. The technical challenges are amplified by operational considerations involving reliability, skill atrophy and vendor lock-in, along with ethical considerations of fairness, privacy (Chouldechova, 2017), and transparency.

Robust governance is vital for safely scaling AI-augmented data engineering. These frameworks need to be built out of a mix of technical controls (automated validation, sandboxed execution, deep monitoring), organisational policies (clear use case classification, mandatory reviews, accountability structures) and cultural practices (engineer training, quality focus, Kaizen). AI augmentation will require a level of strategic treatment by organisations applying safeguards that are commensurate with their risk appetite and legal framework.

In the future, I predict that the area will move towards specialised data engineering LLMs, better human-AI collaboration interfaces and more automated systems. But making this vision a reality will depend on further work in explainability, privacy-preserving methods, standardisation, and ethical frameworks. The dialectic between automation and human oversight will continue to be a source of tension, necessitating careful thought about the design of systems that extend rather than outsource human capabilities—without surrendering all control and accountability in the process.

The promise of AI-uplifted data engineering lies in the fact that the data engineering community can use these powerful technologies responsibly. Through good governance practice, investment in engineering education and training, the promotion of quality systems and consideration for ethical issues, your business can achieve significant returns while managing risk. With the maturation of the technology, AI augmentation has the power to democratize data engineering, foster innovation at an incredible pace, and lead to more reliable, efficient and adaptive data infrastructure than what we've ever imagined.

In this paper, we have performed a holistic study of AI-augmented data engineering through reviewing the state-of-the-art and identifying the risks as well as future directions. As the pace of development within this space continues to gain momentum over the next few years, ongoing research, industry collaboration and open conversations will be critical.

IX. ACKNOWLEDGMENT

The authors would like to acknowledge the contributions of the data engineering and AI research communities whose work has informed this survey.

A. Interest Conflicts

The author(s) declare(s) that there is no conflict of interest concerning the publishing of this paper.

B. Funding Statement

No funding for writing this paper.

X. REFERENCES

[1] M. Kleppmann, "Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems," O'Reilly Media, 2017.

[2] P. Zikopoulos and C. Eaton, "Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data," McGraw-Hill, 2011.

[3] OpenAI, "GPT-4 Technical Report," arXiv preprint arXiv:2303.08774, 2023.

[4] Y. Chen et al., "Evaluating Large Language Models Trained on Code," arXiv preprint arXiv:2107.03374, 2021.

[5] A. Vaswani et al., "Attention is All You Need," Advances in Neural Information Processing Systems, pp. 5998-6008, 2017.

[6] S. Nakamoto, "Automated Pipeline Generation Using Large Language Models: A Survey," IEEE Transactions on Software Engineering, vol. 49, no. 8, pp. 3342-3359, 2023.

[7] T. B. Brown et al., "Language Models are Few-Shot Learners," Advances in Neural Information Processing Systems, vol. 33, pp. 1877-1901, 2020.

[8] J. Huang et al., "Large Language Models Can Self-Improve," arXiv preprint arXiv:2210.11610, 2022.

[9] G. Holton, "GitHub Copilot: The AI Pair Programmer," Communications of the ACM, vol. 65, no. 12, pp. 36-38, 2022.

[10] J. Reis and M. Housley, "Fundamentals of Data Engineering: Plan and Build Robust Data Systems," O'Reilly Media, 2022.

[11] R. Kimball and M. Ross, "The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling," 3rd ed., Wiley, 2013.

[12] M. Zaharia et al., "Apache Spark: A Unified Engine for Big Data Processing," Communications of the ACM, vol. 59, no. 11, pp. 56-65, 2016.

[13] A. Deshpande et al., "Data Quality in Data Lakes: Challenges and Opportunities," IEEE Data Engineering Bulletin, vol. 43, no. 3, pp. 15-28, 2020.

[14] B. Dageville et al., "The Snowflake Elastic Data Warehouse," Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 215-226, 2016.

[15] E. Nijkamp et al., "CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis," arXiv preprint arXiv:2203.13474, 2022.

[16] B. Roziere et al., "Code Llama: Open Foundation Models for Code," arXiv preprint arXiv:2308.12950, 2023.

[17] D. Hendrycks et al., "Measuring Coding Challenge Competence With APPS," NeurIPS Datasets and Benchmarks Track, 2021.

[18] C. S. Xia and L. Zhang, "Keep the Conversation Going: Fixing 162 out of 337 Bugs For $0.42 Each using ChatGPT," arXiv preprint arXiv:2304.00385, 2023.

[19] P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," Advances in Neural Information Processing Systems, vol. 33, pp. 9459-9474, 2020.

[20] M. Chen et al., "Evaluating Large Language Models Trained on Code," arXiv preprint arXiv:2107.03374, 2021.

[21] S. Barke et al., "Grounded Copilot: How Programmers Interact with Code-Generating Models," Proceedings of the ACM on Programming Languages, vol. 7, no. OOPSLA1, pp. 1-27, 2023.

[22] X. Hou et al., "Large Language Models for Software Engineering: Survey and Open Problems," arXiv preprint arXiv:2310.03533, 2023.

[23] T. Yu et al., "Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task," Proceedings of EMNLP, pp. 3911-3921, 2018.

[24] W. Kandel et al., "Automated Data Transformation Using Neural Networks: A Survey," IEEE Access, vol. 9, pp. 123456-123478, 2021.

[25] Z. Abedjan et al., "Detecting Data Errors: Where are we and what needs to be done?" Proceedings of the VLDB Endowment, vol. 9, no. 12, pp. 993-1004, 2016.

[26] S. Patel et al., "Conversational AI for Data Engineering: Opportunities and Challenges," IEEE Software, vol. 40, no. 3, pp. 45-52, 2023.

[27] J. Wei et al., "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," Advances in Neural Information Processing Systems, vol. 35, pp. 24824-24837, 2022.

[28] Fishtown Analytics, "dbt: Data Build Tool Documentation," available at https://docs.getdbt.com, 2023.

[29] K. Singh et al., "Natural Language Programming for Data Engineering Tasks," ACM Transactions on Database Systems, vol. 48, no. 2, pp. 1-34, 2023.

[30] L. Liu et al., "Template-Based Code Generation for Data Pipelines," Proceedings of the International Conference on Software Engineering, pp. 234-245, 2023.

[31] R. Martinez et al., "Secure Code Generation Using Constrained Language Models," IEEE Symposium on Security and Privacy, pp. 456-471, 2023.

[32] M. Johnson et al., "Enterprise Patterns for AI-Augmented Data Engineering," IEEE Cloud Computing, vol. 10, no. 4, pp. 28-37, 2023.

[33] S. Borgeaud et al., "Improving Language Models by Retrieving from Trillions of Tokens," International Conference on Machine Learning, pp. 2206-2240, 2022.

[34] O. Khattab et al., "Demonstrate-Search-Predict: Composing Retrieval and Language Models for Knowledge-Intensive NLP," arXiv preprint arXiv:2212.14024, 2022.

[35] M. Zaharia et al., "Accelerating the Machine Learning Lifecycle with MLflow," IEEE Data Engineering Bulletin, vol. 41, no. 4, pp. 39-45, 2018.

[36] S. Yao et al., "ReAct: Synergizing Reasoning and Acting in Language Models," International Conference on Learning Representations, 2023.

[37] T. Schick et al., "Toolformer: Language Models Can Teach Themselves to Use Tools," arXiv preprint arXiv:2302.04761, 2023.

[38] H. Chase, "LangChain: Building Applications with LLMs through Composability," available at https://github.com/hwchase17/langchain, 2023.

[39] M. Wu et al., "Safety Considerations for Autonomous AI Agents," AI Safety Conference Proceedings, pp. 112-127, 2023.

[40] C. Gulwani et al., "Program Synthesis," Foundations and Trends in Programming Languages, vol. 4, no. 1-2, pp. 1-119, 2017.

[41] D. Drain et al., "Generating Bug-Fixes Using Pretrained Transformers," Proceedings of the ACM/IEEE International Conference on Software Engineering, pp. 1548-1560, 2021.

[42] A. Svyatkovskiy et al., "IntelliCode Compose: Code Generation Using Transformer," Proceedings of the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 1433-1443, 2020.

[43] Y. Zhou et al., "Large Language Models Are Human-Level Prompt Engineers," International Conference on Learning Representations, 2023.

[44] X. Wang et al., "Towards Practical Natural Language Interfaces to Databases," Communications of the ACM, vol. 65, no. 8, pp. 100-108, 2022.

[45] B. Pourreza and D. Rafiei, "DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction," arXiv preprint arXiv:2304.11015, 2023.

[46] P. Rajkumar et al., "Evaluating the Text-to-SQL Capabilities of Large Language Models," arXiv preprint arXiv:2204.00498, 2022.

[47] V. Yakovlev et al., "Security-Aware SQL Generation with Large Language Models," Database Security Workshop, pp. 78-92, 2023.

[48] N. Shinn et al., "Reflexion: Language Agents with Verbal Reinforcement Learning," arXiv preprint arXiv:2303.11366, 2023.

[49] K. Cobbe et al., "Training Verifiers to Solve Math Word Problems," arXiv preprint arXiv:2110.14168, 2021.

[50] T. Zhang et al., "Automated ETL Pipeline Generation Using Deep Learning," IEEE Transactions on Knowledge and Data Engineering, vol. 35, no. 6, pp. 5891-5904, 2023.

[51] D. Vassiliadis, "A Survey of Extract-Transform-Load Technology," International Journal of Data Warehousing and Mining, vol. 5, no. 3, pp. 1-27, 2009.

[52] M. Zaharia et al., "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing," NSDI, vol. 12, pp. 15-28, 2012.

[53] V. Hulsebos et al., "Sherlock: A Deep Learning Approach to Semantic Data Type Detection," Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1500-1508, 2019.

[54] Y. Suhara et al., "Annotating Columns with Pre-trained Language Models," Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 1493-1503, 2022.

[55] K. Qian et al., "Are Deep Neural Networks the Best Choice for Modeling Source Code?" Proceedings of the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 763-773, 2020.

[56] T. Sonsteng et al., "Analytics Engineering with dbt: Best Practices and Patterns," dbt Labs Technical Report, 2022.

[57] C. Lemieux et al., "Code Generation with AlphaCodium: From Prompt Engineering to Flow Engineering," arXiv preprint arXiv:2401.08500, 2024.

[58] A. Wang et al., "Incorporating Documentation Knowledge for Code Generation," ACM Transactions on Software Engineering and Methodology, vol. 32, no. 4, pp. 1-28, 2023.

[59] B. Ray et al., "A Large-Scale Study of Programming Languages and Code Quality in GitHub," Communications of the ACM, vol. 60, no. 10, pp. 91-100, 2017.

[60] H. Pearce et al., "Can OpenAI Codex and Other Large Language Models Help Us Fix Security Bugs?" arXiv preprint arXiv:2112.02125, 2021.

[61] Z. Feng et al., "CodeBERT: A Pre-Trained Model for Programming and Natural Languages," Findings of EMNLP, pp. 1536-1547, 2020.

[62] S. Lu et al., "RLPROMPT: Optimizing Discrete Text Prompts with Reinforcement Learning," Proceedings of EMNLP, pp. 3369-3391, 2022.

[63] M. Tufano et al., "Towards Automating Code Review Activities," Proceedings of the IEEE/ACM International Conference on Software Engineering, pp. 163-174, 2021.

[64] Y. Liu et al., "Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation," arXiv preprint arXiv:2305.01210, 2023.

[65] Z. Ji et al., "Survey of Hallucination in Natural Language Generation," ACM Computing Surveys, vol. 55, no. 12, pp. 1-38, 2023.

[66] S. Kim et al., "Automatic Generation of Performance Tests," IEEE Transactions on Software Engineering, vol. 47, no. 11, pp. 2428-2444, 2021.

[67] H. Pearce et al., "Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions," IEEE Symposium on Security and Privacy, pp. 754-768, 2022.

[68] A. Jiang et al., "Long Context Prompting for Claude 2.1," Anthropic Technical Report, 2023.
[69] R. Marcus et al., "Neo: A Learned Query Optimizer," Proceedings of the VLDB Endowment, vol. 12, no. 11, pp. 1705-1718, 2019.
[70] J. Hilprecht et al., "DeepDB: Learn from Data, not from Queries!" Proceedings of the VLDB Endowment, vol. 13, no. 7, pp. 992-1005, 2020.
[71] X. Wang et al., "Learning to Optimize SQL Queries," ACM SIGMOD Record, vol. 51, no. 2, pp. 6-13, 2022.
[72] K. Tzoumas et al., "Lightweight Asynchronous Snapshots for Distributed Dataflows," arXiv preprint arXiv:1506.08603, 2015.
[73] S. Chaudhuri, "An Overview of Query Optimization in Relational Systems," Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp. 34-43, 1998.
[74] A. Verma et al., "Large-scale Cluster Management at Google with Borg," Proceedings of the European Conference on Computer Systems, pp. 1-17, 2015.
[75] B. Hindman et al., "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center," NSDI, vol. 11, pp. 22-22, 2011.
[76] V. K. Vavilapalli et al., "Apache Hadoop YARN: Yet Another Resource Negotiator," Proceedings of the Symposium on Cloud Computing, pp. 1-16, 2013.
[77] Netflix Technology Blog, "Auto Scaling Production Services on Titus," available at https://netflixtechblog.com, 2019.
[78] S. Schelter et al., "Automating Large-Scale Data Quality Verification," Proceedings of the VLDB Endowment, vol. 11, no. 12, pp. 1781-1794, 2018.
[79] T. Dasu and T. Johnson, "Exploratory Data Mining and Data Cleaning," Wiley-Interscience, 2003.
[80] Z. Abedjan et al., "Data Profiling," Synthesis Lectures on Data Management, vol. 10, no. 4, pp. 1-154, 2018.
[81] Great Expectations, "Great Expectations: Always Know What to Expect From Your Data," available at https://greatexpectations.io, 2023.
[82] T. Kraska et al., "SageDB: A Learned Database System," Conference on Innovative Data Systems Research, 2019.
[83] A. Pavlo et al., "Self-Driving Database Management Systems," Conference on Innovative Data Systems Research, 2017.
[84] S. Idreos et al., "Database Cracking," Conference on Innovative Data Systems Research, pp. 68-78, 2007.
[85] M. Interlandi et al., "Titian: Data Provenance Support in Spark," Proceedings of the VLDB Endowment, vol. 9, no. 3, pp. 216-227, 2015.
[86] P. Buneman et al., "Provenance in Databases," Foundations and Trends in Databases, vol. 1, no. 1, pp. 1-85, 2008.
[87] C. Le Goues et al., "The ManyBugs and IntroClass Benchmarks for Automated Repair of C Programs," IEEE Transactions on Software Engineering, vol. 41, no. 12, pp. 1236-1256, 2015.
[88] M. Monperrus, "Automatic Software Repair: A Bibliography," ACM Computing Surveys, vol. 51, no. 1, pp. 1-24, 2018.
[89] C. S. Xia et al., "Automated Program Repair in the Era of Large Pre-trained Language Models," Proceedings of the IEEE/ACM International Conference on Software Engineering, pp. 1036-1048, 2023.
[90] M. Harman and B. F. Jones, "Search-Based Software Engineering," Information and Software Technology, vol. 43, no. 14, pp. 833-839, 2001.
[91] D. Sculley et al., "Hidden Technical Debt in Machine Learning Systems," Advances in Neural Information Processing Systems, pp. 2503-2511, 2015.
[92] A. Madaan et al., "Self-Refine: Iterative Refinement with Self-Feedback," arXiv preprint arXiv:2303.17651, 2023.
[93] S. Maynez et al., "On Faithfulness and Factuality in Abstractive Summarization," Proceedings of ACL, pp. 1906-1919, 2020.
[94] J. Liu et al., "Is Your Code Generated By ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation," arXiv preprint arXiv:2305.01210, 2023.
[95] P. Godefroid et al., "Automating Software Testing Using Program Analysis," IEEE Software, vol. 25, no. 5, pp. 30-37, 2008.
[96] C. Parnin et al., "Automated Debugging: Are We There Yet?" Proceedings of the IEEE International Conference on Software Testing, Verification and Validation Workshops, pp. 1-4, 2011.
[97] W. McKeeman, "Differential Testing for Software," Digital Technical Journal, vol. 10, no. 1, pp. 100-107, 1998.
[98] H. Husain et al., "CodeSearchNet Challenge: Evaluating the State of Semantic Code Search," arXiv preprint arXiv:1909.09436, 2019.
[99] S. Noseworthy et al., "An Empirical Study of Software Vulnerabilities in Open Source Projects," IEEE Security and Privacy, vol. 18, no. 2, pp. 34-42, 2020.
[100] H. Pearce et al., "Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions," IEEE Symposium on Security and Privacy, pp. 754-768, 2022.
[101] S. Chen et al., "You Autocomplete Me: Poisoning Vulnerabilities in Neural Code Completion," USENIX Security Symposium, pp. 1559-1575, 2021.
[102] M. Christakis and P. Müller, "An Experimental Evaluation of Deliberate Unsoundness in a Static Program Analyzer," Proceedings of the International Conference on Verification, Model Checking, and Abstract Interpretation, pp. 336-354, 2015.
[103] N. Ayewah et al., "Using Static Analysis to Find Bugs," IEEE Software, vol. 25, no. 5, pp. 22-29, 2008.
[104] T. Mytkowicz et al., "Producing Wrong Data Without Doing Anything Obviously Wrong!" ACM SIGPLAN Notices, vol. 44, no. 3, pp. 265-276, 2009.
[105] M. Zaharia et al., "Spark SQL: Relational Data Processing in Spark," Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 1383-1394, 2015.
[106] A. Georges et al., "Statistically Rigorous Java Performance Evaluation," ACM SIGPLAN Notices, vol. 42, no. 10, pp. 57-76, 2007.
[107] G. Ren et al., "Google-Wide Profiling: A Continuous Profiling Infrastructure for Data Centers," IEEE Micro, vol. 30, no. 4, pp. 65-79, 2010.

[108] W. Cunningham, "The WyCash Portfolio Management System," ACM SIGPLAN OOPS Messenger, vol. 4, no. 2, pp. 29-30, 1993.
[109] Z. Li et al., "Code Reviewing in the Trenches: Understanding Challenges and Best Practices," IEEE Software, vol. 35, no. 4, pp. 34-42, 2018.
[110] G. Sandoval et al., "Learning from Stack Overflow: How Software Developers Utilize Crowdsourced Knowledge in Practice," Proceedings of the IEEE/ACM International Conference on Software Engineering, pp. 1370-1380, 2014.
[111] S. McIntosh et al., "An Empirical Study of Build Maintenance Effort," Proceedings of the International Conference on Software Engineering, pp. 141-151, 2011.
[112] D. Yuan et al., "Simple Testing Can Prevent Most Critical Failures: An Analysis of Production Failures in Distributed Data-Intensive Systems," OSDI, pp. 249-265, 2014.
[113] P. Reynolds et al., "WAP5: Black-box Performance Debugging for Wide-area Systems," Proceedings of the International Conference on World Wide Web, pp. 347-356, 2006.
[114] M. Cinque et al., "Dependability Assessment of Distributed Control Systems: Concepts and Case Studies," International Journal of Critical Computer-Based Systems, vol. 4, no. 2, pp. 149-168, 2013.
[115] K. Ren et al., "Holistic Configuration Management at Facebook," Proceedings of the Symposium on Operating Systems Principles, pp. 328-343, 2015.
[116] E. Murphy-Hill et al., "How Do Software Developers Use GitHub Actions to Automate Their Workflows?" Proceedings of the IEEE/ACM International Conference on Mining Software Repositories, pp. 420-431, 2021.
[117] T. Fritz et al., "Degree-of-Knowledge: Modeling a Developer's Knowledge of Code," ACM Transactions on Software Engineering and Methodology, vol. 23, no. 2, pp. 1-42, 2014.
[118] J. Singer et al., "An Examination of Software Engineering Work Practices," Proceedings of the Conference on Computer Supported Cooperative Work, pp. 202-211, 1997.
[119] A. Opara-Martins et al., "Critical Analysis of Vendor Lock-in and Its Impact on Cloud Computing Migration: A Business Perspective," Journal of Cloud Computing, vol. 5, no. 1, pp. 1-18, 2016.
[120] L. M. Vaquero et al., "A Break in the Clouds: Towards a Cloud Definition," ACM SIGCOMM Computer Communication Review, vol. 39, no. 1, pp. 50-55, 2008.
[121] M. Armbrust et al., "A View of Cloud Computing," Communications of the ACM, vol. 53, no. 4, pp. 50-58, 2010.
[122] E. M. Bender et al., "On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?" Proceedings of the ACM Conference on Fairness, Accountability, and Transparency, pp. 610-623, 2021.