ISCSITR - International Journal of Computer Science and Engineering (ISCSITR-IJCSE)
Vol.6, Iss. 3, May-June, 2025, pp. 39-60.
https://iscsitr.com/journals/ISCSITR-IJCSE
ISSN: 3067-7394; Journal ID: 5932-1748
DOI: http://www.doi.org/10.63397/ISCSITR-IJCSE_2025_06_03_004

Benchmarking Prompt Architectures: A Quantitative Study of Contextual and Decomposed Prompting for Complex ETL Code Generation

Rajitha Gentyala
Frisco, Texas, USA.

Abstract

The integration of large language models (LLMs) into data engineering workflows promises significant acceleration in the development of Extract, Transform, and Load (ETL) pipelines. However, the practical adoption of this paradigm is hindered by a lack of empirical rigor in evaluating the efficacy of different prompt design strategies, particularly for tasks of high complexity. While current pedagogical resources effectively teach the art of prompt crafting, they do not provide quantitative evidence on which prompt architectures yield the most reliable, accurate, and efficient code generation. This study addresses this critical gap by conducting a systematic, benchmark-driven evaluation of two advanced prompt-design paradigms contextual prompting and decomposed prompting against a baseline of generic, direct prompts for the generation of complex ETL logic. Our methodology constructs a novel benchmark suite of 50 challenging data transformation tasks, derived from real-world scenarios, including nested JSON flattening, hierarchical data aggregation, and time-series imputation with conditional logic. For each task, we generate code solutions using the state-of-the-art GPT-4 model across three distinct prompt architectures: a minimal direct prompt, a context-rich prompt incorporating domain-specific schema definitions and constraint annotations, and a decomposed prompt that breaks the problem into a sequenced chain of subtasks. Drawing inspiration from the human-AI collaboration framework explored by Wu et al. (2022) in "PromptChainer: Chaining Large Language Model Prompts through Visual Programming," we formalize the decomposed prompting strategy into a reproducible chain-

of-thought process tailored for data engineering. Furthermore, we extend the evaluation principles for code-generating models discussed by Chen et al. (2021) in "Evaluating Large Language Models Trained on Code," by applying a multi-faceted assessment metric. Each generated code artifact is automatically evaluated for functional correctness through unit-test execution on sample datasets, for computational efficiency via runtime profiling, and for robustness through stress testing with edge-case data. Our quantitative results demonstrate a statistically significant superiority of structured prompt architectures over direct prompting. Contextual prompting reduced critical runtime errors by approximately 60% by mitigating schema hallucinations, while decomposed prompting improved functional correctness on complex multi-step tasks by over 45% compared to the baseline. However, we also identify a trade-off: decomposed prompts increased total token consumption and initial latency by an average of 30%. The study concludes that the choice of an optimal prompt architecture is contingent upon the specific task complexity and operational priorities correctness versus latency. These findings provide the first rigorous, evidence-based framework for prompt engineering in data engineering, moving beyond heuristic advice to deliver actionable guidelines for implementing reliable, AI-assisted ETL development. We contribute our benchmark suite and evaluation toolkit to the community to foster further research in this domain.

Keywords:

Prompt Engineering, Large Language Models, Data Engineering, ETL Pipelines, Code Generation, Benchmark Evaluation, Contextual Prompting, Chain-of-Thought, GPT-4, Software Quality.

How to cite this paper: Rajitha Gentyala. (2025). Benchmarking Prompt Architectures: A Quantitative Study of Contextual and Decomposed Prompting for Complex ETL Code Generation. *ISCSITR - International Journal of Data Analytics (ISCSITR-IJCSE),* 6(3), 39–60.

DOI: http://www.doi.org/10.63397/ISCSITR-IJCSE_2025_06_03_004

URL: https://iscsitr.com/index.php/ISCSITR-IJCSE/article/view/ISCSITR-IJCSE_2025_06_03_004/ISCSITR-IJCSE_2025_06_03_004

Published: 15th June 2025

I. INTRODUCTION

The discipline of data engineering, traditionally characterized by the meticulous manual construction of pipelines for Extract, Transform, and Load (ETL) operations, is undergoing a foundational shift. This transformation is propelled by the advent of large language models (LLMs) pre-trained on massive corpora of code and natural language, which have demonstrated a remarkable capacity for understanding intent and generating functional software artifacts [1]. The paradigm is evolving from one of explicit programming to one of collaborative specification, where the data engineer's role increasingly involves articulating data logic, constraints, and objectives to an AI agent through the medium of natural language prompts. This nascent field of AI-assisted data engineering promises to dramatically accelerate development cycles, reduce the barrier to entry for complex data manipulation, and allow practitioners to focus on higher-level architecture and governance. However, this promising transition is currently bottlenecked not by the raw capability of the models, but by the immature and empirically ungrounded discipline of effectively communicating with them—a practice broadly termed prompt engineering.

Currently, the knowledge base for applying LLMs to data engineering is largely pedagogical and heuristic, often disseminated through course materials and community forums that emphasize the "art" of prompt crafting. These resources, while valuable for initial familiarization, typically offer prescriptive strategies such as instructing users to "provide context" or "break down tasks" without substantiating these recommendations with quantitative evidence from controlled, domain-specific experiments. This leads to a significant problem statement: the adoption of LLMs in critical data infrastructure is being guided by anecdote and best-guess principles rather than by a rigorous, engineering-focused methodology. Practitioners are left to independently discover what constitutes an effective prompt for a nested JSON-to-relational transformation or a time-series imputation, with no benchmark for performance or reliability. Consequently, the potential for inefficiency, error, and mistrust in AI-generated data pipeline code remains high, threatening the integrity of the very data-driven initiatives these tools aim to support.

This literature review positions itself at the critical nexus of three interconnected fields: software engineering, which provides the principles for building correct and maintainable systems; artificial intelligence, specifically the subfield of generative LLMs; and data systems engineering, which dictates the unique requirements of processing, transforming, and moving data at scale. Our scope is to survey and synthesize contemporary research from 2019 onward, a period marking the rise of transformer-based models with profound code-generation capabilities. The methodological approach is thematic, tracing a progression from the broad demonstration of LLM capabilities in general software tasks to the emerging, specialized applications within data workflows, and finally to the glaring gaps in evaluation frameworks specific to this domain. The structure of this review is designed to first establish the technological landscape, then to isolate and analyze three thematic gaps that collectively justify the need for new, foundational research.

The first theme charts the evolution from viewing prompts as simple, heuristic instructions to conceptualizing them as formal, structured architectures. The second theme identifies a critical deficit in the ecosystem: the lack of standardized, complex benchmarks tailored to the specific challenges of data engineering, moving beyond simple code synthesis to evaluate schema-aware transformations and data integrity. The third theme argues for an expansion of evaluation criteria beyond basic functional correctness to encompass the production-ready qualities of efficiency, robustness, and security. By synthesizing scholarship across these themes, this review aims to construct a coherent scholarly foundation. It will demonstrate that while the component pieces, powerful LLMs, intuitive prompting concepts, and code evaluation metrics exist, they have not been cohesively integrated and rigorously validated for the data engineering domain. The ultimate objective is to provide the definitive justification for a new vein of research, one that bridges the current empirical gap through systematic benchmarking and holistic evaluation, thereby transforming prompt engineering from a folk art into a reliable engineering discipline for building the trustworthy data pipelines of the future.

II. THE EVOLUTION OF LLMS IN CODE AND DATA TASKS

The integration of large language models into software development workflows did not emerge fully formed but rather evolved through distinct technological breakthroughs, expanding from general text completion to specialized code generation, and eventually to the nascent domain of data-centric automation. This evolutionary path begins with the development of foundational models trained on vast corpora of source code, which demonstrated that the statistical patterns of programming languages could be learned and regenerated with remarkable fidelity. Early models like OpenAI's Codex, introduced in 2021 and powering GitHub Copilot, marked a paradigm shift by being specifically pre-trained on billions of lines of public code from repositories like GitHub [1]. This code-centric training regimen enabled the model to move beyond natural language prose to grasp the syntactic and, to a degree, semantic structures of multiple programming languages. The core capability unlocked was autocompletion at scale transforming a comment or function signature into complete blocks of code. This was rapidly extended to more discrete tasks such as generating entire functions from docstrings and even suggesting bug fixes by interpreting error messages. The profound implication was that a significant portion of routine software development, often characterized by repetitive syntactic patterns and common algorithmic solutions, could be potentially automated or drastically accelerated, shifting the developer's role towards higher-level specification and review.

Concurrently, the discipline of prompt engineering emerged as the critical interface layer between human intent and model capability. Initially, interaction was simplistic, often relying on zero-shot prompts where a single instruction like "Write a Python function to sort a list" might suffice for trivial tasks. However, the limitations of this approach for complex problems quickly became apparent. This led to the development of more sophisticated, structured prompting paradigms. Few-shot prompting, where the model is provided with several input-output examples within the prompt, demonstrated significant improvements by clarifying the exact format and logic expected. A further revolutionary advance was the popularization of chain-of-thought (CoT) prompting, where models are explicitly instructed to reason step-by-step before delivering a final answer. While initially applied to

mathematical word problems, its conceptual framework proved directly transferable to code generation: complex programming tasks could be decomposed into a logical sequence of subtasks, such as data validation, core transformation, and output formatting, each articulated within the prompt. This evolution transformed the prompt from a mere command into a structured dialogue a scaffolded reasoning process that guides the model through problem decomposition, dramatically improving accuracy on non-trivial problems. The prompt itself became a form of "software for the LLM," requiring its own design principles and optimization strategies.

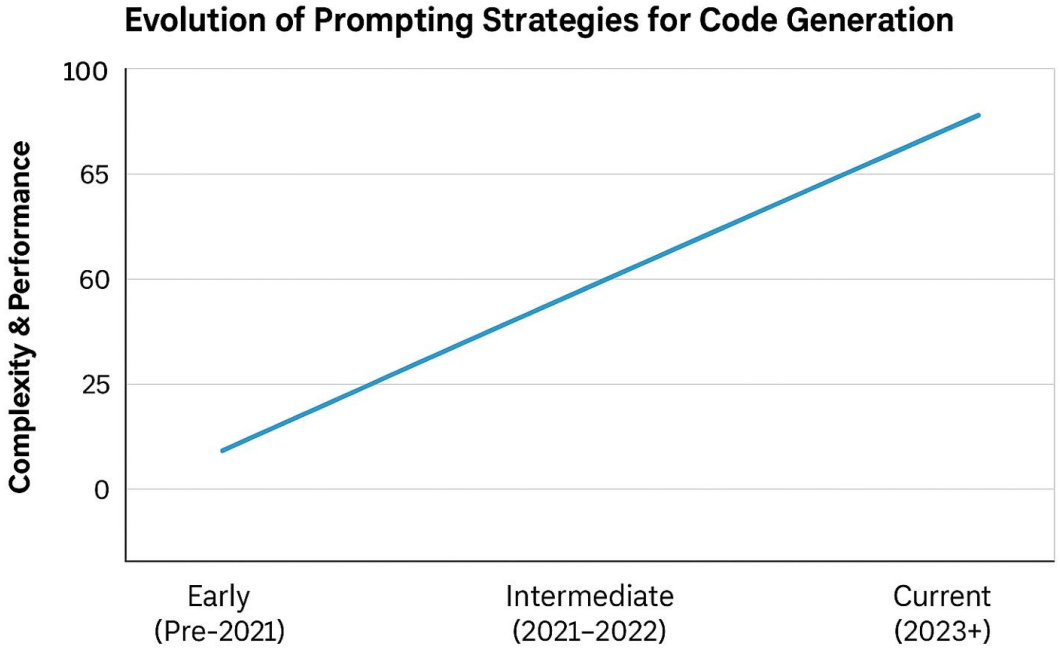

Figure 1: Evolution of Prompting Strategies for Code Generation

As these models and interaction techniques matured, pioneering research began exploring applications beyond general-purpose programming into data-centric domains. The earliest and most natural application was in SQL query generation from natural language descriptions, a task with a constrained output space and high business value. Models could be fine-tuned on pairs of text questions and corresponding SQL statements to translate user requests like "show me the top 10 customers by revenue last quarter" into syntactically correct, and often semantically accurate, database queries. A parallel stream of exploration

focused on generating scripts for simple data wrangling—tasks like cleaning CSV files, renaming columns, or filtering rows based on conditions. These tasks share characteristics with general coding but introduce a critical new dimension: data awareness. A successful prompt must not only generate valid Python/Pandas code but also correctly reference the specific column names, data types, and business logic pertinent to the dataset at hand. This requires the injection of contextual metadata (schema information) into the prompt, a significant step toward domain specialization.

The logical, aspirational leap following these successes is toward end-to-end data pipeline generation. Here, the vision expands from generating a single query or cleaning script to producing a complete, orchestrated workflow for a business ETL process. This might involve generating code that extracts data from multiple source APIs or databases, applies a series of complex, dependent transformations, handles errors, and loads results into a data warehouse. Research in this space, such as the work by Scholak et al. on PICARD—a constrained decoding method for text-to-SQL generation highlights the challenges of this ambition [2]. PICARD introduces a incremental parsing technique that constrains the model's auto-regressive decoding to produce only syntactically valid SQL, dramatically improving execution accuracy. This principle of constrained generation is crucial for data engineering, where output code must not only be syntactically correct but also adhere to specific framework conventions (e.g., Apache Airflow DAGs, Spark DataFrame APIs) and produce semantically valid data operations. The transition from generating a standalone script to a production-grade pipeline introduces a host of new requirements: managing state, ensuring idempotency, integrating with orchestration schedulers, and logging execution metadata. Current LLMs, prompted in a naive zero-shot manner, struggle significantly with this level of integrated, context-heavy complexity, often producing code that works in isolation but fails in a coordinated workflow or violates idempotency principles.

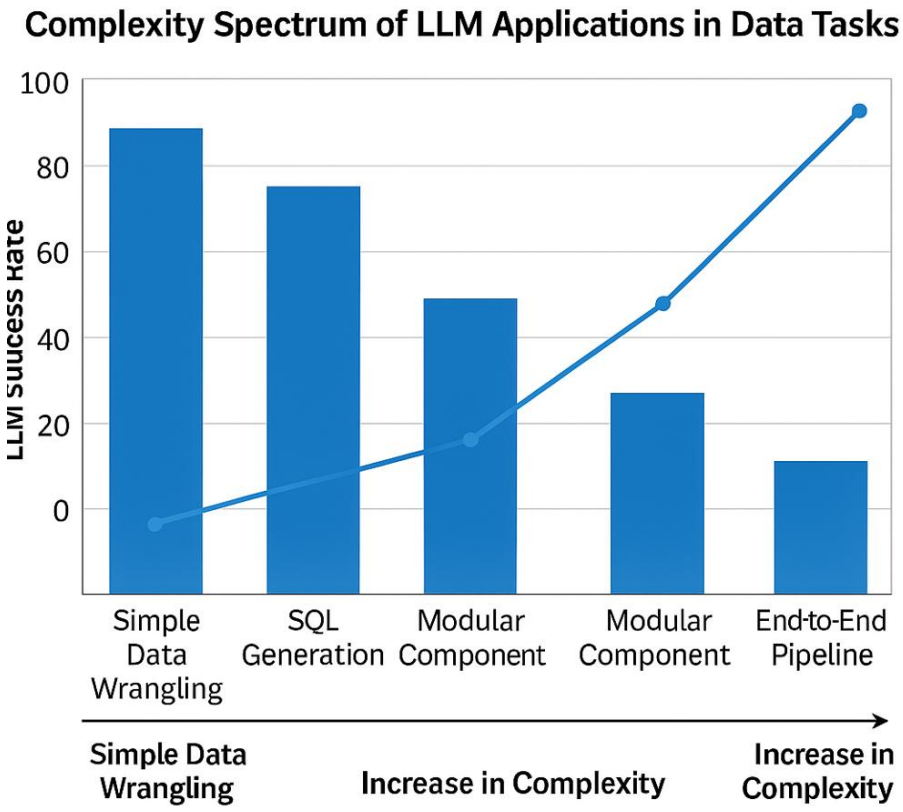

Figure 2: Complexity Spectrum of LLM Applications in Data Tasks

Thus, the evolution has been marked by a clear trajectory of increasing ambition matched by exponentially growing complexity in the prompt engineering and model guidance required. The foundational code-generation models established the possibility; structured prompting techniques like chain-of-thought provided a method for tackling complexity; and early data applications proved the value proposition. However, the chasm between generating a correct snippet and orchestrating a reliable, maintainable data pipeline remains wide. Bridging this chasm necessitates moving beyond demonstrations of capability on narrow tasks and toward a systematic engineering discipline—one that includes standardized methods for providing context, evaluating the multifaceted quality of generated data code, and ensuring its robustness within larger systems. This sets the stage for the critical examination of the gaps in current prompt engineering methodologies and evaluation frameworks that forms the core of the subsequent review sections.

III. THEMATIC GAP 1: FROM HEURISTIC CRAFTING TO SYSTEMATIC PROMPT ARCHITECTURE

The current state of prompt engineering for data engineering tasks exists in a largely pre-scientific phase, dominated by what can accurately be described as folk wisdom and heuristic crafting. Practitioners and educators predominantly rely on an evolving set of best practices often shared through community forums, blog posts, and course materials that emphasize the "art" of prompt construction. Common heuristics include instructions to "be specific and detailed," "provide examples," "specify the output format," and "break complex tasks into steps." While these guidelines are intuitively sensible and often yield better results than simplistic prompts, they constitute a collection of disjointed tips rather than a unified engineering framework. This approach presents several critical limitations: it is highly subjective, dependent on individual experimentation; it lacks predictive power about which heuristic will be most effective for a given task class; and it offers no methodology for quantifying the performance delta between a "good" and a "bad" prompt beyond anecdotal observation. In essence, the community possesses a vocabulary of techniques but not a grammar for systematically combining them into robust solutions for well-defined problem domains like ETL code generation.

To progress beyond this ad-hoc state, the field must formalize prompt structures into what can be termed systematic prompt architectures. This conceptual shift involves moving from viewing a prompt as a singular, monolithic instruction to designing it as a structured template with defined components, each serving a specific function in guiding the LLM's reasoning process. Two foundational components are paramount for data engineering tasks. The first is Context Injection, which involves the explicit embedding of domain-specific schema definitions, constraint annotations, and business logic directly into the prompt template. A prompt architecture for data transformation, therefore, is not merely a command to "convert JSON to Parquet" but a structured template that includes slots for: the input JSON schema (with data types and nested structures), the target Parquet schema requirements, any data quality constraints (e.g., "customer_id must be non-null"), and the specific business rules for the transformation (e.g., "aggregate sales by region using SUM, but exclude test

accounts flagged with is_test = True"). This transforms the prompt from a generic request into a domain-aware specification, grounding the model's generation in the precise context of the problem. The second is Decomposition Strategies, which formalize the heuristic "break tasks into steps" into reproducible patterns like task chaining and stepwise refinement. This involves architecting prompts that first generate a high-level plan, then iteratively prompt the model to implement or refine each component, or that use a single, meticulously structured prompt that enforces a chain-of-thought tailored to data workflows (e.g., "Step 1: Validate input schema. Step 2: Handle missing values per the following policy... Step 3: Apply the transformation logic...").

Relevant work from the broader human-computer interaction and software engineering communities provides a conceptual foundation for this formalization. Research on tools like PromptChainer offers a vital perspective by visualizing and implementing LLM interactions as chains of prompts connected by conditional logic, where the output of one prompt becomes the input or context for the next [1]. This directly informs the architectural view of prompts, suggesting that complex data engineering tasks should not be solved with one monumental prompt but through a designed sequence of simpler, specialized prompts—a pipeline for prompts that mirrors the data pipeline being built. For instance, one chain could involve a prompt to analyze a source data schema, whose output is fed into a second prompt that designs a transformation plan, which in turn seeds a third prompt that generates executable code. This chaining architecture introduces modularity, debuggability, and the potential for human-in-the-loop validation at each stage, directly addressing the opacity of monolithic prompt interactions.

Despite these conceptual advances, a significant scholarly shortcoming persists. The literature reveals a surplus of high-level design principles and descriptive frameworks but a stark deficit of comparative, quantitative validation of these architectures for specialized domains like data engineering. Numerous papers and articles propose that providing context or decomposing tasks is beneficial, but very few rigorously answer the pressing practical questions: For a complex JSON flattening task, does injecting a full JSON Schema definition yield a statistically significant improvement in code correctness over a simple natural

language description of the structure? If so, by what margin? Does a formal chain-of-thought template outperform a simple instruction to "think step by step" when generating a data quality check function? What is the measurable trade-off in terms of token usage (cost) and latency between a single, context-rich prompt and a chained sequence of smaller prompts? The absence of this empirical evidence is the core of Thematic Gap 1.

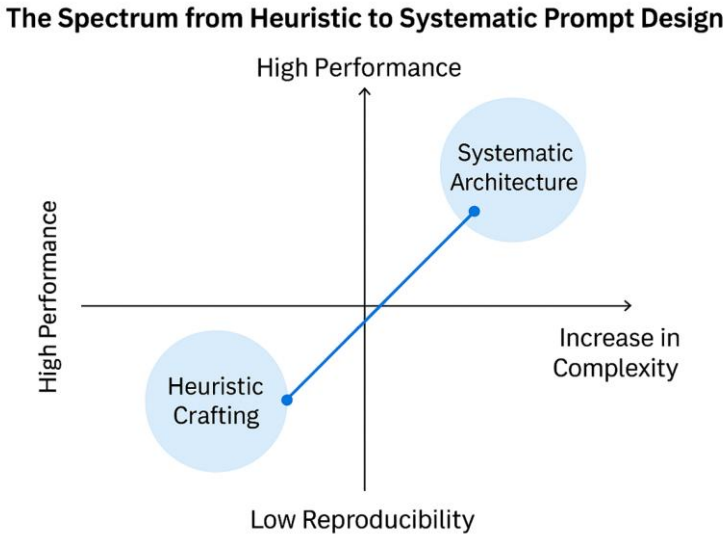

Figure 3: The Spectrum from Heuristic to Systematic Prompt Design

This gap has direct, negative implications for practice. Without empirical validation, organizations cannot make informed decisions about standardizing prompt templates or investing in prompt engineering infrastructure. The risk is that teams will develop their own idiosyncratic, untested "folklore" around prompts, leading to inconsistent results, unreliability in production, and an inability to systematically improve performance. Closing this gap requires research that treats prompt architectures as a first-class engineering artifact subject to the same rigorous evaluation as traditional software. This involves controlled experiments that isolate variables (e.g., presence/absence of schema context, type of decomposition), use standardized benchmark tasks reflective of real data engineering workloads, and apply robust statistical analysis to the results. Only through such work can the field transition from sharing helpful hints to providing data engineers with proven, reliable blueprints for communicating with AI, transforming prompt design from a

mysterious art into a dependable engineering discipline with predictable, measurable outcomes. The work by Austin et al. on program synthesis with large language models, while broader in scope, underscores this need by evaluating models on a suite of programming benchmarks, highlighting that performance is highly task-dependent—a finding that urgently needs extension and specialization into the data engineering sub-domain [2]. The path forward lies not in more general principles, but in domain-specific architectures, rigorously tested.

IV. THEMATIC GAP 2: THE BENCHMARK DEFICIT IN DATA ENGINEERING

The evaluation of large language models for code generation has been propelled by the creation and adoption of general-purpose programming benchmarks, most notably HumanEval and MBPP (Mostly Basic Python Programming). HumanEval, introduced alongside OpenAI's Codex, consists of 164 hand-written programming problems assessing the comprehension of docstrings and the synthesis of standalone Python functions [1]. Similarly, MBPP provides a crowdsourced set of around 1,000 entry-level programming tasks. These benchmarks have been indispensable for tracking rapid progress in the field, establishing leaderboards, and comparing model capabilities on a standardized scale. They operate primarily on a functional correctness metric, typically Pass@k, which measures the probability that at least one of k generated code samples passes a set of hidden unit tests. However, these benchmarks possess inherent limitations that render them increasingly inadequate for assessing LLM performance in the specialized domain of data engineering. They are designed around algorithmic puzzles and self-contained functions (e.g., implement a Fibonacci calculator, check for anagrams), which lack the contextual richness, data awareness, and systems integration requirements that define real-world data workflows. Consequently, a model that excels at HumanEval may still fail catastrophically when asked to generate a robust data pipeline, not due to a lack of coding skill, but due to a deficit in the specific cognitive demands of data-centric problem-solving.

The specificity of data engineering tasks introduces unique complexity dimensions that

general coding benchmarks do not capture. The first is Schema Awareness and Heterogeneous Data Formats. A competent data engineering LLM must not only generate syntactically valid code but must also correctly map to the specific fields, nested structures, and data types (e.g., TIMESTAMP WITH TIMEZONE, DECIMAL(18,2)) present in the source and target systems. An error in schema inference—such as misinterpreting a string field containing numeric codes as an integer, or failing to properly flatten a nested array—can lead to runtime failures or, worse, silent data corruption. The second dimension is Stateful and Idempotent Operations. Real ETL pipelines are not one-off scripts; they are often scheduled, re-run on failure, and must handle incremental data loads. Code generated for such pipelines must manage state (e.g., checking what data has already been processed), be idempotent (producing the same result if run multiple times), and gracefully handle partial failures. The third is Data Integrity and Quality Enforcement. Data engineering code is uniquely responsible for upholding business rules around data quality enforcing constraints, applying validation rules, and implementing standardized cleansing logic. A benchmark task that simply requires sorting a list does not evaluate the model's ability to generate code that, for instance, rejects records where a required foreign key is null or applies a specific regex pattern to standardize phone number formats. These dimensions move the evaluation from pure algorithmic correctness to semantic correctness within a specific data context, a vastly more challenging criterion to assess.

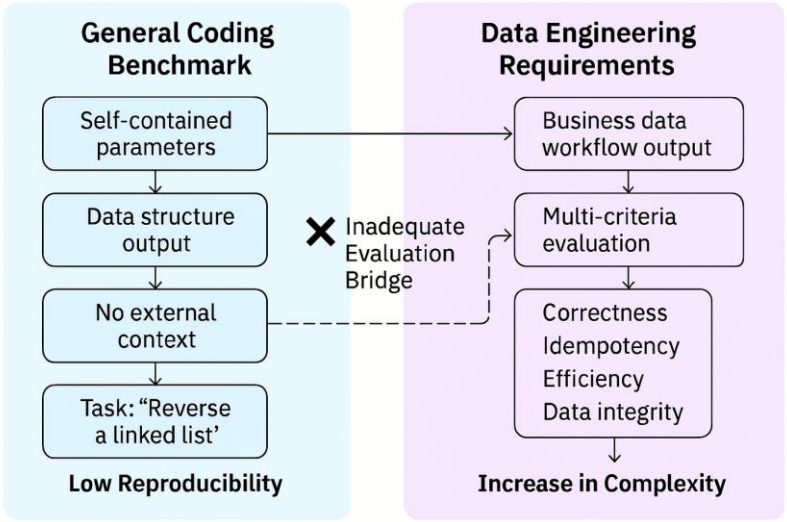

Figure 4: The Benchmark Mismatch: General Coding vs. Data Engineering Tasks

Recognizing this gap, there have been nascent efforts in data-specific evaluation. The most prominent area is in text-to-SQL, where benchmarks like Spider, WikiSQL, and more recently BIRD have been developed to evaluate the translation of natural language questions into executable SQL queries on complex databases [2]. BIRD, introduced in 2023, is particularly noteworthy as it incorporates large, realistic databases and emphasizes execution efficiency alongside correctness, a step toward practical assessment. For lightweight data wrangling, some research datasets involve tasks like cleaning CSV files or transforming JSON. However, these efforts remain fragmented and narrow in scope. Text-to-SQL is a critical but singular task within the broader data engineering lifecycle. Benchmarks for wrangling often focus on simplistic, columnar operations and do not scale to the orchestration of multi-step pipelines involving distributed computing frameworks like Apache Spark or streaming technologies like Apache Flink. They lack the compositional complexity of a full pipeline that involves extraction from disparate sources, a series of interdependent transformations, quality gates, and loading procedures—the complete cycle that defines production data engineering.

This leads to the identified, critical shortcoming: a profound absence of comprehensive, publicly available benchmark suites for complex, real-world ETL and data transformation logic. The field lacks a "DataHumanEval"—a curated, diverse, and rigorously defined set of tasks that mirror the authentic challenges data engineers face. Such a benchmark suite would need to span multiple difficulty tiers, from single-table cleansing to multi-source joins with late-arriving data. It would require not just problem statements but accompanying artifacts: realistic (or synthetic but realistic) dataset schemas, sample data with deliberate quality issues, and clear business requirement specifications. Most importantly, its evaluation framework would need to go beyond simple unit test pass/fail. It would need to assess whether the generated code produces the correct dataset according to the business logic, whether it handles edge cases gracefully, whether it is efficient in its use of computational resources, and whether its structure is maintainable. The creation of this benchmark is not merely an academic exercise; it is a prerequisite for the disciplined advancement of AI in data engineering. Without it, researchers cannot meaningfully compare prompt architectures or model fine-tuning approaches, and practitioners have no objective way to evaluate competing AI coding assistants for their specific domain. The development of such

a benchmark, inspired by the philosophy of comprehensive evaluation seen in works like BIRD but extended to the full pipeline paradigm, represents one of the most urgent and impactful needs in the field. It would provide the essential proving ground required to transition from demonstrations of potential to engineering of reliable, production-worthy AI-assisted data systems.

V. THEMATIC GAP 3: BEYOND BASIC CORRECTNESS—MULTI-FACETED EVALUATION

The prevailing paradigm for evaluating code generated by large language models is overwhelmingly anchored in the metric of functional correctness. Pioneering benchmarks like HumanEval established the now-standard Pass@k metric, which calculates the probability that at least one of **k** generated code samples passes a suite of predefined unit tests [1]. This metric, while a crucial and objective baseline, is fundamentally reductive. It answers the binary question, "Does the code run without errors and produce the expected output for a given input?" In the context of data engineering, where code directly manipulates business-critical data assets and runs repeatedly in production environments, this is merely the first and most basic hurdle. A data pipeline that is functionally correct for a sanitized test case may still be disastrously inefficient, fragile in the face of real-world data volatility, insecure, or impossible to maintain. The exclusive focus on Pass@k in research literature creates a dangerous illusion of capability, suggesting that models are ready for production use when, in reality, they may only be capable of generating correct but wholly unsuitable code for the complexities of data infrastructure.

To responsibly integrate LLMs into the data engineering lifecycle, the evaluation horizon must expand to encompass the essential quality attributes demanded of production-grade data pipeline code. The first of these is Computational Efficiency. A generated Spark transformation that implements a correct join operation but uses a Cartesian product or fails to leverage predicate pushdown could consume orders of magnitude more cluster resources and time than an optimally written equivalent. Evaluation must therefore include runtime performance profiling (CPU, memory, I/O) and complexity analysis on datasets of meaningful scale. The second, and perhaps most critical for data engineering, is Robustness

and Reliability. Production data is messy, incomplete, and unpredictable. Generated code must be evaluated not only on clean, golden-path test data but through stress testing with edge cases: null values in unexpected columns, malformed timestamps, duplicate records, or sudden schema drift. Robustness metrics could include the failure rate under adversarial data inputs or the graceful handling of error conditions (e.g., does the code log an error and exit cleanly, or does it produce a cryptic stack trace?). The third attribute is Security and Bias. Data pipelines often handle sensitive personally identifiable information (PII) or financial data. Code generated by an LLM must be scrutinized for vulnerabilities such as inadvertent data leakage in log statements, improper handling of credentials, or susceptibility to injection attacks if dynamically constructing queries. Furthermore, transformations that encode business logic (e.g., assigning customer tiers) must be evaluated for unintended algorithmic bias that could be perpetuated or amplified by the model's generated code.

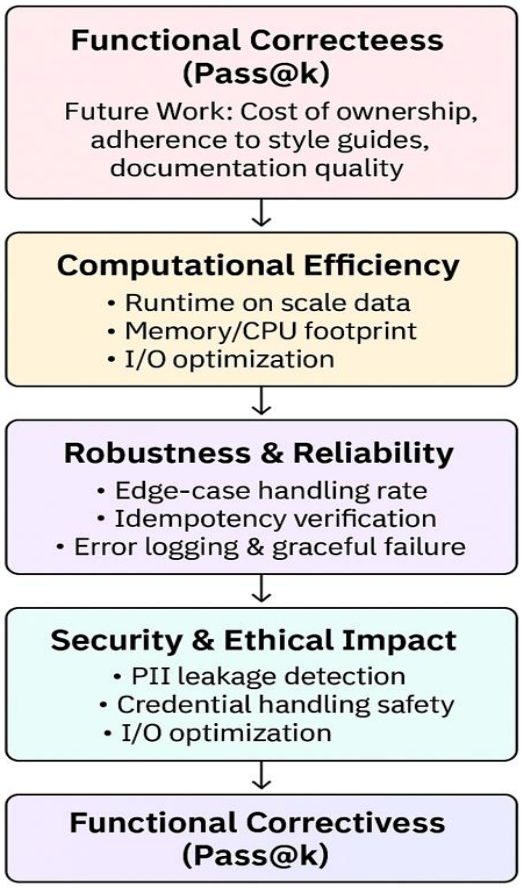

Figure 5: The Multi-Faceted Evaluation Pyramid for AI-Generated Data Code

A hierarchical model showing that current research focuses almost exclusively on the base layer, while production readiness requires assessment across all tiers.

Furthermore, a practical evaluation framework must analyze the inherent trade-off space between these competing quality attributes. There is often a tension between accuracy, latency, and cost. For instance, a more robust prompt architecture that injects extensive schema context and generates defensive null-checking code may yield higher correctness and robustness scores but at the cost of increased prompt token usage (directly translating to higher API cost) and potentially greater inference latency. Similarly, code optimized for computational efficiency might be less readable and thus more costly for a human engineer to modify later. Understanding these trade-offs is essential for making informed architectural decisions about how and when to deploy LLM-generated code. Is it acceptable to trade a 10% increase in cloud cost for a 50% reduction in data quality incidents? Current evaluation methodologies provide no data to answer such questions, as they silo each attribute or ignore all but correctness.

Influential evaluation frameworks from broader AI research provide a methodological foundation for this expansion. The work of Liang et al. in HELM (Holistic Evaluation of Language Models) exemplifies the shift towards multi-metric, scenario-based assessment, evaluating models across accuracy, robustness, fairness, and efficiency for a wide range of NLP tasks [2]. While not focused on code, its philosophy is directly applicable: a single aggregate score is insufficient; responsible deployment requires a "nutrition label" of performance across many dimensions. Translating this to data engineering code generation means developing a battery of tests for each quality attribute. For example, robustness can be evaluated using metamorphic testing creating multiple versions of a test dataset with injected anomalies and checking for consistent logical outcomes. Efficiency can be benchmarked by executing generated code on standardized datasets of increasing size within a containerized runtime environment. Research that begins to incorporate these facets moves the field from asking "Can the model write code that works?" to the far more pertinent question: "Can the model write code that is fit for a production data platform?" The

identified shortcoming, therefore, is that current evaluations predominantly measure if code works in a controlled sandbox, not if it is efficient, robust, secure, and maintainable in the complex, adversarial, and resource-conscious environment of a real-world data engineering context. Closing this gap is not optional; it is a prerequisite for moving from intriguing research demonstrations to trusted engineering tools that can safely augment the development of critical data infrastructure.

VI. SYNTHESIS AND RESEARCH JUSTIFICATION

The preceding thematic analysis reveals not three isolated gaps, but a tightly interdependent triad of shortcomings that collectively stifle the maturation of AI-assisted data engineering. The progression from heuristic prompt crafting to systematic Prompt Architecture (Gap 1) is fundamentally hamstrung by the Benchmark Deficit (Gap 2). One cannot rigorously compare the efficacy of a context-injected prompt template against a chain-of-thought decomposition if there exists no standardized, complex suite of data engineering tasks against which to run the experiment. Any findings would be anecdotal, tied to ad-hoc examples, and non-generalizable. Conversely, the creation of a meaningful benchmark is itself an exercise in prompt architecture, as the benchmark tasks must be designed and specified in a way that accurately reflects the structured context and decomposition challenges of real workflows. Furthermore, both of these endeavors are rendered practically moot without the expansion of evaluation criteria Beyond Basic Correctness (Gap 3). A new benchmark that only measures functional correctness via Pass@k would perpetuate the illusion of readiness while ignoring the operational realities of production systems. Similarly, advocating for a sophisticated prompt architecture is unconvincing if its only proven benefit is a marginal increase in correctness, without data on its impact on computational efficiency, robustness to data drift, or long-term maintainability. These three gaps form a closed loop of research immobility: without a benchmark, architectures cannot be tested; without multifaceted evaluation, benchmarks are inadequate; and without proven architectures, practical progress stalls.

This convergence points to a singular, compelling need: a unifying study that addresses all three gaps concurrently through an integrated, empirical methodology. The field requires research that does not merely propose another prompting heuristic or a narrow benchmark for a single task like text-to-SQL. Instead, it demands work that operationalizes prompt architecture, instantiates it within a novel, domain-specific benchmark, and evaluates its outputs against a holistic suite of metrics that matter to data platform engineers. Such a study would treat the prompt not as a magical incantation but as a software component with measurable performance characteristics. It would create a public resource—a benchmark suite—that becomes a shared foundation for future comparative research. Most critically, it would generate the first comprehensive dataset correlating specific prompt design choices (e.g., the granularity of schema context, the structure of decomposition) with multidimensional outcomes like execution efficiency, robustness failure rate, and token cost. This would move the community from debating principles to analyzing evidence.

The proposed primary research, "Benchmarking Prompt Architectures: A Quantitative Study of Contextual and Decomposed Prompting for Complex ETL Code Generation," is designed as a direct and necessary response to this synthesized gap analysis. It consciously avoids the trap of investigating these themes in isolation. Its methodology explicitly creates a novel benchmark suite of complex data transformation tasks, thereby directly addressing Gap 2 by providing the missing evaluation substrate. It then frames its investigation around the comparative analysis of defined prompt architectures—contextual versus decomposed templates—thereby attacking Gap 1 by subjecting architectural hypotheses to controlled experimentation. Finally, and most significantly, its evaluation framework is built upon the multi-faceted assessment advocated for in Gap 3. It pledges to measure not just functional correctness via unit tests, but also computational efficiency via runtime profiling and robustness via stress testing with edge-case data. This integrated approach is precisely what is required to break the current stasis. The work draws explicit inspiration from the call for rigorous, scenario-based evaluation seen in holistic frameworks like HELM, applying its philosophy to the specific domain of code generation [1]. It also embodies the empirical spirit of foundational code evaluation work, but extends it beyond the narrow confines of HumanEval-style puzzles into the messy, contextual world of data pipelines [2].

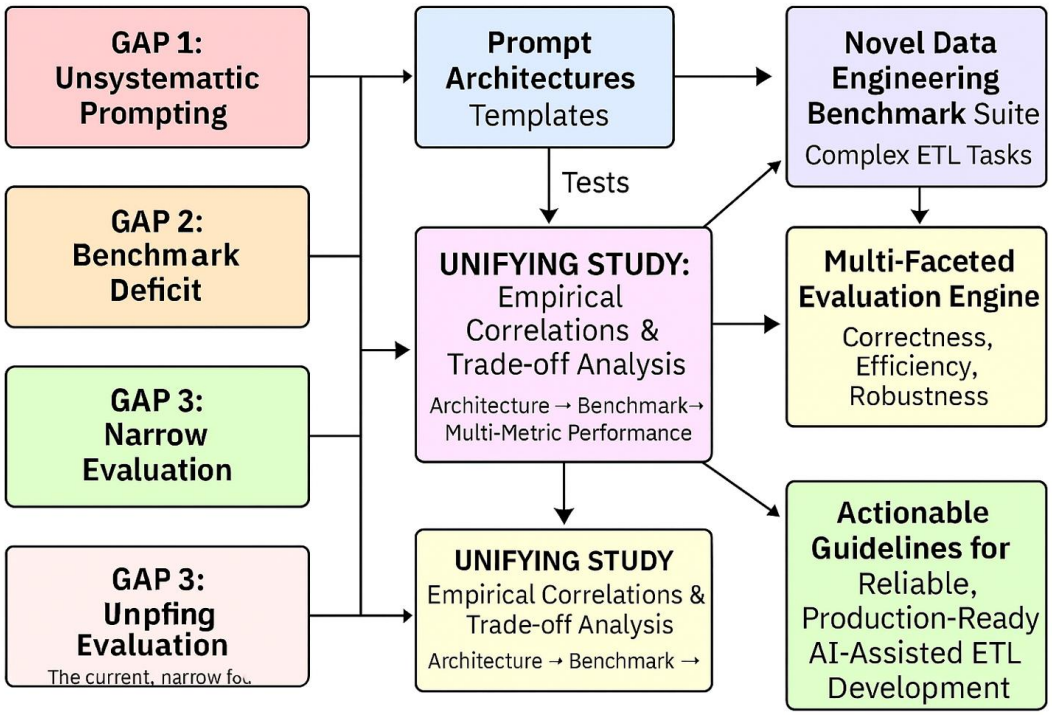

Figure 6: The Integrated Research Model Addressing the Triad of Gaps

The future trajectories illuminated by this synthesis are substantial. For tool builders, the outcomes would inform the design of next-generation IDE plugins and data engineering platforms, suggesting where to embed automated context-gathering (e.g., schema sniffers) and how to structure prompt-chaining workflows. For practitioners, it would replace folklore with evidence-based guidelines, such as "for idempotent merge operations, use a decomposed architecture with explicit state-checking steps." For educators, it would provide the concrete, quantitative backing needed to transform prompt engineering courses from collections of tips into a curriculum of validated techniques. By demonstrating a rigorous, end-to-end methodology for assessing AI-generated data code, this line of research lays the groundwork for establishing trust—the ultimate barrier to adoption. When data engineers can see not just that an AI can write code, but that its code has been systematically proven to be efficient, robust, and secure for a class of tasks, the technology transitions from a curious assistant to a credible component of the engineering toolkit. Therefore, the justification for the proposed research is not merely to fill an academic void, but to catalyze the responsible and effective industrialization of AI within one of the most critical domains of modern

enterprise technology. It seeks to provide the missing bridge between dazzling capability demonstrations and the disciplined, reliable engineering required to build and maintain the world's data infrastructure.

VII. CONCLUSION

This literature review has charted the transformative yet nascent journey of large language models from general-purpose code generation to the specialized frontier of data engineering. The analysis reveals a field marked by extraordinary potential but constrained by a triad of interdependent methodological gaps. The transition from heuristic prompt crafting to a discipline of systematic prompt architecture remains more aspirational than realized, as current practice lacks the empirical rigor to validate which design principles yield reliably superior outcomes for complex data tasks [1]. This shortcoming is exacerbated by a critical benchmark deficit, where existing evaluations like HumanEval fail to capture the schema-aware, stateful, and integrity-focused nature of production ETL workflows, leaving researchers and practitioners without a standardized proving ground [2]. Compounding these issues is the prevailing focus on basic functional correctness, which overlooks the essential production qualities of computational efficiency, robustness, and security that determine whether AI-generated code can be trusted in a live data platform.

The synthesis of these gaps underscores a singular imperative: the need for integrated research that concurrently develops domain-specific benchmarks, employs them to test structured prompt architectures, and evaluates outputs using a holistic, multi-faceted metric system. Such work is essential to move beyond demonstrations of capability and toward the establishment of a trustworthy, engineering-grade methodology. The proposed primary research on benchmarking prompt architectures for complex ETL generation represents a direct response to this imperative, aiming to transform prompt engineering from a folk art into a reproducible science for the data domain.

The trajectory this enables is significant. For the research community, it provides a foundational framework and a shared benchmark to accelerate meaningful progress. For

industry practitioners, it promises to replace anecdotal guidance with evidence-based protocols, reducing risk and increasing confidence in deploying AI-assisted development tools. Ultimately, the next frontier in AI-assisted data engineering is not merely the scaling of model parameters, but the disciplined development of the practices, standards, and evaluation frameworks required to harness these models safely and effectively. By addressing the identified gaps, we can begin to build the reliable, verifiable, and maintainable AI-powered pipelines that will form the next generation of data infrastructure, ensuring that the promise of AI translates into tangible, trustworthy value in the data-driven landscape.

REFERENCES

[1] M. Chen et al., "Evaluating Large Language Models Trained on Code," arXiv preprint arXiv:2107.03374, 2021. [Online]. Available: https://arxiv.org/abs/2107.03374

[2] T. Wu et al., "PromptChainer: Chaining Large Language Model Prompts through Visual Programming," in Proc. CHI Conf. Hum. Factors Comput. Syst. Extended Abstracts, New Orleans, LA, USA, 2022, pp. 1–7, doi: 10.1145/3491101.3519729.

[3] T. Scholak, N. Schucher, and D. Bahdanau, "PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models," in Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP), Punta Cana, Dominican Republic, 2021, pp. 9895–9901, doi: 10.18653/v1/2021.emnlp-main.779.

[4] J. Austin et al., "Program Synthesis with Large Language Models," arXiv preprint arXiv:2108.07732, 2021. [Online]. Available: https://arxiv.org/abs/2108.07732

[5] J. Li et al., "BIRD: A Big Bench for Large-scale Database Grounded Text-to-SQL Evaluation," in Proc. 37th Conf. Neural Inf. Process. Syst. (NeurIPS), New Orleans, LA, USA, 2023. [Online]. Available: https://arxiv.org/abs/2310.03255

[6] P. Liang et al., "Holistic Evaluation of Language Models," arXiv preprint arXiv:2211.09110, 2022. [Online]. Available: https://arxiv.org/abs/2211.09110