PromptCompEval: Evaluation Methodology for Comparing Prompt Compilation Frameworks

Jayanaka Dantanarayana, Savini Kashmira, Zichen Zhang,
University of Michigan
jayanaka, savinik, edenzha@umich.edu

Abstract

Prompt Engineering (PE) remains the dominant approach for controlling Large Language Model (LLM) behavior in AI-integrated applications, but it often incurs substantial developer effort. While prompt compilation techniques aim to reduce this burden, existing evaluations rely primarily on accuracy-centric benchmarks that fail to capture the trade-off between correctness and development cost. We introduce a prompt-centric evaluation framework that jointly measures task accuracy and developer effort through the Accuracy–Effort Cost (AEC) metric. We further propose a benchmark suite reflecting real-world AI-Integrated application patterns, including planning, tool use, memory management, and agentic coordination. Our results show that prompt compilation techniques, particularly when augmented with semantic annotations, achieve superior cost-adjusted performance compared to manual prompting under realistic workloads.

1 Introduction

Large Language Models (LLMs) are increasingly embedded within software systems to support multiple tasks. As a result, AI-integrated applications rely on LLMs during runtime to perform essential operations, seamlessly blending conventional software logic with AI-driven intelligence (Dantanarayana et al., 2025a; Weber, 2024; Beurer-Kellner et al., 2023; Li et al., 2023). As LLMs take on these responsibilities, developers must specify how the models behave in context, what data they operate on, what structure their outputs must follow, and what constraints govern their reasoning. Today, this control is primarily achieved through Prompt Engineering (PE), where developers manually craft natural language instructions to define and regulate LLM behavior (Liu et al., 2023). This emerging model is reshaping development practices and redefining how modern applications are conceived and engineered.

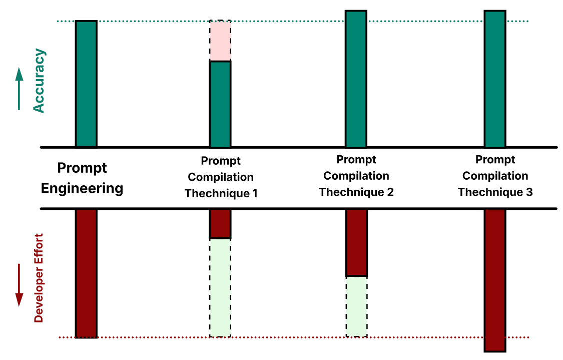


Figure 1: Trade-off between task accuracy and developer effort across prompting paradigms. Prompt Engineering achieves high accuracy at high development cost, while prompt compilation techniques reduce effort with varying impacts on accuracy, motivating cost-adjusted evaluation.

While effective in many cases, PE carries several challenges: prompts are brittle, verbose, difficult to maintain, and often tightly coupled to specific models or datasets (Liu et al., 2023). More importantly, PE demands substantial developer effort, often requiring dozens to hundreds of lines of handcrafted instructions to achieve desired behavior. As systems scale and integrate complex workflows such as multi-agent coordination or tool-based reasoning, this manual burden becomes increasingly unsustainable. To reduce this burden, a growing line of work has introduced prompt compilation frameworks, including DSPy (Khattab et al., 2023), LMQL (Beurer-Kellner et al., 2023), and Meaning-Typed Programming (MTP) (Dantanarayana et al., 2025a). These systems automatically translate structured code semantics into prompts by using compilation techniques, reducing the need for handcrafted natural language instructions and enhancing structural coherence and maintainability.

However, despite these emerging alternatives, the field currently lacks a principled way to measure the trade-off between output quality and developer effort. Most existing studies evaluate prompting methods using static reasoning benchmarks such as GSM8K (Cobbe et al., 2021), a grade school math dataset, or HotPotQA (Yang et al., 2018), a question answering dataset focused on multi hop reasoning. Although these benchmarks are useful for measuring model capabilities, they do not capture the requirements of modern AI integrated workflows, which often involve tool usage, memory and context management, planning, and multi-agent coordination. Our analysis of a public repository of prompts from deployed agentic systems (x1xhlol and Valbuena, 2025) shows that real world prompts encode far richer contextual and operational semantics than those found in standard academic benchmarks. For example, GSM8K prompts contain roughly 20 lines of text, whereas Cursor, a deployed AI coding assistant, includes prompts exceeding 460 lines. Consequently, evaluating prompting strategies solely on static benchmarks provides an incomplete picture of their effectiveness in real world applications. As a result, existing evaluations offer limited insight into how prompting approaches behave in practical systems.

Furthermore, prior work (Khattab et al., 2023; Beurer-Kellner et al., 2023) does not quantify how developer effort and accuracy are balanced, even though reducing effort is a primary motivation for prompt compilation. PE can require hundreds of lines of handcrafted instructions, whereas prompt compilation may involve modifying type annotations, data structures, or high level abstractions. Without a unified metric that jointly captures performance and effort, it is difficult to compare prompting paradigms or understand the trade offs between these approaches.

To address these limitations, we introduce an evaluation methodology for comparing PE and prompt compilation techniques under realistic AI integrated workloads. At its core is a unified prompt centric metric that captures the trade off between task performance and developer effort. We formalize this through the Accuracy–Effort Cost (AEC), which measures how effectively a prompting approach achieves correct outputs relative to the code level modifications it requires. By normalizing syntactic changes across implementations, AEC provides a single interpretable value that distinguishes approaches relying on large handcrafted prompts from those achieving similar accuracy with more lightweight specifications.

Complementing the metric, we develop a benchmark suite that reflects the operational demands of real world LLM applications. Instead of static reasoning tasks, these benchmarks capture patterns common in deployed agentic systems, including multimodal perception, tool interaction, memory and retrieval, multi agent workflows, and planning oriented behavior. This suite enables a more faithful evaluation of prompting methods in settings where coordination, context management, and structured control flow are central.

Using this framework, we compare PE, MTP, and DSPy. The results show that PE is accurate but requires high manual effort, prompt compilation lowers effort but can lose task semantics, and Semantic Engineering in MTP preserves accuracy while keeping developer cost low. This highlights the importance of evaluating prompting approaches by both correctness and developer effort.

2 Motivation and Background

2.1 Prompting Paradigms in AI-Integrated Applications

AI-Integrated applications increasingly rely on LLMs as runtime components that perform essential tasks such as planning, information retrieval, tool invocation, and structured content generation (Li et al., 2023; Beurer-Kellner et al., 2023). To control LLM behavior in these systems, developers primarily rely on *Prompt Engineering (PE)*, alongside an emerging trend toward *prompt compilation techniques*.

PE encodes task logic, constraints, and output structure through manually written natural-language prompts (Liu et al., 2023). Prompt compilation techniques, in contrast, generate prompts automatically from program structure, types, or declarative specifications, often integrating prompting logic directly into the software stack (Khattab et al., 2023; Beurer-Kellner et al., 2023; Dantanarayana et al., 2025a). Both paradigms are actively used in practice and aim to ensure correctness, robustness, and usability of LLM-based systems.

2.2 Limitations of Existing Evaluation Methodologies

Despite growing adoption of prompt compilation techniques, current evaluation methodologies remain largely unchanged. Most studies evaluate prompting approaches using static reasoning benchmarks such as GSM8K (Cobbe et al., 2021) or Hot-

PotQA (Yang et al., 2018) and report accuracy as the primary metric. While these benchmarks are effective for comparing model capabilities, they are not designed to evaluate prompting strategies themselves.

These datasets rely on short, text-only prompts and single-turn interactions, failing to reflect the operational complexity of AI-integrated systems. As a result, evaluations conducted on such benchmarks provide limited insight into how prompting approaches perform in real-world applications, where prompts encode extensive contextual logic, behavioral constraints, and orchestration patterns (Dantanarayana et al., 2025b).

2.3 The Missing Dimension: Developer Effort

A key motivation behind prompt compilation is the reduction of developer effort, yet this dimension is almost entirely absent from existing evaluations. PE often achieves high correctness but requires large, handcrafted prompts that are difficult to maintain (Dantanarayana et al., 2025b). Prompt compilation frameworks aim to reduce this burden by automating prompt generation, but may lose task-specific semantics unless additional structure or annotations are provided.

Accuracy-only evaluations systematically obscure this trade-off. Without measuring effort, it is impossible to distinguish approaches that achieve similar performance through vastly different levels of developer investment. Consequently, existing studies cannot answer whether a reduction in accuracy is justified by a significant decrease in development cost, or whether additional semantic annotations meaningfully improve cost-adjusted performance.

2.4 Mismatch Between Benchmarks and Real-World Workloads

In addition to metric limitations, existing benchmarks do not reflect real-world LLM application workloads. Deployed systems frequently involve multimodal inputs, dynamic tool use, memory and context management, multi-agent coordination, and planning-oriented behavior (Dantanarayana et al., 2025b).

Standard benchmarks such as GSM8K and HotPotQA do not expose these dimensions. As a result, they fail to reveal differences between prompting paradigms that only emerge under realistic conditions, leading to evaluation outcomes that may not generalize beyond narrow reasoning tasks.

Table 1: Capabilities exercised by deployed AI-integrated applications, capturing core dimensions often missing from standard reasoning benchmarks.

Category	**# Apps**	**Example Applications**
C1	6	VSCode Agent, Gemini AI Studio
C2	18	Claude Code, Replit, Lovable
C3	8	Claude Code
C4	6	Perplexity, Devin AI
C5	9	Traycer AI, Manus
C6	16	Qoder, Antigravity

C1: Multimodal Perception, **C2:** Tool Use and Integration,
C3: Memory and Context Management,
C4: Information Retrieval, **C5:** Multi-Agent Tasks,
C6: Planning and Reasoning

We categorize these capabilities into 6 recurring application dimensions observed in deployed systems: C1: multimodal perception, C2: tool use and integration, C3: memory and context management, C4: information retrieval, C5: multi-agent coordination, and C6: planning and reasoning.

We provide detailed definitions of these dimensions and describe how they are exercised by our benchmark tasks in Section 4.

Analysis of deployed AI-Integrated applications (x1xhlol and Valbuena, 2025) reveals that real-world systems routinely exercise a broad range of capabilities beyond single-turn reasoning. As shown in Table 1, modern applications frequently combine tool use, memory and context management, information retrieval, multi-agent coordination, and planning-oriented behavior. These capabilities rarely appear in isolation and often co-occur within a single application. Consequently, any evaluation methodology intended to compare prompting paradigms must account for all of these dimensions during analysis.

2.5 Motivation for a Prompt-Centric Evaluation Framework

The limitations outlined above motivate the need for a new evaluation framework that treats prompting strategies as first-class objects of study. Such a framework must (1) capture the trade-off between correctness and developer effort, and (2) evaluate prompting approaches on benchmarks representative of real-world AI-integrated applications.

In this work, we address these needs by introducing a unified metric that explicitly balances accuracy and developer effort, alongside a benchmark suite designed to reflect the operational characteristics of deployed LLM systems. Together, these components enable systematic and meaningful comparison of Prompt Engineering and prompt compilation techniques under realistic conditions.

3 Methodology

3.1 Overview

As discussed in previous sections, conventional evaluation methodologies are insufficient for comparing PE and prompt compilation techniques because they fail to capture the trade-off between task correctness and developer effort. Existing evaluations primarily measure output quality and implicitly assume comparable development cost across prompting approaches. In practice, however, prompting paradigms differ substantially in the amount of manual specification they require, making accuracy-only comparisons misleading.

To address this limitation, we introduce a prompt-centric metric called the *Accuracy–Effort Cost (AEC)*, which explicitly accounts for both task performance and developer effort. AEC is inspired by composite cost metrics such as the Energy–Delay Product (EDP) (Laros III et al., 2013) used in computer architecture to balance competing system objectives. Analogously, AEC penalizes approaches that achieve high accuracy through disproportionate development effort, while favoring methods that deliver strong performance with lower developer cost.

As shown in Figure 2 this metric is applicable to any benchmark that admits multiple implementations and allows fair comparison across prompting approaches. AEC relies on two key measurements, namely task accuracy and developer effort. In the remainder of this section, we describe how each quantity is measured and how they are combined to define AEC.

3.2 Measuring Task Accuracy

Let T denote a task and let M be a prompting approach (e.g., PE, MTP, DSPy). We define the accuracy of M on T as:

$$A(M,T) \in [0,1], \tag{1}$$

where A is computed using task-appropriate metrics such as accuracy, F1 score, or pass rate. For each task, we normalize accuracy relative to a Prompt Engineering baseline M_{PE}:

$$\hat{A}(M,T) = \frac{A(M,T)}{A(M_{\text{PE}},T)}. \tag{2}$$

This normalization ensures that $\hat{A}(M_{\text{PE}},T) = 1$ and allows us to interpret performance differences relative to a commonly adopted reference approach.

3.3 Measuring Developer Effort

Developer effort captures the amount of manual effort required to integrate an LLM into an application using a given prompting approach. Measuring this is non-trivial which requires some approximation. Selecting a representative approximated metric is essential for the truthfulness of AEC. The MTP (Dantanarayana et al., 2025a) paper introduces such an approximation which is *Lines of Code Added or Modified* for AI-Integration. This metric assumes a base implementation with a placeholder function for the feature to be handled by an LLM. The for each AI-Integration approach, the lines of code that needs to be added or modified is counted. This metric is a good metric as some prompt compilation techniques such as DSPy not only requires code addition but also modification of existing code.

In our research we selected this metric to approximate developer effort and obtain a normalized effort against a baseline AI-Integration approach.

Let $E(M,T)$ denote the effort required to implement task T using method M. We define effort as:

$$E(M,T) = \text{LoC}_{\text{added}} + \text{LoC}_{\text{modified}}, \tag{3}$$

where lines of code are counted after normalizing formatting and removing comments unrelated to prompting behavior. Effort is normalized with respect to the Prompt Engineering baseline:

$$\hat{E}(M,T) = \frac{E(M,T)}{E(M_{\text{PE}},T)}. \tag{4}$$

By construction, $\hat{E}(M_{\text{PE}},T) = 1$. Approaches that reduce developer burden yield $\hat{E}(M,T) < 1$.

3.4 Accuracy–Effort Cost (AEC)

Accuracy–Effort Cost (AEC) measures the developer effort required to achieve a unit of task accuracy. It captures the intuition that a prompting approach is preferable if it delivers comparable correctness with less manual specification.

We define the Accuracy–Effort Cost of method M on task T as the ratio of normalized developer effort to normalized task accuracy:

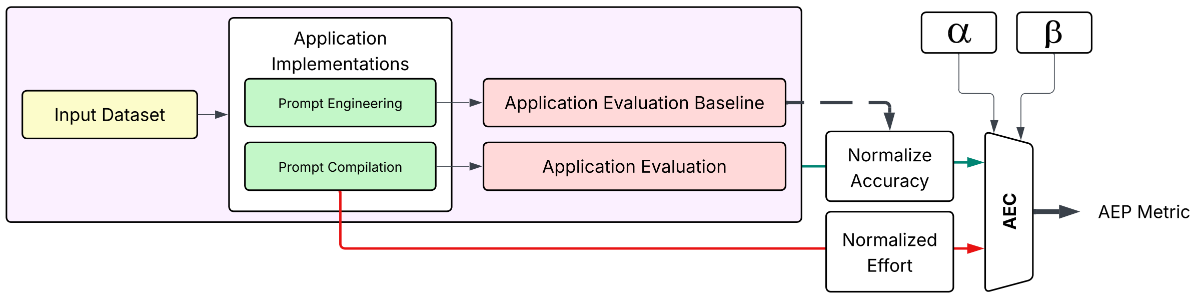


Figure 2: Evaluation methodology using AEC. Each application will be evaluated for accuracy on a defined dataset, while the effort is approximated using the LoC Added or Modified. These will then be used to calculate the AEC.

$$\mathrm{AEC}(M, T) = \frac{\hat{E}(M, T)}{\hat{A}(M, T)}. \tag{5}$$

Lower AEC values indicate better cost-adjusted performance. This formulation captures the central trade-off between correctness and effort. A method that achieves slightly lower accuracy but substantially reduces developer effort may yield a lower AEC than an approach that achieves high accuracy at significantly higher cost.

3.5 Generalized Accuracy–Effort Cost

Different applications may prioritize task accuracy and developer effort differently. To accommodate such preferences, we define a generalized form of AEC with tunable exponents:

$$\mathrm{AEC}_{\alpha,\beta}(M, T) = \hat{E}(M, T)^{\beta} \times \hat{A}(M, T)^{-\alpha}, \tag{6}$$

where $\alpha > 0$ controls sensitivity to accuracy and $\beta > 0$ controls sensitivity to developer effort. This formulation mirrors generalized composite cost metrics used in systems evaluation, allowing the metric to emphasize correctness or development cost depending on application requirements.

In this work, we primarily report results for $\alpha = \beta = 1$, corresponding to balanced Accuracy–Effort Cost. We additionally explore accuracy-sensitive and effort-sensitive variants by increasing α or β, respectively.

3.6 Aggregating Across Tasks

For a benchmark suite consisting of tasks $\mathcal{T}$, we compute the overall Accuracy–Effort Cost of method M as the mean across tasks:

$$\mathrm{AEC}(M) = \frac{1}{|\mathcal{T}|} \sum_{T \in \mathcal{T}} \mathrm{AEC}(M, T). \tag{7}$$

This aggregation enables comparison of prompting approaches across heterogeneous tasks while preserving per-task normalization and avoiding bias toward any single benchmark.

3.7 Interpretation

AEC provides a single interpretable metric that reflects how efficiently a prompting approach converts developer effort into task correctness. Unlike accuracy-only evaluations, AEC exposes cases where improvements in correctness are achieved at disproportionate development cost. Conversely, it highlights approaches that achieve competitive accuracy with significantly reduced manual specification.

By explicitly modeling the trade-off between accuracy and effort, AEC enables more meaningful evaluation of prompting paradigms in AI-integrated applications and provides a principled foundation for prompt-centric benchmark design.

4 Benchmark Design

Existing benchmarks such as GSM8K and HotPotQA evaluate LLMs under static, text-only settings and are insufficient for capturing the operational complexity of AI-integrated applications. To evaluate prompting strategies under realistic conditions, we design a benchmark suite grounded in common patterns observed in deployed LLM systems, following prior analyses in the Semantic Engineering literature.

4.1 Application Dimensions

We characterize real-world AI-integrated applications along six recurring dimensions, adapted from analyses of deployed agentic systems:

- **C1: Multimodal Perception.** Tasks where LLMs must reason jointly over text and non-textual inputs, such as images.

Table 2: Benchmark tasks, datasets, and associated application dimensions (C1–C6). Synthetic datasets are designed to reflect common AI-integrated application patterns.

Benchmark Task	Dataset	Dimensions
Memory Retrieval	Synthetic user memory profiles with short factual and preference-based entries queried via partial context	C3, C4
Multimodal Information Extraction	Sampled images from LAION-400M paired with text queries requiring visual attribute extraction	C1
Tool-Augmented Workflow	Synthetic API tasks requiring tool selection, argument formatting, and output interpretation	C2
Task-Oriented Planning	Synthetic goal specifications requiring generation of ordered multi-step action plans	C6
Agentic Coordination	Synthetic multi-agent tasks with role-specific responsibilities and shared objectives	C5

- **C2: Tool Use and Integration.** Scenarios requiring the model to select, invoke, and interpret outputs from external tools or APIs.
- **C3: Memory and Context Management.** Applications that depend on retrieving, summarizing, or reasoning over prior interaction history or long-term memory.
- **C4: Information Retrieval.** Workloads where correctness depends on retrieving relevant information from large corpora or knowledge bases.
- **C5: Multi-Agent Coordination.** Tasks involving multiple agents or roles that must communicate and coordinate to achieve a shared goal.
- **C6: Planning and Adaptive Decision-Making.** Applications that require generating, ordering, and adapting multi-step plans based on intermediate outcomes.

These dimensions are largely absent from standard reasoning benchmarks but are central to how prompting strategies are designed, implemented, and maintained in practice.

4.2 Benchmark Tasks

Based on these dimensions, we construct a suite of benchmark tasks that resemble simplified but representative real-world application patterns. Each task is designed to exercise one or more application dimensions while remaining sufficiently controlled for systematic evaluation.

Our benchmark suite includes:

- **Memory Retrieval**, which evaluates context-dependent querying over stored interaction histories and associated knowledge sources (C3, C4).
- **Multimodal Information Extraction**, which requires extracting semantically relevant information from images paired with text queries (C1).
- **Task-Oriented Planning**, where the model generates structured multi-step plans based on goals and system state (C6).
- **Tool-Augmented Workflows**, which assess correctness when selecting, invoking, and interpreting outputs from external tools or APIs (C2).
- **Agentic Coordination Tasks**, which involve decomposing goals across multiple roles or agents and coordinating their interactions (C5).

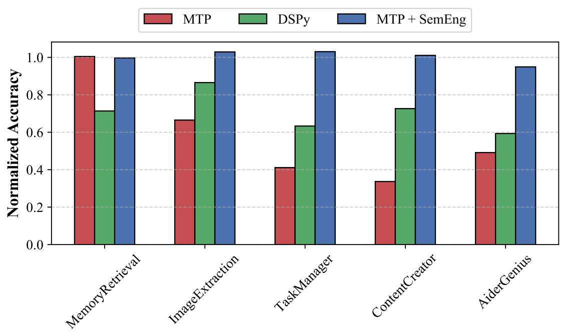
Each benchmark defines structured inputs, expected outputs, and task-specific evaluation metrics, enabling consistent comparison across Prompt Engineering and prompt compilation techniques.

5 Evaluation

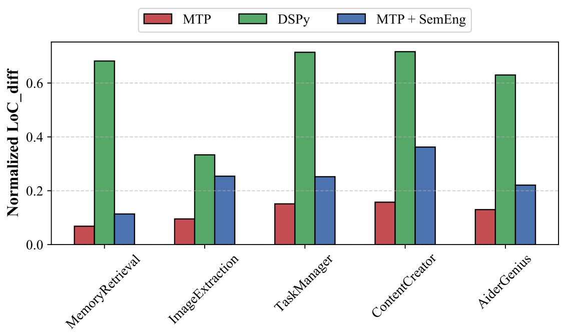
This section evaluates PE and prompt compilation techniques under the proposed evaluation framework. Our evaluation is designed to isolate the impact of prompting methodology by separately analyzing task accuracy, developer effort, and their combined trade-off using the AEC.

5.1 Evaluation Methodology

We evaluate three prompting approaches: PE, MTP, and DSPy. PE serves as the baseline, reflecting current best practices in manually engineered prompts. All approaches are evaluated on the benchmark suite described in Section 4, with datasets summarized in Table 2.



(a) Normalized task accuracy across benchmark tasks.



(b) Normalized developer effort measured as lines of code added or modified relative to Prompt Engineering.

Figure 3: Accuracy and developer effort results on the proposed benchmarks. While Prompt Engineering achieves strong accuracy, prompt compilation techniques substantially reduce developer effort.

For each benchmark task, all approaches use the same language model, decoding parameters, and task inputs to ensure that observed differences are attributable solely to the prompting methodology. Each task is implemented independently using each approach.

Task correctness is measured using task-appropriate metrics such as accuracy, F1 score, or semantic similarity. Developer effort is measured as lines of code added or modified relative to the PE implementation, with prompt text treated as code. Non-functional comments and formatting-only changes are excluded.

Accuracy and effort are normalized with respect to PE and combined using the Accuracy-Effort Cost (AEC), as defined in Section 3.

5.2 Task Accuracy

Figure 3a compares task accuracy across prompting approaches. PE achieves strong performance across benchmarks, particularly for tasks involving planning, memory management, and agentic coordination, where manual prompts can encode detailed task semantics.

Prompt compilation approaches such as MTP and DSPy perform comparably on simpler tasks but exhibit reduced accuracy on benchmarks requiring richer contextual semantics or structured orchestration. Semantic Engineering mitigates this gap by enriching program semantics with lightweight annotations, recovering much of the lost accuracy and approaching or matching PE performance in several cases.

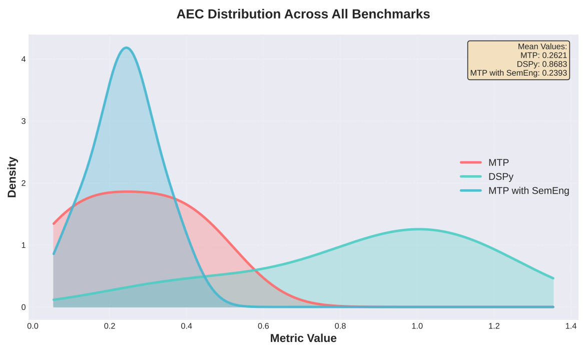
Table 3: Benchmark Comparison Results

Benchmark	Metric	MTP	DSPy	MTP v2
Memory Retrieval	AEC	**0.0679**	0.7801	0.1140
	AE^{2}C	**0.0046**	0.5319	0.0130
	A^{2}EC	**0.0676**	0.8926	0.1144
Image Extraction	AEC	**0.1432**	0.3852	0.2468
	AE^{2}C	**0.0136**	0.1284	0.0627
	A^{2}EC	0.2453	0.4451	**0.2398**
Task Manager	AEC	0.3679	1.1281	**0.2446**
	AE^{2}C	**0.0556**	0.8058	0.0617
	A^{2}EC	0.8947	1.7815	**0.2374**
Content Creator	AEC	0.4675	0.9865	**0.3584**
	AE^{2}C	**0.0736**	0.7069	0.1298
	A^{2}EC	1.3879	1.3583	**0.3547**
Aider Genius	AEC	0.2642	**1.0618**	0.2326
	AE^{2}C	**0.0343**	0.6688	0.0514
	A^{2}C	0.5375	1.7898	**0.2451**

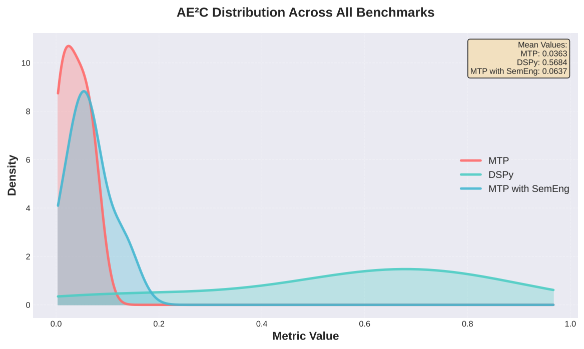
5.3 Developer Effort

Figure 3b compares developer effort measured as lines of code added or modified relative to the PE baseline. PE incurs the highest effort across all benchmarks due to large handcrafted prompts that encode formatting rules, reasoning scaffolds, and workflow constraints.

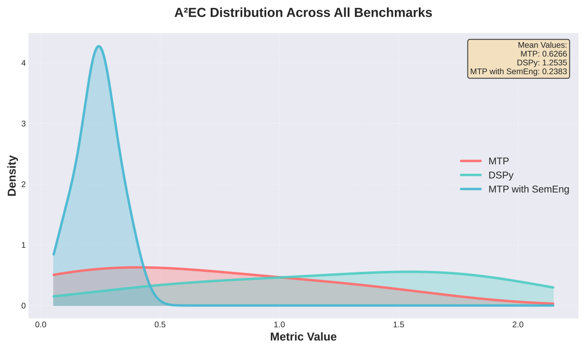
Prompt compilation techniques substantially reduce developer effort. MTP and DSPy require far fewer code modifications by leveraging program structure and declarative specifications, while Semantic Engineering adds lightweight annotations that still result in significantly lower effort than PE. These results show that approaches with similar accuracy can differ substantially in the developer effort they require.



(a) AEC distribution.



(b) AE^{2}C distribution (effort-sensitive).



(c) A^{2}EC distribution (accuracy-sensitive).

Figure 4: Distributions of cost-adjusted performance metrics across all benchmark tasks.

5.4 Accuracy–Effort Cost (AEC)

We evaluate prompting approaches using the Accuracy–Effort Cost (AEC) metric, which jointly captures task correctness and developer effort. Table 3 reports cost-adjusted performance across benchmark tasks under balanced (AEC), effort-sensitive (AE^{2}C), and accuracy-sensitive (A^{2}EC) formulations. Complementing these per-task results, Figure 4 presents the distributions of each metric aggregated across all benchmarks. Lower values indicate better cost-adjusted performance.

Although PE consistently achieves high task accuracy, its substantially higher developer effort leads to unfavorable cost-adjusted outcomes when evaluated using AEC. This pattern is evident across benchmarks such as *Task Manager* and *Content Creator*, where extensive handcrafted prompts inflate effort and result in higher AEC values despite strong correctness (Table 3). As a result, PE is frequently dominated by approaches that trade marginal accuracy loss for significant reductions in developer effort.

In contrast, prompt compilation techniques achieve markedly lower AEC by reducing effort through structured specifications and automated prompt generation. Across most tasks, both MTP and DSPy outperform PE under AEC and AE^{2}C, demonstrating that effort reduction alone can substantially improve cost-adjusted performance. However, DSPy exhibits higher variance and consistently worse outcomes in semantically rich tasks, reflecting sensitivity to accuracy degradation.

Semantic Engineering (MTP v2) consistently yields the most favorable trade-offs across metrics. As shown in Table 3, MTP v2 achieves the lowest AEC in four of five benchmarks and dominates under the accuracy-sensitive A^{2}EC metric in tasks such as *Task Manager*, *Content Creator*, and *Aider Genius*. These gains arise from recovering task semantics with minimal additional specification effort, allowing MTP v2 to approach PE-level accuracy while maintaining the low development cost of compilation.

The distributional results in Fig. 4 reinforce these conclusions. Under the balanced AEC metric (Fig. 4a), MTP v2 exhibits the lowest mean value and a tight distribution concentrated near zero, indicating robust cost-adjusted performance across heterogeneous tasks. When effort is emphasized (AE^{2}C; Fig. 4b), all compilation approaches substantially outperform PE, while MTP v2 preserves accuracy more effectively than DSPy. Under the accuracy-sensitive A^{2}EC metric (Fig. 4c), MTP v2 again demonstrates the most favorable distribution, avoiding the high-cost tail observed for DSPy.

Overall, these results show that conclusions drawn from accuracy alone differ substantially from those obtained using AEC. In multiple benchmarks, approaches with slightly lower accuracy outperform PE once developer effort is accounted for. Semantic Engineering consistently occupies a near–Pareto-optimal region in the accuracy–effort space, highlighting the importance of semantic recovery over manual prompt construction for scalable and maintainable AI-integrated applications.

6 Related Work

Prompt Engineering (PE) is the primary way developers control LLM behavior by embedding instructions, constraints, tool use, and output formats in natural language prompts. However, as prompts grow longer and more complex, especially in multi-step agentic systems, this approach becomes brittle and difficult to maintain.

To address these limitations, recent work has explored treating interaction with LLMs as a programming problem rather than an ad hoc prompting task. Frameworks such as LMQL (Beurer-Kellner et al., 2023) and DSPy (Khattab et al., 2023) introduce higher-level abstractions that compile declarative specifications into structured prompts and model calls. These approaches aim to reduce manual PE effort while improving modularity and reuse, though they often still rely on substantial natural-language specifications and do not explicitly evaluate developer effort.

Meaning-Typed Programming (MTP) (Dantanarayana et al., 2025a) introduces a language abstraction that uses program structure and types to derive prompt semantics and manage LLM interactions automatically. Follow-up work on Semantic Engineering (Dantanarayana et al., 2025b) shows that richer program semantics can reduce prompt verbosity without sacrificing correctness. Building on these ideas, our work provides an evaluation framework to systematically compare prompt engineering with prompt compilation approaches under realistic workloads.

Most prior evaluations of prompting techniques rely on static reasoning benchmarks such as GSM8K (Cobbe et al., 2021) or HotPotQA (Yang et al., 2018) and report accuracy as the primary metric. While effective for measuring model capabilities, these benchmarks are not designed to evaluate prompting strategies or AI-integrated applications, which often involve tool use, memory management, planning, and multi-agent coordination.

Recent analyses have highlighted the mismatch between such benchmarks and real-world LLM systems, where prompts encode extensive contextual logic and operational constraints (Dantanarayana et al., 2025b). However, existing evaluations largely omit developer effort as a first-class dimension, despite its central role in motivating prompt compilation frameworks.

Our work complements prior systems and programming-language research by introducing a prompt-centric evaluation methodology that jointly considers task correctness and developer effort. By combining a cost-based metric with benchmarks that reflect real-world AI-integrated application patterns, we provide a foundation for evaluating prompting paradigms beyond accuracy alone.

7 Conclusion

This paper argues that existing accuracy-centric evaluations are insufficient for comparing Prompt Engineering and prompt compilation techniques in AI-integrated applications. We introduce a prompt-centric evaluation framework that combines a cost-based metric, Accuracy–Effort Cost (AEC), with benchmarks reflecting real-world application capabilities such as tool use, memory management, and planning. Our results show that accounting for developer effort reveals trade-offs that are invisible under conventional evaluations, enabling more meaningful comparison of prompting paradigms. We hope this work encourages future evaluations to treat developer effort as a first-class consideration.

References

Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. 2023. Prompting is programming: A query language for large language models. *Proc. ACM Program. Lang.*, 7(PLDI).

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Jayanaka L. Dantanarayana, Yiping Kang, Kugesan Sivasothynathan, Christopher Clarke, Baichuan Li, Savini Kashmira, Krisztian Flautner, Lingjia Tang, and Jason Mars. 2025a. Mtp: A meaning-typed language abstraction for ai-integrated programming. *Proc. ACM Program. Lang.*, 9(OOPSLA2).

Jayanaka L. Dantanarayana, Savini Kashmira, Thakee Nathees, Zichen Zhang, Krisztian Flautner, Lingjia Tang, and Jason Mars. 2025b. Prompt less, smile more: Mtp with semantic engineering in lieu of prompt engineering. *Preprint*, arXiv:2511.19427.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2023. Dspy: Compiling declarative language model calls into self-improving pipelines. *Preprint*, arXiv:2310.03714.

James H. Laros III, Kevin Pedretti, Suzanne M. Kelly, Wei Shu, Kurt Ferreira, John Vandyke, and Courtenay Vaughan. 2013. *Energy Delay Product*, pages 51–55. Springer London, London.

Ziyang Li, Jiani Huang, and Mayur Naik. 2023. Scallop: A language for neurosymbolic programming. *Proc. ACM Program. Lang.*, 7(PLDI).

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Comput. Surv.*, 55(9).

Irene Weber. 2024. Large language models as software components: A taxonomy for llm-integrated applications. *Preprint*, arXiv:2406.10300.

x1xhlol and Lucas Valbuena. 2025. System prompts and models of ai tools. `https://github.com/x1xhlol/system-prompts-and-models-of-ai-tools`. Repository of system prompts, internal tools, and AI models from 30+ AI development platforms.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.