IPRPD
International Journal of Arts, Humanities and Social Sciences
ISSN 2693-2547 (Print), 2693-2555 (Online)
Volume 07; Issue no 04: April 2026
DOI: 10.56734/ijahss.v7n4a4

BUY OR BUILD? A PRACTITIONER'S FRAMEWORK FOR LARGE LANGUAGE MODEL INTEGRATION IN ENTERPRISE SYSTEMS

Dr. Khaled EL Tannir[1]

[1]*Artificial Intelligence Researcher, Montréal, Canada*

Abstract

The rapid proliferation of Large Language Models (LLMs) has confronted organizations with a consequential architectural decision: whether to build proprietary models, host open-source alternatives, or consume commercially available models through third-party APIs. This paper presents a multi-dimensional decision framework that synthesizes technical, financial, and strategic considerations into a coherent evaluation methodology for enterprise LLM adoption. Drawing on the end-to-end development of an LLM-powered document processing system—the Bills Converter—we trace the reasoning behind choosing a closed-source, API-based approach over self-hosted or custom-built alternatives. Our analysis covers deployment architectures, open-source versus closed-source trade-offs, tokenization economics, pricing structures, budgeting constraints, competitive differentiation strategies, and the emerging challenge of training data scarcity. We argue that the buy-versus-build decision is not binary but rather a phased continuum, where initial API adoption can give way to hybrid architectures as organizational maturity and requirements evolve. The framework is intended to serve as a practical reference for engineering teams and decision-makers navigating this rapidly shifting landscape.

Keywords

Large Language Models, LLM Deployment, API Integration, Build vs. Buy, Cost Optimization, Enterprise AI, Foundation Models

1 Introduction

The emergence of transformer-based Large Language Models has fundamentally altered the calculus of enterprise software development. A 2023 survey by McKinsey & Company reported that 72% of organizations had adopted some form of artificial intelligence in at least one business function, with 65% making regular use of generative AI—a figure that had nearly doubled from 33% the previous year[1]. Market analysts project the global LLM market to expand from approximately $6.4 billion in 2024 to $140.8 billion by 2033, propelled by adoption in healthcare, finance, customer service, and document processing[2].

For engineering teams considering LLM integration, the threshold question is deceptively simple: should the organization build its own model, host an existing one, or consume a commercial model through an API? In practice, this choice ramifies into a web of interconnected technical, financial, and strategic decisions—each of which shapes product quality, operational cost, time-to-market, and long-term competitive positioning.

This paper offers a structured framework for navigating these decisions. Rather than treating the question as a single binary choice, we decompose it into several interdependent dimensions: deployment architecture (Section 2), model provenance and licensing (Section 3), tokenization mechanics and their cost implications (Section 4), pricing models (Section 5), budgeting methodology (Section 6), competitive differentiation through customization (Section 7), and the emerging constraint of training data scarcity (Section 8). Section 9 presents a case study drawn from the development of an LLM-powered billing document processor, which concretizes these considerations in a real-world engineering context. Section 10 then synthesizes these dimensions into a strategic migration framework, identifying the operational thresholds that signal when an organization should transition from API consumption toward hybrid or self-hosted architectures.

The intended audience includes engineering managers, solutions architects, and technical decision-makers who need a consolidated, experience-grounded reference for LLM adoption planning. While the field evolves rapidly, the structural trade-offs examined here—control versus convenience, upfront cost versus marginal cost, flexibility versus operational simplicity—remain durable across model generations.

2 Deployment Architecture: Self-Hosting versus API Consumption

At the most fundamental level, an organization must decide how it will access an LLM: by running the model on infrastructure it controls, or by calling an external service over the network. Each approach carries distinct implications for performance, cost, privacy, and operational complexity.

2.1 Self-Hosting: Capabilities and Constraints

Deploying an LLM on proprietary infrastructure—whether on bare-metal servers or through a managed cloud service such as AWS, Google Cloud, or Azure—provides the highest degree of control over model behavior, data privacy, and customization. Organizations that handle regulated data (e.g., healthcare records, financial transactions) often find this control indispensable because it keeps all inference traffic within a trusted perimeter. Self-hosting also permits unrestricted fine-tuning: the model's weights can be adjusted on proprietary corpora without negotiating data-sharing agreements with external vendors.

The costs, however, are substantial. Running a large open-source model comparable to GPT-4—such as Falcon 180B—on a cloud platform can incur thousands of dollars per week in compute and storage charges alone[3]. Beyond raw infrastructure costs, self-hosting demands ongoing expertise in GPU cluster management, model serving frameworks (e.g., vLLM, TGI), network security, and version management. As the model and its supporting data grow, these burdens compound, making self-hosting a poor fit for teams that lack dedicated MLOps capacity or for applications where usage patterns are highly variable.

2.2 API-Based Consumption: Accessibility and Trade-offs

Third-party API providers—OpenAI, Anthropic, Google, Cohere, and others—abstract away infrastructure entirely, offering LLM capabilities as a metered service. This model drastically lowers the barrier to entry: developers need no specialized hardware, no deep familiarity with model internals, and can begin prototyping within hours. Pricing scales with usage, so a team running early experiments pays a fraction of what a production deployment would cost.

The trade-offs are well-documented. API consumers cede control over model updates, deprecation timelines, and rate-limiting policies. Data sent for inference transits external networks, raising privacy and compliance questions. Customization is limited to what the provider exposes—typically prompt engineering, system messages, and, in some cases, fine-tuning endpoints. Latency depends on the provider's infrastructure and geographic proximity, which can be a constraint for latency-sensitive applications.

2.3 Decision Heuristics

In our experience, the deployment choice is best framed by three variables: (i) the sensitivity of the data being processed, (ii) the team's operational maturity in managing ML infrastructure, and (iii) the predictability of usage volume. When data sensitivity is moderate, the team is small, and usage is uncertain, API consumption is almost always the rational starting point. Self-hosting becomes compelling when data cannot leave a controlled environment, when inference volume is high enough to amortize infrastructure costs, or when the application requires model modifications that APIs do not support.

3 Model Provenance: Open-Source versus Closed-Source

Orthogonal to the hosting decision—but deeply entangled with it—is the choice between open-source and closed-source models. This distinction affects not only what an organization can do with a model, but also how it plans for long-term evolution and risk.

3.1 Open-Source Models

Models released under open-source or permissive licenses—BERT, GPT-2, Mistral, Llama 2, Falcon, and their many derivatives—grant users the right to inspect, modify, and, in most cases, commercially deploy the weights. This transparency supports fine-tuning on domain-specific corpora, architectural modifications, and full auditability of model behavior. Open-source models are the natural complement to self-hosted deployments, since the organization retains end-to-end ownership of the inference pipeline.

Selecting an open-source model requires careful attention to several factors. Parameter count, often embedded in the model's name (e.g., "8B" for eight billion parameters), offers a rough proxy for capacity but does not fully determine performance; training data quality, data volume, and training methodology matter at least as

much[4]. Practically, parameter count constrains hardware requirements: an 8B-parameter model demands at minimum 8 GB of GPU memory for inference at full precision, and considerably more during fine-tuning. Community benchmarks, such as Hugging Face's Open LLM Leaderboard, provide comparative performance data, but these should be interpreted alongside task-specific evaluations relevant to the intended application.

3.2 Closed-Source Models

Closed-source models—GPT-4, Gemini, Claude, and others—are accessible exclusively through provider APIs. Their internal architectures, training data compositions, and weight values are proprietary. In exchange for this opacity, closed-source providers typically offer state-of-the-art performance, continuous improvement without user intervention, robust documentation, and enterprise-grade service-level agreements.

The principal limitation is inflexibility. Users cannot modify the core model; they can only influence behavior through prompting strategies, system-level instructions, and—where offered—limited fine-tuning interfaces. Organizations that depend on a closed-source model are also subject to vendor lock-in: pricing changes, deprecation of model versions, or shifts in acceptable-use policies can disrupt downstream applications with little recourse.

3.3 Selection Criteria

Beyond licensing, model selection should account for fluency and coherence of generated text, factual accuracy, and behavior with respect to bias and safety[5]. For regulated industries, the auditability afforded by open-source weights may be a prerequisite. For teams prioritizing time-to-value and willing to accept provider dependency, closed-source APIs offer a faster path to production.

4 Tokenization and Its Economic Implications

Understanding tokenization is essential not only for prompt engineering but also for cost estimation, because most API pricing is denominated in tokens. A token is the atomic unit of text processing in transformer-based LLMs: it may represent a complete word, a subword fragment, or a single character. The word "swimming," for example, is typically decomposed into two tokens: "swim" and "ing." OpenAI's general approximation is that one token corresponds to roughly four English characters, or about three-quarters of a word[6].

4.1 Context Windows and Token Budgets

Every LLM imposes a context window—the maximum number of tokens the model can attend to in a single forward pass—and a per-request token limit that caps the combined length of the input prompt and the generated output. These constraints have direct design consequences. Consider a 4,000-token limit: a prompt consuming 3,000 tokens leaves only 1,000 tokens for the response, often forcing the model to truncate or compress its output. Conversely, a 500-token prompt allows the model 3,500 tokens of headroom, which is adequate for most generative tasks.

The practical implication is that prompt engineering is, in part, an exercise in token budgeting. Developers must weigh the informational needs of the prompt against the expected length and quality of the output, all within a hard ceiling that the model cannot exceed. Overloading the prompt risks incomplete, summarized, or even mid-sentence truncated responses—outcomes that degrade user experience and waste expenditure.

4.2 Token-Level Cost Accounting

Because API providers bill per token, every architectural decision—prompt length, system message complexity, conversation history management, output verbosity—has a direct monetary cost. At scale, even modest inefficiencies compound. Reducing a prompt by 200 tokens across one million daily requests, at a rate of $0.03 per thousand tokens, yields a saving of $6,000 per day. This arithmetic makes tokenization literacy a prerequisite for any team operating LLM-powered services at production volumes.

5 Pricing Structures: Hosting versus Pay-per-Token

The financial architecture of an LLM deployment follows from the hosting decision described in Section 2, but warrants its own detailed treatment because pricing asymmetries between the two models can easily dominate total cost of ownership.

5.1 Self-Hosted Pricing

Self-hosting converts variable inference costs into fixed infrastructure expenditure. An organization leasing GPU instances for model serving pays by the hour regardless of how many (or how few) queries are processed. This is economically efficient only when utilization rates are consistently high. For workloads with sharp peaks and long

idle periods, the idle capacity represents pure waste. Early-stage projects face an especially harsh version of this calculus: they must commit infrastructure spend before they have reliable demand forecasts.

5.2 Token-Based API Pricing

Token-based pricing aligns cost with consumption, making it inherently more forgiving of demand uncertainty. Providers differentiate between input and output tokens, often charging more for the latter, and offer tiered pricing across model families. As a concrete illustration, consider a chatbot application serving 1,000 users, each submitting an average of seven queries generating approximately 3,000 input tokens and 2,000 output tokens per session. At GPT-4-class pricing, this workload costs roughly $315 per billing cycle. Migrating the same workload to a smaller model such as GPT-3.5 Turbo reduces the cost by approximately 90%, though with measurable degradation in response sophistication[7].

The key strategic insight is that token-based pricing permits incremental experimentation. A team can prototype with a premium model, evaluate quality, then downgrade to a cheaper model for cost savings—or maintain a tiered architecture where high-value interactions are routed to more capable (and expensive) models while routine queries are handled by lighter alternatives.

6 Budgeting for LLM Projects

The performance of an AI system is jointly determined by its dataset and its model size—a relationship that has been characterized as a scaling law[8]. A richer training corpus increases the diversity and depth of patterns a model can learn; a larger parameter count increases the model's capacity to retain and generalize from those patterns. However, both dimensions scale with cost: acquiring large, curated datasets is expensive, and training a proportionally large model demands commensurately powerful hardware. Sam Altman disclosed in a 2023 interview that training GPT-4 cost OpenAI upwards of $100 million[9]—a figure that places state-of-the-art model training beyond the reach of all but the most well-capitalized organizations.

For teams that do not build their own foundation models, budgeting still requires discipline. API costs, while lower in absolute terms, can escalate unpredictably if usage patterns are not monitored. A prudent budgeting methodology involves three steps: first, estimating usage volume and token consumption based on application design; second, allocating budget between data acquisition (if fine-tuning is planned), API fees, and engineering labor; and third, establishing cost ceilings with automated alerts to prevent overruns.

A recurring theme in our project experience is that smaller, specialized models frequently outperform larger general-purpose models on domain-specific tasks—and at a fraction of the cost. Where the task is well-defined, investing in prompt optimization or lightweight fine-tuning of a smaller model often yields better returns than simply paying for a larger one. The practical takeaway for budget-constrained teams is to avoid defaulting to the most capable model available; instead, start with the smallest model that meets quality thresholds and scale up only where measurable gains justify the incremental expense.

7 The Differentiation Paradox: Competing on Shared Infrastructure

If every competitor can access the same foundation model through the same API, where does competitive advantage reside? This question—central to the traditional "build versus buy" literature[10]—takes on new dimensions in the LLM era.

Classical business strategy holds that core, value-generating activities should be retained internally, while non-core functions can be safely outsourced. Yet many organizations recognize AI as strategically central even as they lack the resources to develop models independently. They find themselves relying on the same commercial APIs as their competitors, creating an apparent contradiction: the capability most critical to their differentiation is, in its raw form, a commodity.

The resolution lies in what we term the "adaptation layer." While the base model is shared, the way an organization applies that model to its specific domain, data, and user needs is not. Three techniques form the core of this adaptation layer. First, prompt engineering: the craft of designing input instructions that elicit precise, context-sensitive behavior from a general-purpose model. Second, retrieval-augmented generation (RAG), which supplements model inference with real-time retrieval from proprietary knowledge bases, grounding outputs in organization-specific facts. Third, fine-tuning, where a pre-trained model's weights are adjusted using proprietary data to improve performance on narrow, domain-specific tasks.

Together, these techniques mean that the competitive unit of analysis is not the model itself but the system built around it. This has workforce implications: demand for engineers skilled in prompt optimization, RAG pipeline design, and fine-tuning workflows is rising sharply, and these roles are becoming a strategic bottleneck for many organizations.

8 Training Data Scarcity: A Structural Constraint on Scaling

The scaling trajectory that has driven LLM progress—larger models trained on larger corpora—faces a constraint that, unlike compute costs, cannot be solved by additional investment alone: the exhaustion of high-quality, publicly available training data.

8.1 The Synthetic Data Feedback Loop

As tools like ChatGPT become widely used, an increasing fraction of newly published text is itself AI-generated. When next-generation models train on web-scraped corpora that include substantial AI-authored content, they risk absorbing the repetitive patterns, stylistic homogeneity, and factual inaccuracies of their predecessors. This creates a feedback loop in which each successive model generation echoes—and potentially amplifies—the shortcomings of the last, increasing the probability of hallucination and reinforcing existing biases[11].

8.2 Legal and Policy Barriers

The landscape of permissible data acquisition has shifted markedly. The New York Times, Shutterstock, and prominent authors including John Grisham have filed suits alleging unauthorized use of copyrighted material for model training. Platforms that once served as rich sources of diverse, human-authored text—Reddit and Quora among them—have revised their terms of service to prohibit large-scale scraping by AI developers. These legal and policy developments constrict the pool of data available for training, forcing model developers toward licensed content.

8.3 The Emerging Data Market

OpenAI has responded by establishing a content licensing program, reportedly offering publishers between $1 million and $5 million annually for access to archival content. Agreements have been announced with Axel Springer, The Associated Press, Le Monde, and Prisa Media, among others[12]. As competing model developers enter the market for premium datasets, licensing costs are expected to rise, potentially concentrating cutting-edge model development among a shrinking set of well-funded firms. For organizations evaluating the build-versus-buy decision, this trend reinforces the economic argument for consuming models rather than training them: the cost of assembling a competitive training corpus is climbing rapidly and will likely continue to do so.

9 Case Study: The Bills Converter Application

To ground the preceding framework in practical experience, we describe the architectural decision-making process behind the Bills Converter, an LLM-powered tool for automated billing document processing developed by the first author over a period of several months.

9.1 Application Context

The Bills Converter was designed for a client who needed to ingest heterogeneous billing documents (PDFs, scanned images, and structured invoices), extract relevant financial data using natural language processing, transform it into structured tabular formats, and load the results into a relational database for analytical querying. The user interface was built with Python Streamlit to provide accessible, browser-based interaction.

9.2 Decision Rationale

The development team evaluated the deployment options discussed in Sections 2 through 7 and arrived at the following conclusions:

Deployment architecture. The team lacked dedicated GPU infrastructure and MLOps expertise. The client's data, while commercially sensitive, did not fall under regulatory frameworks that would prohibit transmission to a third-party API provider. API consumption was therefore selected as the deployment model, eliminating infrastructure overhead and enabling the team to begin prototyping within days rather than weeks.

Model provenance. Given the need for high-quality natural language understanding across diverse document formats, the team opted for OpenAI's GPT-4 family—a closed-source model offering benchmark-leading performance in text comprehension, entity extraction, and structured output generation. The trade-off in customization flexibility was accepted in exchange for reliability and development speed.

Pricing and budgeting. Token-based API pricing allowed the project to begin with negligible fixed costs. During prototyping, the team profiled token consumption per document type and estimated production costs by extrapolating from the observed per-document token counts. This data informed the budget allocation and pricing tier selection.

Differentiation strategy. While the underlying model was a commodity accessible to any API consumer, the team invested in domain-specific prompt engineering to improve extraction accuracy for the client's particular

document formats. System prompts were iteratively refined through manual evaluation, and a lightweight RAG component was incorporated to supply contextual reference data (e.g., vendor catalogs, known billing codes) at inference time.

9.3 Outcomes and Observations

The phased approach—starting with API consumption and investing differentiation effort in the adaptation layer rather than in model development—proved effective for this project's scale and constraints. The prototype reached functional status within a compressed timeline, and the client was able to evaluate the system's utility before committing to a longer-term architecture. The team documented several observations that may generalize:

First, prompt engineering absorbed a disproportionate share of development effort relative to initial expectations. Small changes in prompt structure produced large variations in extraction accuracy, underscoring the importance of systematic prompt evaluation. Second, token consumption varied significantly across document types—complex multi-page invoices consumed three to five times more tokens than simple single-page bills—making per-document cost prediction unreliable without empirical profiling. Third, the team retained the option to migrate to a self-hosted open-source model if usage volume eventually justified the infrastructure investment, but as of this writing, the cost-benefit analysis has not yet favored that transition.

10 The Strategic Migration Path: From API to Hybrid Architecture

The transition from consuming a third-party API to hosting a proprietary or open-source model is rarely a wholesale replacement; instead, it is a strategic migration triggered by specific operational thresholds. Based on the framework presented in the preceding sections, organizations should evaluate three primary migration triggers that signal when the initial API-based architecture has reached its limits and a more involved deployment model is warranted.

10.1 Economic Crossover Points

The most objective trigger for migration is what we term the Inference Amortization Point. As discussed in Section 5, API pricing is forgiving of demand uncertainty because it scales linearly with volume: an organization pays only for the tokens it consumes, with no idle capacity penalty. Conversely, self-hosting carries high fixed infrastructure costs—GPU leases, storage, networking, MLOps labor—but a marginal cost per token that approaches zero as utilization nears full capacity. These two cost curves inevitably cross at some usage volume.

Organizations should monitor a metric we call Token Density per Dollar: the number of tokens processed per unit of expenditure under each pricing model. When the monthly API expenditure consistently exceeds the fully loaded cost of a dedicated GPU cluster—including not only hardware lease but also the salaries and tooling of the MLOps team required to operate it—the economic case for migration becomes compelling. In practice, this crossover often occurs when inference volume stabilizes at a predictable, high level, which removes the demand uncertainty that initially favored the API model.

10.2 Data Gravity and Compliance Thresholds

As demonstrated in the Bills Converter case study (Section 9), initial prototyping often proceeds with commercially sensitive data that does not fall under strict regulatory frameworks prohibiting transmission to external providers. However, as an application matures and expands into more heavily regulated domains—healthcare records subject to HIPAA, financial transactions governed by GDPR or MiFID II, government documents with sovereignty requirements—the calculus shifts.

We describe this shift as a Data Gravity threshold: the point at which the accumulated volume and sensitivity of the data being processed create a gravitational pull toward on-premises or private-cloud inference. When the cost of maintaining compliance while routing data through external APIs—audit trails, data processing agreements, cross-border transfer mechanisms—exceeds the operational convenience that the API provides, migrating to a self-hosted open-source model within a trusted perimeter becomes a prerequisite rather than an option. This transition is not purely economic; it is often mandated by legal counsel or regulatory auditors, and organizations that have not architected for portability may face costly re-engineering.

10.3 Performance-Driven Specialization

A final trigger for migration is what we term the Differentiation Bottleneck. The adaptation layer described in Section 7—prompt engineering, retrieval-augmented generation, and system-level instructions—provides substantial leverage for customizing a general-purpose model's behavior. However, these techniques operate entirely at the input and context level; they cannot modify the model's internal representations or weight distributions. When domain-specific accuracy plateaus despite iterative prompt refinement—a phenomenon we observed during the Bills Converter development, where certain document formats stubbornly resisted prompt-based correction—the organization confronts a ceiling that only weight-level modification can breach.

At this juncture, fine-tuning an open-source model (e.g., Mistral, Llama 2, or their successors) on proprietary, domain-specific corpora offers a path forward. Fine-tuning effectively "bakes" domain knowledge directly into the model's weights, often producing a smaller, faster, and more specialized model that outperforms a general-purpose closed-source API on the target task distribution. The trade-off is clear: the organization assumes the operational burden of model hosting and the engineering cost of a fine-tuning pipeline, but gains performance headroom and independence from the base provider's update cycle. For applications where marginal accuracy improvements translate directly into business value—automated compliance checking, medical document parsing, financial data extraction—this trade-off is frequently worthwhile.

10.4 Migration Decision Matrix

The following matrix summarizes the operational thresholds that guide the transition along the buy-versus-build continuum. It serves as a diagnostic tool for organizations to determine if their current LLM architecture remains optimized for their evolving requirements.

Migration Driver	**Phase 1: API Consumption**	**Transition Trigger**	**Phase 2: Hybrid/Self-Hosted**
Economics	**Pay-per-token:** High variable cost; zero idle waste.	**Inference Amortization**: Monthly API expenditure consistently exceeds the fully loaded cost of a dedicated GPU cluster and MLOps labor.	**Fixed Infrastructure:** High upfront/fixed cost; marginal cost per query approaches zero.
Data & Compliance	**External Transit:** Inference data transits external networks to third-party providers.	**Data Gravity:** Accumulated volume or increased sensitivity (e.g., HIPAA, GDPR) creates a mandate for local inference control.	**Trusted Perimeter:** Inference traffic remains entirely within a private cloud or on-premises environment.
Performance	**Adaptation Layer:** Customization achieved via prompt engineering and RAG.	**Differentiation Bottleneck:** Domain-specific accuracy plateaus; certain edge cases or formats stubbornly resist context-level correction.	**Weight Modification**: Fine-tuning on proprietary corpora to "bake" domain knowledge directly into the model's weights.

11 Discussion and Future Directions

The framework presented here is necessarily a snapshot of a fast-moving field. Several trends are likely to reshape the buy-versus-build calculus in the near term.

First, the rapid improvement of open-source models is narrowing the quality gap with closed-source alternatives. If this trend continues, the cost advantage of self-hosting will strengthen, potentially shifting the equilibrium toward build for a wider range of organizations. Second, emerging inference optimization techniques—quantization, speculative decoding, mixture-of-experts architectures—are reducing the hardware requirements for self-hosting, which lowers the infrastructure barrier. Third, the regulatory landscape around data privacy and AI governance is evolving unevenly across jurisdictions; organizations operating in multiple regulatory environments may find that deployment architecture choices are increasingly dictated by compliance requirements rather than by cost or performance considerations alone.

We also note a structural tension in the current market. As training data becomes scarcer and more expensive, the cost of building foundation models is rising, which favors buying. Simultaneously, as inference costs decline and open-source model quality improves, the cost of running models is falling, which favors building (or at least self-hosting). How these opposing trends resolve will depend on the pace of progress in synthetic data generation, the outcomes of ongoing intellectual property litigation, and the competitive dynamics among the small number of firms currently capable of producing frontier models.

12 Conclusion

This paper has argued that the LLM buy-versus-build decision should be understood not as a one-time binary choice but as a multi-dimensional, evolving evaluation that spans deployment architecture, model licensing, tokenization economics, pricing structures, budgeting discipline, differentiation strategy, data availability, and strategic migration planning. We have proposed a practitioner-oriented framework that decomposes this evaluation into tractable sub-decisions, each with identifiable trade-offs and decision heuristics, and have identified three concrete migration triggers—economic crossover, data gravity, and performance-driven specialization—that signal

when an organization should advance along the continuum from pure API consumption toward hybrid or self-hosted architectures.

The Bills Converter case study illustrates one path through this decision space: API-based consumption of a closed-source model, with differentiation achieved through prompt engineering and retrieval-augmented generation rather than model customization. This approach minimized upfront investment, accelerated time-to-prototype, and preserved optionality for future architectural evolution—including the migration paths articulated in Section 10.

We hope this framework proves useful to practitioners facing analogous decisions, and we encourage future work that supplements it with quantitative benchmarks—particularly comparative total-cost-of-ownership analyses and systematic evaluations of open-source versus closed-source model performance across diverse application domains.

References

1. McKinsey & Company: The state of AI in 2023: Generative AI's breakout year. McKinsey Global Survey (2023)
2. Dimension Market Research: Global Large Language Model Market Report 2024–2033. Dimension Market Research (2024)
3. Hugging Face: Falcon-180B model card and deployment benchmarks. Hugging Face Hub (2023)
4. Kaplan, J., McCandlish, S., Henighan, T., et al.: Scaling laws for neural language models. arXiv preprint arXiv:2001.08361 (2020)
5. Liang, P., Bommasani, R., Lee, T., et al.: Holistic evaluation of language models. Transactions on Machine Learning Research (2023)
6. OpenAI: Tokenizer documentation and API pricing reference. OpenAI Platform Documentation (2024)
7. GPT for Work: OpenAI ChatGPT API pricing calculator. https://gptforwork.com/tools/openai-chatgpt-api-pricing-calculator (2024)
8. Hoffmann, J., Borgeaud, S., Mensch, A., et al.: Training compute-optimal large language models (Chinchilla). arXiv preprint arXiv:2203.15556 (2022)
9. Altman, S.: Interview remarks on GPT-4 training costs. Reported in multiple technology outlets (2023)
10. Williamson, O.E.: The economics of organization: The transaction cost approach. American Journal of Sociology 87(3), 548–577 (1981)
11. Shumailov, I., Shumilo, Z., Zhao, Y., et al.: The curse of recursion: Training on generated data makes models forget. arXiv preprint arXiv:2305.17493 (2023)
12. OpenAI: Content partnerships and licensing program announcements. OpenAI Blog (2024)
13. Vaswani, A., Shazeer, N., Parmar, N., et al.: Attention is all you need. In: Advances in Neural Information Processing Systems, vol. 30 (2017)
14. Devlin, J., Chang, M.-W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. In: NAACL-HLT (2019)
15. Touvron, H., Martin, L., Stone, K., et al.: Llama 2: Open foundation and fine-tuned chat models. arXiv preprint arXiv:2307.09288 (2023)