International Journal of Scientific Research and Modern Technology (IJSRMT) *ijsrmt.com*
Volume 3, Issue 11, 2024
DOI: https://doi.org/10.38124/ijsrmt.v3i11.1567

Production LLM Deployment: A Practitioner's Field Guide to Patterns and Pitfalls

Akhil Reddy Mandadi[1]

[1]Independent Researcher

Publication Date: 2024/11/28

Abstract

The deployment of Large Language Models (LLMs) from experimental showcases to enterprise mission-critical use cases in software engineering, customer support, knowledge management, healthcare, finance, and education has taken a new turn. While progress has been made in the ability to model, many organizations face serious problems moving from successful prototypes to reliable production systems. It's not enough to just choose a reliable model; reliability, scalability, observability, governance, security and cost-efficiency are all essential for practical deployment. With real-world deployment experience in a variety of production environments, this paper offers a practitioner-friendly field guide combining common engineering patterns and common pitfalls of production deployment.

The discussion does not claim to be exhaustive in its consideration of the literature but focuses on practices for successful deployment that are generally successful under production constraints. A total of eight architectural and operational patterns are explored, such as prompt versioning, retrieval grounding, structured output validation, fallback mechanisms, cost telemetry, observability, progressive rollout strategies and human-in-the-loop validation.

More so, the five common anti-patterns observed during deployment are explored to demonstrate failure patterns that are common and often lead to decreased system reliability, higher operational costs, and loss of user confidence. Organizations are accompanied with practical engineering guidance and deployment considerations that can be adapted to different operational contexts, each pattern. Finally, the paper outlines actionable recommendations and a checklist for deployment readiness, to help engineering teams build resilient, maintainable and trustworthy production LLM systems to ensure continuous improvement in fast-changing AI ecosystems.

Keywords: *Large Language Models (LLMs); Production AI Systems; AI Engineering; Prompt Engineering; Retrieval-Augmented Generation (RAG); MLOps; LLM Deployment; AI System Architecture*.

I. INTRODUCTION

- *Background*

The rise of foundation models has revolutionized the creation and application of AI systems in various sectors. Foundation models are pre-trained on a large-scale dataset using diverse data, and can be fine-tuned and prompted to a wide range of downstream tasks via retrieval systems. This transformation has been catalyzing the rapid deployment of sophisticated Large Language Models (LLMs), like GPT models, Claude, Gemini, and some open-source options such as LLaMA, Mistral, and Falcon, allowing organizations to easily embed conversational intelligence into their operations faster than ever (Hadi et al., 2023; Yang et al., 2024). These models are increasingly crucial to today's digital transformation efforts as they gain greater ability to reason, to generate code, to understand documents, and to interact with people using natural language.

There is a growing trend of organizations adopting LLM applications in a wide variety of work environments. Many organizations are turning to an LLM application for various business contexts. Conversational agents are used in customer support platforms to automatically respond to inquiries, handle simple ones, and ensure round-the-clock availability. Internal assistants improve employee efficiency by accessing organizational information, analyzing documents and helping to make decisions. Coding assistants support software developers by code generation, debugging, refactoring and documentation, which increases the efficiency of software development (Fan et al., 2023; Rusum, 2023).

Mandadi, A. R. (2024). Production LLM Deployment: A Practitioner's Field Guide to Patterns and Pitfalls. *International Journal of Scientific Research and Modern Technology*, *3*(11), 223–235. https://doi.org/10.38124/ijsrmt.v3i11.1567

In enterprise environments, search systems that combine LLM and RAG can provide relevant information retrieval from knowledge bases, and in the healthcare sector, they can manage clinical documentation, access medical knowledge, and patient communications (Thirunavukarasu et al., 2023; Huang et al., 2023). In highly regulated sectors such as financial services, financial institutions also use LLMs in areas like customer service interactions, document processing, fraud detection assistance, and regulatory reporting, further highlighting their expanding reach and impact on foundation models (Yang et al., 2024).

Although this is widespread practice, making a building an LLM demonstration is still a much simpler task than putting a production system in place. All that is needed for a prototype is API access and well considered prompts to generate convincing output. Production deployment is by contrast a much broader engineering exercise that takes into account issues of scalability, security, observability, governance, latency, reliability, and operational cost. To ensure efficient and timely reply from production systems, they should be capable of providing accurate, trustworthy, and efficient replies under various workloads and ensure compliance to the organizational policies and regulatory requirements (Ozkaya, 2023; Patel et al., 2024). However, it is important to note that the success of a production deployment isn't just about prompt engineering and the selection of a model it relies on software engineering, MLOps, and operational governance, as well, to guarantee sustainable and reliable AI services (Modalavalasa, 2022; Vankayala, 2023).

- *The Production Gap*

While many LLM prototypes can perform well in their development phase, there is a significant disconnect between the experiments and reliable production deployment? Applications that work well in a lab environment often fail to operate in the real world, when they deal with real users, real workloads and changing business needs. This production gap is due to the fact that the technical, organizational and operational challenges present in the deployment environment are not necessarily considered during the making of prototypes (Yang et al., 2024; Ozkaya 2023).

Hallucination is another common problem that occurs when LLMs provide false or fabricated information, usually with high confidence. This can severely impact user confidence and pose serious risks in industries where precision is vital, such as healthcare, finance, and enterprise knowledge management (Liu et al., 2023; Thirunavukarasu et al., 2023). Also, there are latency requirements, where the notion of 'near real time' is established even if the models are computing complex reasoning or retrieval tasks. Furthermore, infrastructure costs make deployment difficult due to the cost of consuming tokens, embedding generation, GPU resources and API usage from outside (Yang et al., 2024).

Monitoring is yet another big challenge. The traditional methods used for software monitoring are not effective in addressing issues like prompt quality, response consistency, model drift, token usage, or retrieval efficiency, which can lead to production failures that are hard to detect (Shankar et al., 2024). Prompt instability can also lead to variable application behavior, where even small changes in prompts might generate significantly different results across different model versions (Fagbohun et al., 2024; Zamfirescu-Pereira et al., 2023). Additionally, when retrieval of information from the RAG system is unsuccessful, it can lead to incomplete, outdated, or irrelevant contextual information being added to the response, despite the model's prowess (Kenthapadi et al., 2024). Organizations need to be concerned about the uncertainty of operations due to frequent updates to commercial model APIs, such as functionality changes, pricing adjustments, rate limits, and changes to the features supported by the APIs (Patel et al., 2024).

- *Research Motivation*

The present academic study on LLMs has significantly advanced understanding of model architecture, benchmark performance, prompt engineering techniques, reasoning abilities, model alignment and evaluation methods. Prior work has extensively explored transformer designs, application areas, performance baselines, the trustworthiness of transformers, and the limitations of models (Fan et al., 2023; Hadi et al., 2023; Liu et al., 2023). Likewise, there are many measures suggested for prompt design, model quality, grounding techniques, and human-AI interaction (Kenthapadi et al., 2024; Lee et al., 2024).

Much less focus has been paid to the engineering truth of taking LLM apps into production. Rather, there's not much discussion about the question of prompt version control, rollback strategies, observability, deployment monitoring, fall back architectures, cost management of the operation and production governance, which are all critical factors of successful deployments. For this reason, organizations still have to heavily depend on internal experimentation and engineering experience when tackling production challenges.

This paper seeks to fill that void with a practitioner-oriented field guide that is based on the common deployment experiences observed in the field, not on the development of theory. Rather than perform another detailed engineering survey of LLM technologies, this paper focuses on engineering patterns, operational lessons, and deployment strategies that have proven to be effective in production environments. The goal is to give practical insights that can help engineering teams involved in designing, deploying, maintaining and continually improving production grade LLM applications.

- *Objectives*

The goal of this paper is to categorize and classify the recurring architectural and operational patterns that have been proven to be successful in the deployment of the LLM for various applications. It also aims to determine common deployment issues and engineering missteps that compromise system reliability, lead to higher costs of

operations, and impact user trust. Based on these findings the paper proposes engineering practices for increasing the scalability, observability, maintainability, governance, and deployment resilience of LLM lifecycles. Finally, the study aims to connect theory with practice, make sense of the practical experience of deployment, and provide a structured framework that can be directly applied to creating production-ready LLM systems by practitioners.

- *Contributions*

This paper has four main contributions in the field of production LLM engineering. First, it introduces eight common production patterns, which, in real-life deployment situations, have proven successful. Second, it enumerates 5 popular anti-patterns that often lead to deployment misfortunes, inefficiencies in operation and poor system performance. Third, it offers pertinent deployment advice based on engineering experience to help design, operate, monitor, and continually improve production LLM applications. Lastly, the paper proposes a practical checklist for the deployment readiness of the LLM systems, which can help evaluate the production readiness, minimize deployment risk, and build reliable engineering workflows to deploy and maintain large-scale LLM systems.

II. PRACTITIONER PERSPECTIVE AND METHODOLOGY

The methodology used in this paper differs from the more common empirical approach based on controlled experiments, benchmark datasets, or systematic literature reviews, and is more of a practitioner's approach based on production deployment experience. The methodological approach acknowledges that numerous operational issues with the Large Language Models (LLMs) become manifest once the models are deployed in real environments, in which systems engage in ongoing interactions with users, external services, and organizational rules, as well as changing business needs. Although previous work has significantly contributed to the understanding of the capabilities of models, prompt engineering, alignment, and evaluation, there are relatively less studies that focus on the management of production reliability, operational risk, deployment governance, observability, and long-term system maintenance (Fan et al., 2023; Hadi et al., 2023; Yang et al., 2024). As a result, the study puts the experience of practical deployment to work as an important source of engineering knowledge, in addition to, not in place of, the existing academic study.

The methods used in this paper are inspired by qualitative synthesis of recurring engineering observations gathered through various productions of LLM deployments. These observations are based on a series of deployment retrospectives, engineering incident reports, operational monitoring logs, architecture reviews, deployment notes and post-implementation analyses from multiple production deployments. The analysis is not based on individual deployment examples, but on describing deployment practices that repeatedly proved to be successful in all application domains and in all organizational contexts. This experiential approach aligns with the recent trend of focusing on operational deployment, trustworthy implementation, and production governance in the rapidly changing LLM ecosystem as called for by Ozkaya (2023), Kenthapadi et al. (2024), and Patel et al. (2024).

It is necessary to explain the methodological process before turning to the nature of evidence in this study. A distinction must be made between the nature of the evidence used in this study and conventional experimental studies before the methodological process is presented. The paper does not make any statistical inferences on causality; rather it systematically integrates engineering knowledge gained from multiple production deployments. Table 1 is a summary of the main evidence sources that guided the practitioner observations throughout this paper.

Table 1 Sources of Practitioner Evidence Used in the Study

Evidence Source	Purpose in the Study	Contribution to Deployment Analysis
Production deployment retrospectives	Review completed production releases	Identify recurring engineering practices and deployment outcomes
Engineering incident reports	Examine operational failures	Reveal common deployment anti-patterns and recovery strategies
Architecture design reviews	Assess production architectures	Identify scalable architectural patterns
Operational monitoring dashboards	Evaluate runtime behaviour	Analyze latency, cost, reliability, and observability trends
Deployment documentation	Examine implementation workflows	Capture engineering decisions and deployment procedures
Post-deployment evaluations	Assess production performance	Validate recurring deployment lessons across environments
Anonymized organizational case observations	Compare multiple implementation contexts	Generalize practitioner experiences while preserving confidentiality

Source: Developed by the Author Based on Practitioner Deployment Observations and Informed by Production Engineering Principles Discussed by Ozkaya (2023), Yang et al. (2024), Patel et al. (2024), and Kenthapadi et al. (2024).

As shown in Table 1, it is a conscious effort to interweave a variety of operational evidence in this study rather than one specific deployment experience. This triangulation helps to enhance the analytical credibility by allowing recurring deployment patterns to be detected in the various engineering activities, such as architecture

design, incident management, and monitoring and production evaluation. This means it will not rely on isolated success stories, but on repeatable engineering practices to consistently promote production stability.

The analytical approach was iterative and structured in a pattern-recognition approach. Engineering observations were obtained from various deployments to look for commonalities in architectural decisions, operational workflows, deployment failures and recovery mechanisms. Any practice that was repeated and led to a stable production performance was counted as production pattern, if any engineering decision that was repeated and caused instability, cost, reliability, or poor user experience was counted as anti-pattern. This analytical approach is aligned with that of practitioners who suggest using practical experience to complement formal evaluation techniques for prompt engineering, deployment evaluation, and production governance (Fagbohun et al., 2024; Zamfirescu-Pereira et al., 2023; Denny et al., 2023).

The methodology also features empirical evidence from a variety of organizational settings, not limited to a single application area. The sample of production environments in the synthesized observations contains a variety of customer support automation, enterprise knowledge management, software engineering assistants, document processing, assistants within internal organizations, assistants within healthcare information systems and financial service applications. The chosen sectors were chosen as some of the fastest-growing production use cases for LLM technologies and also for their challenging demands for accuracy, security, scalability, governance, and operational resilience (Thirunavukarasu et al., 2023; Huang et al., 2023; Thukral et al., 2023). Analyzing several deployment contexts increases the likelihood of transferability of the engineering patterns identified because a lot of production issues stem from common features of the operation and not from any specific characteristics of a particular industry.

The methodological perspective also acknowledges that LLM systems for production are continually evolving after installation. Conventional software systems tend to have a somewhat fixed set of functions, but with production LLM applications, there is a need for continuous refinement of prompts, optimization of retrieval, infrastructure scaling, model upgrades, security audits, and API capability evaluations of the shifting capabilities. Recent research also found it is important to manage the models throughout their lifecycle, rather than simply deploying them once, especially as organizations evolve their datasets, user expectations, and model capabilities (Lee et al., 2024; Qian et al., 2024). As such, the observations coalesced in this paper are engineering practices that build up over time in the operational life cycle rather than isolated deployments.

To highlight the methodological process followed during the whole study, the practitioners' oriented analytical process to derive recurring production patterns and anti-patterns from operational deployment experiences is presented as Figure 1.

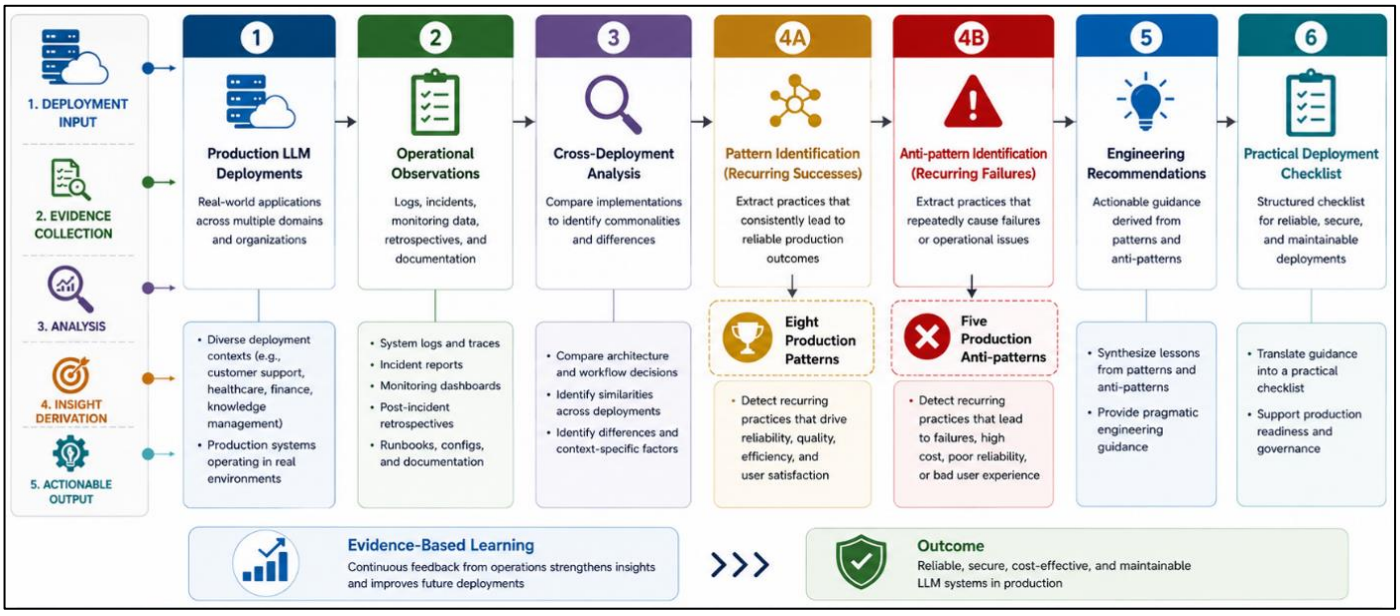

Fig 1 Practitioner-Oriented Methodological Workflow

Source: Developed by the Author Based on the Practitioner Methodology Synthesized from Deployment Engineering Practices Reported by Yang et al. (2024), Patel et al. (2024), Ozkaya (2023), Fan et al. (2023), and Kenthapadi et al. (2024).

In this study the practitioner-oriented workflow is shown in figure 1. From the real world deployment observations, a systematic analysis is performed to uncover production patterns and anti-patterns, which further guide the engineering recommendations and deployment checklist described in the second part of the paper. The aim of this workflow is to use the study to showcase what can be learned from real-world deployment practices to build robust, scalable and maintainable production LLM systems.

The presented paper is not a systematic literature search or a controlled empirical study but an experience report based on the experiences of practitioners. While there are several studies that provide in-depth knowledge of model architectures, prompting techniques, trustworthy AI, software engineering, security, testing, and application domains (Mohsin et al., 2024; Modalavalasa, 2022; Yu et al., 2023; Rusum, 2023; Bhat et al., 2024; Pangakis et al., 2023; Muhs & Stankowski, 2024; Rane et al., 2023), the guidance on production deployment is relatively sparse. This study's findings provide a synthesis of recurring operational experiences to create practical patterns and anti-patterns to help span the gap between academic research and real-world LLM deployment.

III. WHY PRODUCTION LLM SYSTEMS ARE DIFFERENT

Large Language Models (LLMs) have revolutionized the way organizations architect their intelligent software systems. The process of building a proof of concept or demonstration application is relatively simple, but deploying an LLM application to production is not a simple process that is limited to prompt design and choosing a model. Production environments need to be able to be available at all times, perform reliably, manage data securely, monitor operations, govern operations and be cost effective when handling dynamic user demands. While the behaviour of traditional software applications is mostly deterministic, the behaviour of LLM systems is probabilistic and can change based on the prompt, retrieved context, model changes, and user interaction. As a result, the deployment of the production system needs engineering expertise that combines software system, MLOps, DevOps, AI governance, and operational risk management to deliver reliable service (Fan et al., 2023; Ozkaya, 2023; Yang et al., 2024).

Previous studies have largely explored the capability of the model, and strategies, alignment, benchmarking and evaluation methods have been extensively studied (Hadi et al., 2023; Liu et al., 2023). But often organizations find that the successful prototype performance does not mean they can rely on it for production operation. When rolling out in the real world, the quality of the response, the infrastructure utilization, latency, compliance with regulations, user trust, and operational costs are all variables that need to be continuously weighed against. These competing requirements are a key factor in the difference between production LLM systems and experimental deployments, and also why the engineering maturity of the system is more important than the model's performance for deployment success (Patel et al., 2024; Kenthapadi et al., 2024).

To illustrate the key differences between prototype implementations and production-grade, LLM systems, Table 2 below summarizes the main differences.

Table 2 Comparison between Prototype and Production LLM Systems

Engineering Dimension	Prototype System	Production LLM System
Primary objective	Demonstrate functionality	Deliver reliable business value
Users	Limited testers	Thousands to millions of users
Infrastructure	Temporary or experimental	Highly available and scalable infrastructure
Monitoring	Minimal	Continuous observability and alerting
Security	Basic protection	Enterprise-grade security and compliance
Cost management	Rarely considered	Continuous token and infrastructure optimization
Prompt management	Manual experimentation	Version-controlled and validated
Failure recovery	Often ignored	Automated fallback and rollback mechanisms
Governance	Limited documentation	Auditing, compliance, and policy enforcement

Source: Developed by the Author Based on Production Engineering Concepts Synthesized from Ozkaya (2023), Yang et al. (2024), Patel et al. (2024), and Fan et al. (2023).

Table 2 shows that production systems are not just different in terms of models but also in several engineering aspects. Prototype applications focus on quick experimentation, whereas a production deployment must go through a systematic engineering process of creating an operational resilient, scalable, governable, and maintainable system. These differences are the reason engineering practices that are tailored to production AI systems must be adopted by organizations.

- *Reliability*

Reliability is among the most important features of production LLM systems because users always want to get the correct answers at the right time and on every occasion. While it may be acceptable to have a few mistakes in a research demonstration, in a production application, service needs to be reliable over an extended period of time. Reliability is thus defined as consistency of response, fault-tolerance, graceful degradation, recovery, and service availability (Ozkaya, 2023).

The probabilistic nature of LLM inference is one big risk on reliability. Stochastic decoding and/or model changes and/or context may sometimes lead to different outputs on the same prompt. This variation can be beneficial for some uses, as it can enhance creativity, but it can be problematic for consistent behavior in enterprise environments where predictable behavior is crucial. In recent years, engineering teams are more inclined to use version control, well-structured output validation, verifying the response, and automated regression testing to reduce unexpected behavioural variations (Fagbohun et al., 2024; Denny et al., 2023).

Systematic testing of responses generated before releasing them to end users also enhances reliability.

Hallucinated or misleading outputs are less likely to be introduced into operational workflows via automated verification techniques, schema validation, confidence scoring and human review mechanisms. Research looking at trustworthy deployment of LLM models also highlights the need for ongoing assessment rather than relying on a model's performance on benchmarks to ensure it acts reliably in the real world (Pangakis et al., 2023; Liu et al., 2023). Thus, reliability becomes not a characteristic of the model itself, but is the result of the well-designed engineering controls around the process of model inference.

➢ *Scalability*

To provide a scalable LLM system that can meet growing user needs while ensuring timely responses and service quality, production LLM applications must be scalable. Production deployments need to be able to handle multiple requests concurrently in a geographically distributed environment, while prototype systems are normally used by a few users. As more and more organizations adopt it, the infrastructure should be capable of supporting the increase in token volume, embedding, retrieval operations, and model inference without impacting performance (Yang et al., 2024).

Scalability can only be achieved with care in architectural planning. Organizations can handle more workloads while keeping their services going with cloud-native infrastructure, distributed request processing, asynchronous task execution, intelligent caching, vector databases, and load-balancing strategies. Another approach that enhances scalability is the Retrieval-Augmented Generation (RAG) architecture, which only retrieves the necessary context information, avoiding the need for unnecessary prompt increases and consequently lowering computational costs (Kenthapadi et al., 2024).

To illustrate and commence a discussion of the other engineering dimensions, an example of the main components that separate production LLM systems from prototype systems is shown in Figure 2.

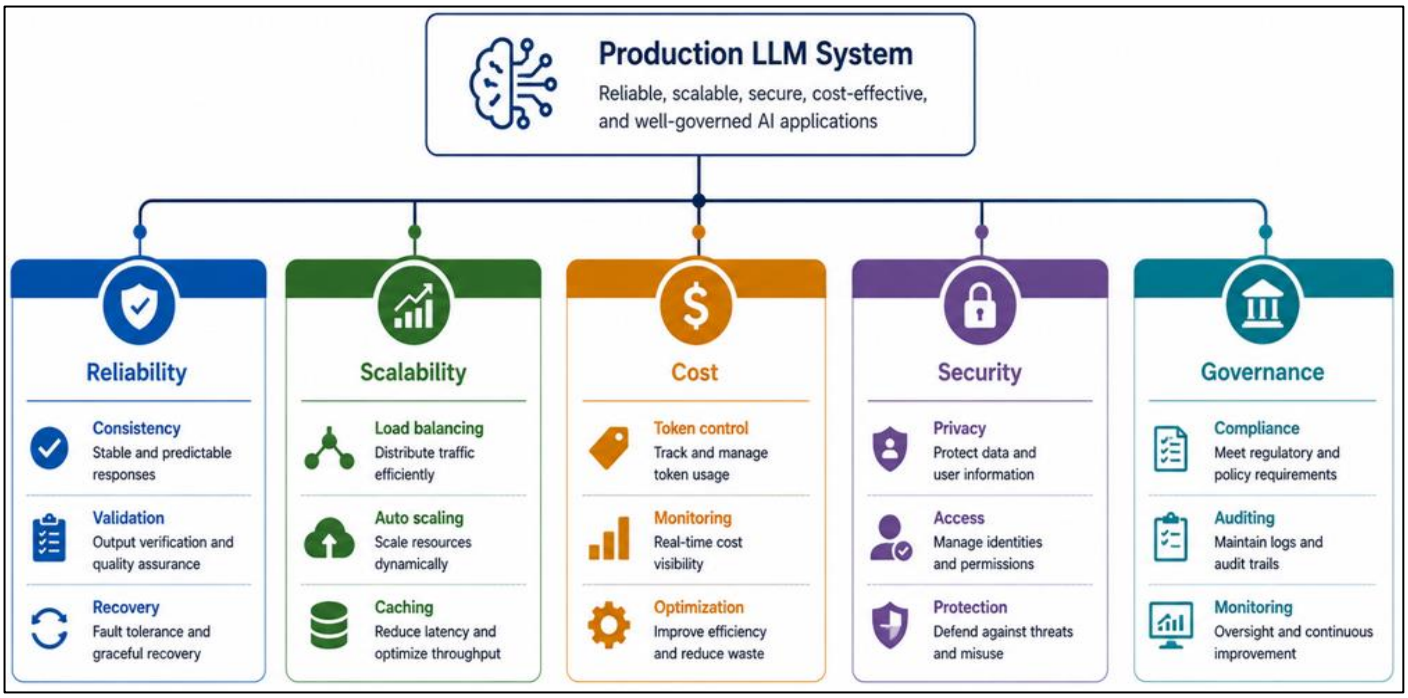

Fig 2 Core Engineering Dimensions of Production LLM Systems
Source: Developed by the Author Based on Production Deployment Principles Synthesized from Ozkaya (2023), Kenthapadi et al. (2024), Yang et al. (2024), and Patel et al. (2024).

Figure 2 shows that production readiness is as much about a combination of engineering features as it is about model accuracy. Reliability, scalability, cost management, security and governance work together to deliver dependable deployment over the whole lifecycle of a product.

➢ *Cost*

One of the main factors that differentiate production LLM systems from prototypes is the cost of operation. Experimental use will usually only be used for a small workload, whereas production deployments will have ongoing costs of storing and retrieving documents, using embedding's, using GPU resources and commercial API calls, as well as the cost of using the tokens. These costs can quickly mount up and threaten the sustainability of deployment if not managed effectively (Yang et al., 2024).

To meet this challenge, organizations must implement cost telemetry to monitor token usage and infrastructure utilization, as well as model usage. This transparency fosters quick optimization, smart model routing, caching strategies, and the informed optimization of costs and performance. The importance of cost monitoring during operations is highlighted by Patel et al. (2024), with efficient resource management being emphasized as a prerequisite for scalable AI systems by Modalavalasa (2022).

➢ *Security*

Security is essential in production LLM deployments since LLM systems often handle customer and organizational sensitive data. Prompt injection, malicious input, unauthorized access, and data leakage are threats that can have a significant impact on the production environment, potentially jeopardizing the security of an organization and the trust of its users (Mohsin et al., 2024).

In order to provide effective protection, multiple security features must be utilized, such as continuous monitoring, input validation, output filtering, encryption, authentication, and authorization. These controls can be woven into the deployment process to provide robust trustworthiness in the deployment of AI and aid in regulatory compliance, especially in sectors like healthcare and finance that are heavily regulated (Liu et al., 2023; Mohsin et al., 2024; Thirunavukarasu et al., 2023).

➢ *Governance*

Governance is responsible for ensuring that production LLM systems are transparent, responsible and compliant with organizational and regulatory requirements. Good governance not only covers the actual enactment of the project, but also encompasses timely project management, auditing, documentation, monitoring and oversight by people involved in the project throughout the project lifecycle (Lee et al. 2024).

Continuous accountability and improvement is enabled through the use of audit logs, monitoring deployment changes, assessing model performance, and reviewing emerging risks. Governance processes should be dynamic, adapting to the different models, datasets, APIs and regulatory expectations as they change in time, as highlighted by Qian et al. (2024) and Shankar et al. (2024) in the context of sustainable production deployment.

IV. EIGHT PROVEN PRODUCTION PATTERNS

The key to successful production deployment of Large Language Models (LLMs) is not to simply invest in the most powerful model, but to use engineering patterns that reliably, scalably, maintainable and operationally support LLM production. In production environments, there are recurring architectural practices that are not specific to a particular industry or application domain, but instead, they solve common deployment challenges including response consistency, infrastructure efficiency, governance, monitoring and operational risk. Current studies are showing that prompt engineering is not enough to build production-ready AI systems, and that there are a number of other engineering disciplines involved (Ozkaya, 2023; Yang et al., 2024; Patel et al., 2024). The next patterns are a summary of engineering practices that have been repeatedly seen to help ensure that the production got deployed reliably and were consistent with the goal to move academic knowledge to real deployment.

Table 3 provides a summary of the engineering function of each pattern in production environments before presenting each pattern in detail.

Table 3 Summary of Proven Production LLM Deployment Patterns

Pattern	Primary Engineering Purpose	Principal Deployment Benefit
Prompt Versioning	Control prompt evolution	Reproducibility and rollback
Retrieval-Augmented Generation (RAG)	Ground responses using trusted knowledge	Improved factual accuracy
Structured Output Validation	Verify generated responses	Reduced operational errors
Fallback Strategies	Maintain service continuity	Increased reliability
Cost Telemetry	Monitor operational expenditure	Cost optimization
Observability	Monitor system behaviour	Faster incident diagnosis
Progressive Rollouts	Deploy changes gradually	Reduced deployment risk
Human-in-the-Loop Validation	Incorporate expert oversight	Improved trust and governance

Source: Developed by the Author Based on Practitioner Deployment Experience and Production Engineering Concepts Synthesized from Ozkaya (2023), Kenthapadi et al. (2024), Patel et al. (2024), and Yang et al. (2024).

Table 3 shows that each deployment pattern serves a different engineering goal but all engineered towards production readiness. These practices are not standalone, but are complementary to each other and create robust, visible and sustainable LLM systems that can withstand real-world applications.

➢ *Pattern 1: Prompt Versioning*

Prompt versioning is to consider prompts as controlled software artifacts instead of static text. All modifications are recorded, recorded, tested, and given a version number for rollback and ease of reproduction. This method is effective because changes to the model often do change how the system behaves even if the application code does not change (Fagbohun et al., 2024). Prompts are kept in version-controlled repositories, along with automated pipelines for evaluating them, architecturally. The advantages are: traceability, collaborative development, ease of auditing and safer deployment. For instance, an enterprise knowledge assistant could deploy Prompt v2.3 following some regression tests, but keep v2.2 as the fallback version in case the response quality drops. Engineering teams should add prompts to their sources of control, keep data sets for evaluations, and record all modifications (Denny et al., 2023; Zamfirescu-Pereira et al., 2023).

➢ *Pattern 2: Retrieval Augmented Generation (RAG)*

Retrieval-Augmented Generation (RAG): It enhances factual grounding by incorporating external knowledge repositories along with LLM reasoning. Relevant documents are retrieved and used in the prompt as well to leverage pretrained knowledge, rather than just using it. This design minimizes hallucinations and enhances accuracy on specific tasks, such as enterprise search, customer support, and document-centric applications (Kenthapadi et al., 2024). Usual implementations are embedding models, vector databases, retrieval services and generation models. In an internal policy assistant, for example, that pulls the newest compliance documents prior to producing replies. Document quality, retrieval precision, chunk optimization and constant index maintenance should all be considered important in engineering practice in order to ensure the reliability of response.

➢ *Pattern 3: Structured Output Validation*

The output of production systems is often needed in a format other than natural language. Structured output validation requires a schema (like JSON or XML) to be applied before generated outputs can be accepted by downstream applications. The validation step ensures that the output data is correct and not malformed, which helps avoid the disruptions of automated processes (Shankar et al., 2024). In terms of architecture, validators act as an immediate follow-up to the model inference and will initiate retries or fallback if the outputs don't align with pre-established schemas. For example, a financial reporting application might block transaction summaries from being added until certain fields in the summary are complete to meet validation requirements. The engineering recommendations are to enforce schemas, automate parsing, verify confidence and log all errors (Pangakis et al., 2023).

➢ *Pattern 4: Fallback Strategies*

Fallback strategies ensure service availability in case of failure of the model inference, retrieval systems or external APIs. There are a number of alternative models, cached responses, rule-based processing and human escalation pathways that are commonly implemented in production architectures. This pattern works well because it helps to reduce the disruption to services while maintaining good user experience (Patel et al., 2024). To change from a large backup to a smaller backup during primary API outage periods, until normal operation returns, a customer support assistant can temporarily switch back to the smaller backup. Engineering teams must establish clear and concrete fall back situations, keep track of how often they are activated and periodically test recovery processes.

Figure 3 provides an overview of how the principal engineering patterns fit into a production LLM architecture, before taking a look at the other deployment patterns.

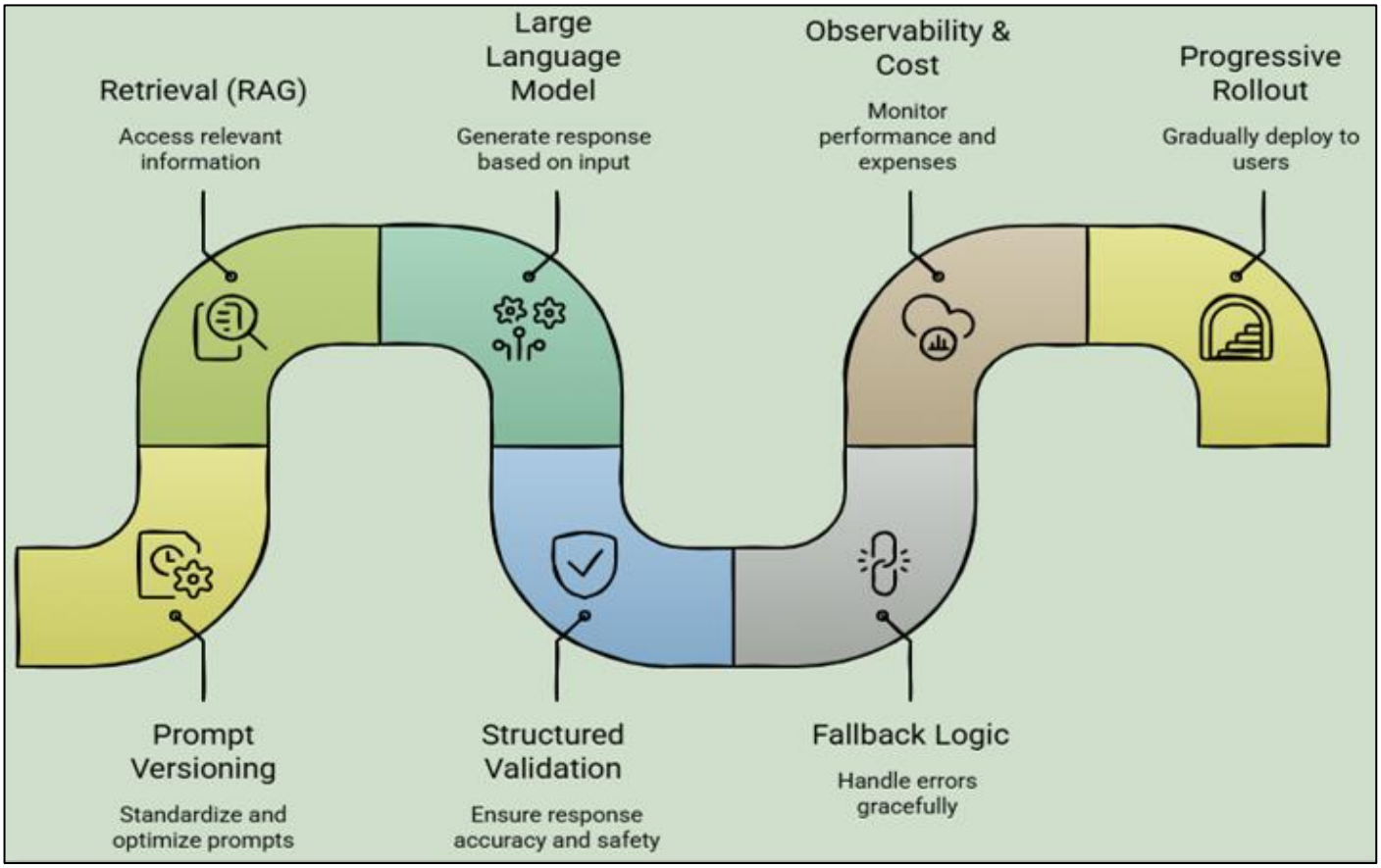

Fig 3 Integrated Production LLM Deployment Patterns

Source: Developed by the Author Based on Practitioner Deployment Architecture Synthesized from Yang et al. (2024), Kenthapadi et al. (2024), Ozkaya (2023), and Patel et al. (2024).

Figure 3 shows that production deployment is not a linear inference process, but a complex engineering process, and the validation, monitoring, governance and operation safeguards all enhance the robustness of the system. Every pattern helps to minimize risk of deployment and increase production resilience.

- *Pattern 5: Cost Telemetry*

Cost telemetry monitors costs of token usage, API usage, embedding generation, infrastructure usage, and inference costs in real time. Prototype systems need a level of financial visibility to run over time, which is not the case with production systems. Dashboards and automated alerts help engineering teams alert them to unusual spending habits and help them optimize prompt design or routing before costs seem to surge (Yang et al., 2024). Institutions should set usage limits, have departmental usage reports, and conduct regular audits of the usage costs to make informed decisions about operation.

- *Pattern 6: Observability*

Observability includes metrics like logging, distributed tracing, latency, prompt analytics, retrieval metrics, and response quality monitoring to give a comprehensive view of the behaviour of production. This pattern helps quickly identify failures that are not detectable by the traditional infrastructure monitoring approach (Shankar et al., 2024). For instance, a production coding assistant can identify longer latencies due to suboptimal retrieval, not just model inference itself. Infrastructure metrics and AI-specific operational metrics should work together to enable continuous improvement and proactive incident response in engineering.

- *Pattern 7: Progressive Rollouts*

Progressive rollout strategies include canary releases, feature flags, staged deployments and A/B testing. Production behaviour is tested under controlled conditions first, before it is rolled out to all users, so as to generate information about it. Engineering teams test production behaviour with a small number of users before rolling it out to the rest of the users, in order to gain information from it. This design reduces the risk of operations and enables informed deployment decisions (based on evidence) (Patel et al., 2024). A new prompt version might only support five percent of users at first, and only be rolled out if monitoring shows that the new version is performing well with respect to quality, latency, and costs. Each deployment stage should be accompanied by a continuous evaluation to guarantee that it is deployed safely.

- *Pattern 8: Human-in-the-Loop Validation*

Human in the loop validation is the process of bringing together expert validation to decisions that demand a high level of accuracy, regulatory compliance or organizational accountability. While LLMs can help with numerous tasks, human oversight is crucial in situations where their mistakes can have large operational implications. Architecturally, outputs beyond a pre-stated confidence level automatically go into the "yes" category, while those that are uncertain or are of high risk are sent for expert review. This pattern can significantly enhance reliability and trust in various sectors, such as healthcare documentation and financial compliance reporting, as demonstrated in these practical examples (Thirunavukarasu et al., 2023; Mohsin et al., 2024). Engineering teams should have clear criteria to escalate to, processes to review feedback and ongoing learning cycles to ensure that human intelligence can be used to augment automated intelligence, not supplant it.

V. FIVE COMMON ANTI-PATTERNS

Successful production deployments have common engineering practices, though they also have common errors, which degrade reliability, add risk for operations, and erode user confidence. These anti-patterns are not just isolated incidents of bad engineering decisions, but are patterns that have been seen in multiple production environments. They frequently appear when an organization is rushing to deploy a solution over following best practices or believing that good benchmark performance is a sure indicator of production readiness. It is important to know these anti-patterns as well as adopting deployment patterns, since avoiding common deployment mistakes can significantly enhance long-term system stability and maintainability (Ozkaya, 2023; Yang et al., 2024; Patel et al., 2024). Table 4 summarizes the characteristics, operational consequences, and recommended mitigations for each anti-pattern, prior to discussing them each in turn.

Table 4 Common Production LLM Anti-Patterns and Mitigation Strategies

Anti-pattern	Primary Consequence	Recommended Mitigation
Prompt Sprawl	Inconsistent model behaviour	Version-controlled prompt repository
Blind Trust in Model Outputs	Hallucinations and inaccurate decisions	Output validation and human review
Ignoring Operational Cost	Escalating infrastructure expenditure	Continuous cost telemetry and optimization
Weak Monitoring	Delayed failure detection	Comprehensive observability and alerting
Deploying Without Rollback	Extended production outages	Rollback plans and staged deployments

Source: Developed by the Author Based on Practitioner Deployment Observations and Supported by Ozkaya (2023), Kenthapadi et al. (2024), Patel et al. (2024), Yang et al. (2024), and Shankar et al. (2024).

Table 4 shows that the breakdown of engineering governance is a problem, not the language modelling problem. The corresponding countermeasures emphasize the need to have disciplined operational procedures during the deployment.

- *Anti-Pattern 1: Prompt Sprawl*

Sprawl is when prompts are changed and distributed without versioning and centralized management. The number of prompts increases as LLM applications scale up, causing them to produce different outputs within teams, poor output reproducibility, and challenging debugging (Fagbohun et al., 2024). Typical issues are duplicate prompts, unrecorded changes and inconsistent versions. The primary problem is that prompting is not regarded as a software asset. Version Control, Documentation, Regression Testing, and Structured Review prior to deployment (Denny et al., 2023; Zamfirescu-Pereira et al., 2023) are classified as mitigation.

- *Anti-Pattern 2: Blind Trust in Model Outputs*

Blind trust means that a team does not need to verify the results from the LLM. Yet, factual inaccuracies, logical fallacies, and hallucinations are prevalent, and both operational and financial risk and reputational risks can occur (Liu et al., 2023). Issues arise such as incorrect recommendations, incorrect decision support, especially in healthcare and finance (Thirunavukarasu et al., 2023). Schema validation, retrieval grounding, confidence scoring and human review of high-risk responses (Pangakis et al., 2023) should be considered for effective mitigation.

- *Anti-Pattern 3: Neglecting to Consider Operational Cost*

Another factor that is not considered when prototyping but rapidly contributes to the operational costs during the production phase is token consumption, API calls, embeddings, and infrastructure utilization (Yang et al., 2024). The symptoms are increased cloud cost as well as inefficient resource usage due to poor financial tracking. To ensure sustainable deployment, organizations need to use cost telemetry, optimize costs promptly, model routing, cache, and regular cost reviews (Patel et al., 2024).

- *Anti-Pattern 4: Weak Monitoring*

When monitoring is weak, it can be hard to understand the behaviour of the model, the time delay between the model making a prediction and the response, the quality of the predictions retrieved, and the health of the system (Shankar et al., 2024). Common symptoms are slow detection of incident, poor response, and long recovery. Comprehensive observability is essential for mitigation, including logging, tracing, timely analytics, latency metrics, retrieval performance and operational dashboards (Kenthapadi et al., 2024).

- *Anti-Pattern 5: Deploying without Rollback*

When updates cannot be reverted, deployment can lead to a considerable amount of risk during operation. Business disruption and long outages can occur when prompt, model or infrastructure changes are unsuccessful (Patel et al., 2024). Symptoms are slower healing and failure to replace stable versions. Version controlled releases, feature flags, staged releases, automated rollback processes, and frequent recovery testing are all part of the solution for organizations to restore services quickly in the event of an incident (Ozkaya, 2023).

VI. ENGINEERING RECOMMENDATIONS

Disciplined engineering is more important for successful production LLM deployment than model capability. Prompts should be treated as code, and should be version controlled, documented, tested, and released in a controlled way (Fagbohun et al., 2024). They should also document everything latency, token utilizations, retrieval performance and costs to help them make informed decisions (Shankar et al., 2024). Observability will help to identify production issues early and solve them quickly from the initial day of building (Kenthapadi et al., 2024). A graceful degradation with fallback models, cached responses and human escalation should also be designed to ensure the continuity of the services (Patel et al., 2024). All of the answers generated must be rigorously checked by structured schemas, retrieval grounding, confidence assessment and expert checks as appropriate (Liu et al., 2023; Pangakis et al., 2023). Moreover, organizations must consider Retrieval-Augmented Generation (RAG) only if they have to make a trade-off between accuracy and simplicity in architecture (Kenthapadi et al., 2024). Last but not least, the engineering teams need to be capable of rollbacking their systems prior to deployment and continuously monitor the performance of their production systems based on operational metrics, incident reviews, and user feedback to ensure reliable, secure, and cost-effective LLM systems (Ozkaya, 2023; Yang et al., 2024).

VII. PRACTICAL DEPLOYMENT CHECKLIST

This study's results suggest that there's a need for rigorous engineering practices to be applied consistently across the life of the production system for successful deployment of Large Language Models. Organizations should define standard deployment processes that enhance the reliability, security, maintainability, and operational resilience of its deployments, beyond just ensuring the model is capable. A deployment checklist is a well-defined way for engineering teams to ensure that their production is ready for new models, prompts, or architectural changes. This type of checklists has become essential for software quality assurance, as they help maintain consistency, minimize deployment risks, and enable ongoing operational assessments (Patel et al., 2024; Ozkaya, 2023). The practical deployment checklist presented in Table 5, which was synthesized from the recurring production patterns, emerged from this study.

Table 5 Practical Production LLM Deployment Checklist

Deployment Requirement	Verification Objective	Status (✓/✗)
Prompt version control	Prompts are version-controlled, documented, and regression tested	☐
Structured output validation	Generated outputs conform to predefined schemas before execution	☐
Cost monitoring	Token usage, API expenditure, and infrastructure costs are continuously monitored	☐
Logging and tracing	Prompt execution, latency, retrieval, and inference logs are recorded	☐
Fallback mechanisms	Backup models, cached responses, or human escalation paths are available	☐
Security controls	Authentication, authorization, encryption, and input validation are implemented	☐
Human review	High-risk responses are reviewed through defined approval workflows	☐
Canary deployments	New prompts or models are released incrementally before full deployment	☐
Rollback procedures	Previous prompt, model, and configuration versions can be restored immediately	☐
Continuous evaluation	Production performance is regularly assessed using operational and quality metrics	☐

Source: Developed by the Author Based on Practitioner Deployment Experience and Informed by Ozkaya (2023), Patel et al. (2024), Kenthapadi et al. (2024), and Yang et al. (2024).

Table 5 summarizes the engineering principles that have been discussed throughout this paper and how they can be translated into an operational readiness checklist that organizations can incorporate into deployment workflows. This process of conformance prior to production release helps engineering teams avoid deployment failures, improves governance, and provides a repeatable approach to building reliable, trustworthy LLM applications.

VIII. LIMITATIONS

The scope of the method used must be taken into consideration when interpreting this study. Findings are based largely on practitioner experience, engineering observation and deployment retrospectives, and operational lessons learned, rather than a systematic empirical investigation and/or a controlled experiment. As a result, the production patterns and anti-patterns identified focus on engineering knowledge rather than statistically proven causal relationships. In addition, the examples of deployment throughout the paper are anonymized to ensure that the organizations are not identified and are thus not necessarily representative of the diversity of production in various industries and organizational settings. Lastly, LLM technologies and deployments are evolving rapidly, and there may be engineering recommendations that will need to be adapted as new LLM models, deployments, and governance evolve over time (Yang et al., 2024; Patel et al., 2024).

IX. FUTURE WORK

Future studies should be directed towards the creation of standardized deployment metrics beyond the traditional model accuracy measures, which also test performance, as a measure of production readiness. Further research is required on automated prompt versioning systems to enable continuous integration and deployment (CI/CD) processes, while still being reproducible and governable. Another research avenue is the study of cost-aware orchestration frameworks that dynamically leverage the selection of models, retrieval strategies, and the use of infrastructure. The growing complexity of production ecosystems will need further research into multi-agent production architectures and interoperable AI governance frameworks, to enhance coordination, accountability, and regulatory compliance. Lastly, improvements in autonomous deployment optimization could allow production systems to dynamically adapt prompts, routing, monitoring thresholds, and infrastructure configurations based on evolving operational conditions to enhance their long-term deployment success and efficiency (Kenthapadi et al., 2024; Yang et al., 2024; Ozkaya, 2023).

X. CONCLUSION

As LLM adoption grows in various domains like software engineering, healthcare, finance, education, enterprise Knowledge Management, and customer service, it has been evident that producing application deployments based on model capabilities is not enough. Despite the recent progress in foundation models, there are still significant hurdles in the way of taking a prototype and scaling it into a reliable system for use in the real world. Existing studies have significantly advanced the understanding of model architectures, techniques, alignment, evaluation and application domains (Fan et al., 2023; Hadi et al., 2023; Yang et al., 2024). But, there has been relatively little focus on the engineering practices needed to maintain the reliability of production, operational resiliency, governance, observability and maintainability over the long term. This discrepancy was the main reason for the present study.

Based on practitioner experience from production deployments, this paper has introduced a field guide that focuses on engineering knowledge gained from recognizable patterns of production deployment rather than on theoretical model building. The analysis singled

out eight patterns that are consistently effective to facilitate reliable deployment: Prompt Versioning, Retrieval-Augmented Generation (RAG), Structured Output Validation, Fallback Strategies, Cost Telemetry, Observability, Progressive Rollouts, and Human in the Loop Validation. These patterns collectively highlight the crucial importance of embedding software engineering practices, operational governance, and ongoing monitoring throughout the AI lifecycle. These results further support the recent claims of the need for disciplined engineering processes along with model capability advancement to ensure trustworthy and sustainable LLM applications (Kenthapadi et al., 2024; Ozkaya, 2023; Patel et al., 2024).

Just as important, the paper revealed five anti-patterns that seem to appear over and over again and lead to failures in production. Enhanced monitoring and cost management, and implementation with rollback mechanisms, were demonstrated to increase reliability, reduce operational risk, and build confidence among users. Blind dependence on model outputs, prompt sprawl, lack of rollback and weak monitoring were identified as reducing reliability, increasing operational risk, and destroying user confidence. These anti-patterns show that the problems with production LLM use are not its fault, but because key engineering controls are missing or not used. Therefore, production deployment is not just a deployment exercise but an ongoing engineering process that needs governance, validation and monitoring, and iterative improvement. The findings resonate with the general themes on trustworthy AI, responsible implementation, and software quality assurance highlighted in the literature (Liu et al., 2023; Mohsin et al., 2024; Shankar et al., 2024).

REFERENCES

[1]. Bhat, A., Shrivastava, D., & Guo, J. L. (2024, July). Do LLMs meet the needs of software tutorial writers? Opportunities and design implications. In Proceedings of the 2024 ACM Designing Interactive Systems Conference (pp. 1760–1773). https://doi.org/10.1145/3643834.3660692

[2]. Denny, P., Leinonen, J., Prather, J., Luxton-Reilly, A., Amarouche, T., Becker, B. A., & Reeves, B. N. (2023). Promptly: Using prompt problems to teach learners how to effectively utilize AI code generators. arXiv preprint arXiv:2307.16364. https://doi.org/10.48550/arXiv.2307.16364

[3]. Fagbohun, O., Harrison, R. M., & Dereventsov, A. (2024). An empirical categorization of prompting techniques for large language models: A practitioner's guide. arXiv preprint arXiv:2402.14837. https://doi.org/10.48550/arXiv.2402.14837

[4]. Fan, A., Gokkaya, B., Harman, M., Lyubarskiy, M., Sengupta, S., Yoo, S., & Zhang, J. M. (2023, May). Large language models for software engineering: Survey and open problems. In 2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE) (pp. 31–53). IEEE. https://ieeexplore.ieee.org/document/10449667

[5]. Hadi, M. U., Qureshi, R., Shah, A., Irfan, M., Zafar, A., Shaikh, M. B., ... & Mirjalili, S. (2023). Large language models: A comprehensive survey of its applications, challenges, limitations, and future prospects. Authorea Preprints, 1(3), 1–26. https://doi.org/10.36227/techrxiv.23589741.v4

[6]. Huang, H., Zheng, O., Wang, D., Yin, J., Wang, Z., Ding, S., ... & Shi, B. (2023). ChatGPT for shaping the future of dentistry: The potential of multi-modal large language model. International Journal of Oral Science, 15(1), 29. https://doi.org/10.1038/s41368-023-00239-y

[7]. Kenthapadi, K., Sameki, M., & Taly, A. (2024, August). Grounding and evaluation for large language models: Practical challenges and lessons learned (survey). In Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (pp. 6523–6533). https://doi.org/10.1145/3637528.3671467

[8]. Lee, J., Hicke, Y., Yu, R., Brooks, C., & Kizilcec, R. F. (2024). The life cycle of large language models in education: A framework for understanding sources of bias. British Journal of Educational Technology, 55(5), 1982–2002. https://doi.org/10.1111/bjet.13505

[9]. Liu, Y., Yao, Y., Ton, J. F., Zhang, X., Guo, R., Cheng, H., ... & Li, H. (2023). Trustworthy LLMs: A survey and guideline for evaluating large language models' alignment. arXiv preprint arXiv:2308.05374. https://doi.org/10.48550/arXiv.2308.05374

[10]. Modalavalasa, G. (2022). LARGE LANGUAGE MODELS FOR INTELLIGENT DATA ENGINEERING: AUTOMATING SCHEMA DESIGN, LINEAGE, AND QUALITY CONTROL. Power System Protection and Control, 50(2), 25–36. https://pspac.info/index.php/dlbh/article/view/183

[11]. Mohsin, A., Janicke, H., Wood, A., Sarker, I. H., Maglaras, L., & Janjua, N. (2024). Can we trust large language models generated code? A framework for in-context learning, security patterns, and code evaluations across diverse LLMs. arXiv preprint arXiv:2406.12513. https://doi.org/10.48550/arXiv.2406.12513

[12]. Muhs, N., & Stankowski, A. (2024). Leveraging LLMs for Reflection: Approaches to Mitigate Assumptions within the Design Process. https://doi.org/10.21606/drs.2024.1367

[13]. Ozkaya, I. (2023). Application of large language models to software engineering tasks: Opportunities, risks, and implications. IEEE Software, 40(3), 4–8. https://ieeexplore.ieee.org/document/10109345

[14]. Pangakis, N., Wolken, S., & Fasching, N. (2023). Automated annotation with generative AI requires validation. arXiv preprint arXiv:2306.00176. https://arxiv.org/abs/2306.00176

[15]. Patel, H., Boucher, D., Fallahzadeh, E., Hassan, A. E., & Adams, B. (2024). A State-of-the-practice

Release-readiness Checklist for Generative AI-based Software Products. arXiv preprint arXiv:2403.18958. https://doi.org/10.48550/arXiv.2403.18958

[16]. Qian, C., Reif, E., & Kahng, M. (2024, May). Understanding the dataset practitioners behind large language models. In Extended Abstracts of the CHI Conference on Human Factors in Computing Systems (pp. 1–7). https://doi.org/10.1145/3613905.3651007

[17]. Rane, N. L., Tawde, A., Choudhary, S. P., & Rane, J. (2023). Contribution and performance of ChatGPT and other Large Language Models (LLM) for scientific and research advancements: A double-edged sword. International Research Journal of Modernization in Engineering Technology and Science, 5(10), 875–899. https://www.academia.edu/107992803/Contribution_and_performance_of_ChatGPT_and_other_Large_Language_Models_LLM_for_scientific_and_research_advancements_a_double_edged_sword

[18]. Rusum, G. P. (2023). Large Language Models in IDEs: Context-Aware Coding, Refactoring, and Documentation. International Journal of Emerging Trends in Computer Science and Information Technology, 4(2), 101–110. https://doi.org/10.63282/3050-9246.IJETCSIT-V4I2P110

[19]. Shankar, S., Li, H., Asawa, P., Hulsebos, M., Lin, Y., Zamfirescu-Pereira, J. D., ... & Wu, E. (2024). Spade: Synthesizing data quality assertions for large language model pipelines. arXiv preprint arXiv:2401.03038. https://doi.org/10.48550/arXiv.2401.03038

[20]. Thirunavukarasu, A. J., Ting, D. S. J., Elangovan, K., Gutierrez, L., Tan, T. F., & Ting, D. S. W. (2023). Large language models in medicine. Nature Medicine, 29(8), 1930–1940. https://doi.org/10.1038/s41591-023-02448-8

[21]. Thukral, V., Latvala, L., Swenson, M., & Horn, J. (2023). Customer journey optimisation using large language models: Best practices and pitfalls in generative AI. Applied Marketing Analytics, 9(3), 281–292. https://www.ingentaconnect.com/content/hsp/ama/2023/00000009/00000003/art00008#Refs

[22]. Vankayala, S. C. (2023). LLM augmented exploratory testing: A framework for intelligent risk discovery, hypothesis generation, and cognitive enhancement in software quality engineering. International Journal of Science, Engineering and Technology, 11(1). https://www.ijset.in/wp-content/uploads/IJSET_V11_issue1_357.pdf

[23]. Yang, J., Jin, H., Tang, R., Han, X., Feng, Q., Jiang, H., ... & Hu, X. (2024). Harnessing the power of LLMs in practice: A survey on ChatGPT and beyond. ACM Transactions on Knowledge Discovery from Data, 18(6), 1–32. https://dl.acm.org/doi/10.1145/3649506

[24]. Yu, S., Fang, C., Ling, Y., Wu, C., & Chen, Z. (2023, October). LLM for test script generation and migration: Challenges, capabilities, and opportunities. In 2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security (QRS) (pp. 206–217). IEEE. https://doi.org/10.1109/QRS60937.2023.00029

[25]. Zamfirescu-Pereira, J. D., Wong, R. Y., Hartmann, B., & Yang, Q. (2023, April). Why Johnny can't prompt: How non-AI experts try (and fail) to design LLM prompts. In Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (pp. 1–21). https://dl.acm.org/doi/full/10.1145/3544548.3581388