International Journal of Scientific Engineering and Science
Volume 9, Issue 5, pp. 178-181, 2025. ISSN (Online): 2456-7361

The Impact of Configuration Management on the Reliability of LLM Pipelines

Nikita Gladkikh
Staff Software Engineer, Primer AI
Pittsburgh, Pennsylvania, USA
nikita.gladkikh@primer.ai

Abstract *— The article examines the impact of configuration management on the reliability of multi-stage pipelines based on large language models (LLM). A minimalistic approach using the PKonfig library, a "zero—deps" solution for strict validation, typing, and support of multiple configuration sources, is proposed and justified. Through comparative analysis with pydantic-settings, Kubernetes ConfigMaps and HashiCorp Vault, the advantage of lightweight management is shown to reduce startup time, increase reproducibility and security. The study is based on a comparative analysis of other studies, which made it possible to broadly consider the impact of configuration management on the reliability of LLM pipelines. The paper proposes a methodology for quantifying the impact of configuration management on LLM pipeline reliability metrics. The value of the results presented in the article is obvious to researchers and developers who are engaged in scaling and supporting pipelines of large language models, where strict version control of configurations directly correlates with the predictability and reproducibility of results. In addition, the data will be in demand by software reliability specialists and system architects in large IT companies and research centers focused on increased resilience and fault tolerance of critical AI infrastructures.*

Keywords *— Configuration management, reliability, multi-stage LLM pipelines, PKonfig, reproducibility, AI applications, minimalistic approach, performance.*

I. Introduction

The application of large language models (LLMs) to monitoring, analysis of unstructured documents, and decision-making automation imposes stringent requirements on the configuration layer: rapid scalability, support for A/B experiments, strict environment isolation, and robust secret management [1]. At the same time, solutions based on Kubernetes ConfigMaps and HashiCorp Vault introduce operational overhead and complicate CI/CD workflows [4, 6, 5].

Contemporary research on the impact of configuration management on the reliability of LLM pipelines can be provisionally grouped into several thematic areas. Several studies have shown that LLMs can effectively augment traditional data-pipeline monitoring systems. For example, Mangaonkar M. and Penikalapati V. K. [1] propose integrating large language models into existing data-flow observability frameworks, using them to automatically annotate anomalies and generate recommendations for correcting inaccurate or missing data during collection. The authors highlight the potential of fine-tuning LLMs on internal logs to improve anomaly-detection accuracy and describe a hybrid platform architecture in which LLMs act as “intelligent agents” interacting with alerting and visualization systems.

A directly related aspect of pipeline continuity and resilience—the automatic detection and remediation of container-configuration errors—is addressed in LLMSecConfig. Ye Z., Le T. H. M., and Babar M. A. [2] demonstrate an approach in which a pre-trained language model is further trained or adapted via prompt engineering on a corpus of misconfiguration scenarios, enabling it to generate corrective patches for Docker and Kubernetes manifests. Their evaluation on real-world scenarios drawn from open-source repositories shows a reduction in deployment vulnerabilities.

At the same time, systematic reviews of container-security issues point to a broad spectrum of risks, with misconfiguration being only one critical element. Sultan S., Ahmad I., and Dimitriou T. [3] summarize the consequences of misconfigured network policies, access controls, and secrets, and propose a comprehensive threat matrix along with mitigation methods, including continuous auditing and penetration testing. Al Mashta L. [4], in turn, conducts a systematic literature review of architectural strategies for container protection, focusing on automated configuration-control mechanisms and best practices for DevSecOps teams.

Turning to data preparation and software security, Le T. H. M. and Babar M. A. [7] investigate the potential of automated data labeling for vulnerability-prediction models, emphasizing weakly supervised and unsupervised learning approaches in which LLMs extract features from source code to create “soft” labels that are then converted into training sets for classical machine-learning algorithms. The authors analyze the error characteristics of this approach and underscore the critical role of corpus quality and label verification.

In the area of enhancing retrieval-augmented generation (RAG) architectures, Yan S. Q. et al. [5] introduce a Corrective Retrieval Augmented Generation mechanism. After an initial retrieval of contextual fragments from an external repository, the LLM produces draft responses, then selectively “pulls in” additional sources for topics where factual inaccuracies or logical inconsistencies have been detected, yielding a final output that incorporates these corrections. Although this iterative process improves fidelity and coherence, it imposes additional demands on pipeline architecture and state management.

Wang Y. et al. [8] study the influence of the temperature hyperparameter on LLM reasoning performance in graph-structured tasks—ranging from routing to centrality-metric computation. Through a series of experiments, they show that lower temperature values enhance accuracy in deterministic computations, whereas higher values lead to more diverse but less structured outputs, which can undermine reproducibility in pipelines requiring deterministic results.

Finally, practical engineering reports and blog posts offer lessons learned from building and operating AI systems. The article "7 Lessons from building a small-scale AI application" [9] on TheLis website emphasizes the importance of a rational strategy for dependency and configuration management in local development environments, as well as the integration of CI/CD pipelines with containerization and orchestration. "AI's Privilege Expansion" [10] on DigitalNative examines the risks of overprivileged service accounts and recommends the principle of least privilege. "Building AI Products—Part I: Back-end Architecture" [11] on PhilCalcado underscores the necessity of strict versioning for models and configuration files and describes microservice architectural patterns that ensure reliability and scalability. Finally, "The Bitter Lesson" [12] on AnkitMaloo serves as a reminder that, in the long run, simple, scalable algorithms often beat sophisticated systems that need manual configuration tuning, a conclusion that should inform engineering priorities.

Collectively, these works demonstrate a diverse range of approaches to ensuring LLM-pipeline reliability via configuration management: from deep integration of models into monitoring and self-healing processes to the development of organizational practices and architectural principles. Yet contradictions persist: some authors contend that LLM-based solutions can serve as the keystone of self-healing systems, while others warn of reproducibility and security risks associated with automated configuration fixes. Moreover, questions remain underexplored, including the scalable verification of LLM-generated corrective proposals in production, the interaction between RAG methods and secret-management systems, and empirical studies of hyperparameter effects in real-world industrial pipelines.

This study aims to determine how the application of minimalist configuration management using PKonfig influences the reliability, reproducibility, and security of multi-stage LLM pipelines, and to empirically demonstrate these effects.

The scientific novelty lies in formulating and substantiating a minimalist "zero-deps" approach to configuration management for multi-stage LLM pipelines based on PKonfig and developing a methodology for quantitatively analyzing the impact of configuration management on key reliability, reproducibility, and security metrics, validated through industrial and educational case studies.

The central hypothesis is that a minimalist, strictly typed approach to configuration management (PKonfig) reduces the incidence of configuration-related failures, enhances reproducibility, and improves the security of multi-stage LLM pipelines without significantly increasing service-startup times.

The research is based on a comparative analysis of existing studies in this domain, thereby offering a comprehensive view of the impact of configuration management on LLM-pipeline reliability.

II. Configuration Management in LLM Pipelines

Modern AI applications built on large language models impose highly formalized requirements on the configuration layer. First, they demand rapid deployment cycles and full A/B testing capability, enabling prompt adjustment of inference-pipeline parameters and comparison of alternative model versions [1]. Second, ensuring reproducibility of multi-stage scenarios with cascading LLM calls is essential: any divergence in results upon rerun—commonly referred to as "drift"—is unacceptable in production environments [3]. In addition, secure handling of secrets—API tokens, encryption keys, sensitive data—must incur minimal leak risk even under intensive horizontal scaling [4]. Finally, serverless architectures and orchestrators such as Kubernetes or Nomad enforce stringent limits on cold-start latency and memory footprint, directly impacting cost-effectiveness [2].

Widely used configuration-management tools only partially address these needs and exhibit systemic shortcomings. For example, pydantic-settings—a popular Python library for typed configurations—provides strict validation and IDE autocomplete yet increases cold-start time due to heavy Pydantic and typing-extensions dependencies [10]. Kubernetes ConfigMaps, the native mechanism for storing YAML files in-cluster, allow dynamic parameter updates but lack application-level static typing and validation, and require a full Kubernetes infrastructure [11]. HashiCorp Vault addresses secret-management needs with encryption and key rotation, but at the cost of an extra service, infrastructure maintenance, and often paid licensing [12].

To overcome these limitations, the lightweight library PKonfig was developed as a "zero-deps" alternative to pydantic-settings. It implements fail-fast validation of required parameters at initialization, preventing late failures when first accessing the LLM. Configurations can be prioritized from environment variables or files in YAML, JSON, TOML, or INI formats without additional code. Explicit typing via primitives Str, Int, Choice, and LogLevel ensures correct parsing, static analysis, and IDE support without Pydantic. An alias-group mechanism simplifies migration across cloud environments by declaring equivalent environment variables for different providers. With no external dependencies, the base package is approximately 15 kB—crucial for serverless functions constrained, for example, to 50 MB in AWS Lambda. Finally, an extensible API allows new configuration sources—Secrets Manager, Consul KV, and others—to be added without modifying the core library [1, 2].

In summary, configuration management in LLM pipelines must simultaneously support high-velocity deployment and A/B testing, strictly deterministic reproducibility of cascading inference scenarios, secure secret handling, and minimal cold-start resource consumption in serverless contexts. Existing solutions—ranging from pydantic-settings and HashiCorp Vault to native Kubernetes ConfigMaps—either lack full

International Journal of Scientific Engineering and Science
Volume 9, Issue 5, pp. 178-181, 2025. ISSN (Online): 2456-7361

application-level validation and static typing or impose significant cold-start, dependency, and operational burdens. The lightweight PKonfig library demonstrates that a zero-deps approach with fail-fast initialization validation, prioritized loading from environment and diverse file formats, explicit type primitives, and a flexible alias-group system can preserve correctness and reproducibility without Pydantic-induced cold-start delays, while its extensible API paves the way for Secrets Manager, Consul, and other back-end integrations without expanding the core library—making it an optimal platform for latency- and security-sensitive LLM applications.

TABLE 1. Comparison of configuration-management solutions in AI environments [1, 2, 3, 5, 6]

Feature	PKonfig	Pydantic-settings	Kubernetes ConfigMaps	HashiCorp Vault
Dependencies	none	Pydantic, typing-extensions	Kubernetes cluster required	dedicated service
Parameter validation	fail-fast at init	at initialization	none	via policies and ACL
Configuration sources	Env, .env, YAML…	Env, JSON, TOML	YAML, Kubernetes API	API, KV engine
Static typing	explicit primitives	Pydantic models	none	none
Secret management	via API extension	none	basic (Secrets)	encryption, rotation
Hot reload	no	no	via Kubernetes rollout	supported
Operational overhead	minimal	moderate	high (cluster management)	high (operational overhead)

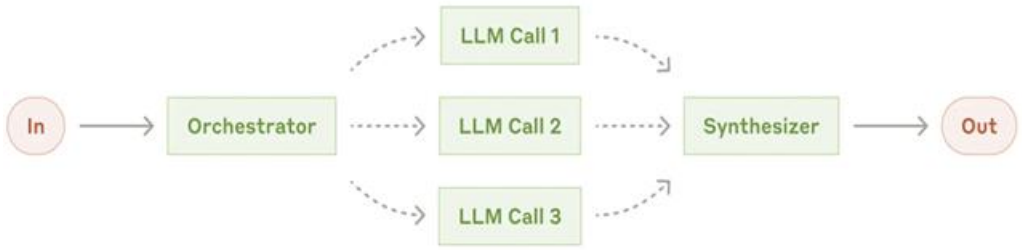

Fig. 1. Multi-stage LLM pipelines (compiled by the author, based on: [1]).

III. Impact of Configuration Management on the Reliability of Multi-Stage LLM Pipelines

Multi-stage LLM pipelines (see Fig. 1) consist of a sequence of interconnected model invocations, often with intermediate pre- and post-processing steps. Each pipeline stage—whether format conversion, annotation, code snippet generation or structured response construction—requires precise tuning of parameters (sampling, system prompts, length limits) and consideration of external factors (model versions, API timeouts, memory constraints) [1].

In the absence of a single source of truth, parameters may "drift" between stages, leading to unpredictable failures, context mismatches and an increase in hallucinations. Structured output and function-call interfaces improve formalism but do not guarantee consistency of settings across calls: an incorrect JSON-schema or mismatched function version can break the entire pipeline [3, 7]. Practitioner reports from Primer AI confirm that disparate configurations frequently caused production-replica failures [1, 8].

Below is Table 2, summarizing key aspects of strict configuration management.

TABLE 2. Key aspects of strict configuration management [1, 2].

Category	Mechanism	Description	Advantages	Implementation Aspects
1. Deterministic execution	Fail-fast validation	Validation of required parameters at service startup	• Immediate error detection• Prevention of silent failures	• Validation schemas (JSON Schema, Protobuf)• Static config analysis
2. Experiment-context isolation	Variable groups	Definition of multiple ENV-setting "groups" for parallel tests	• Parallel A/B testing without rebuilds• Rapid config switching	• Namespaces in configuration• Hierarchical structure
3. Secure secret storage	ConfigSource adapters	Plugin model for reading secrets from Vault, HSM or cloud secret managers	• No leaks to logs or VCS• Centralized access control	• Support for HashiCorp Vault, AWS Secrets Manager, Azure Key Vault
4. Minimizing cold-start latency	Zero dependencies	Complete absence of external libraries in the runtime image	• Ultra-fast initialization• Lower resource consumption	• Use of scratch or distroless images• Static linking

Together, these mechanisms reduce failure risks and enhance reliability for multi-call LLM pipelines. At Primer AI, a multi-layered pipeline for unstructured-document analysis—comprising preprocessing, retrieval-augmented generation, post-processing and function calls—suffered frequent outages and lengthy recovery times due to configuration mismatches prior to PKonfig adoption.

Transitioning to a centralized, strongly typed and lightweight solution enabled Primer AI to improve pipeline stability and accelerate on-call response, thereby reducing mean time to recovery (MTTR). Furthermore, clear environment isolation simplified parameter A/B testing without the need to recompile images.

IV. Application Of LLM Systems In Organizations

Modern programming platforms increasingly employ LLM assistants to evaluate solutions based on natural-language descriptions and test scenarios. A typical pipeline architecture comprises code ingestion; static analysis and test execution; formulation of an LLM prompt to explain errors and generate

recommendations; result aggregation; and feedback delivery [1, 2].

Configuring an LLM-assistant-based pipeline demands high adaptability: response-generation parameters (temperature, max_tokens), API-call timeouts and test-suite composition vary according to the discipline and user proficiency; the access-control module must isolate the sandbox environment, preventing any interference with global test settings and LLM prompts [11, 12]; and to avert DDoS-style overloads, strict inference timeouts and quotas—enforced by the serving infrastructure—must be managed centrally.

Integrating PKonfig into the platform's CI/CD workflow established a formal boundary between sandbox and production environments, allowing two independent sets of environment variables (sandbox_, prod_) without duplicating deployment code [8, 9]. Scaling to 1 000 concurrent users now requires a single YAML configuration edit, eliminating container rebuilds and minimizing downtime. A single source of truth maintains parameter consistency across the test runner, LLM module and feedback service.

Enhanced reliability of LLM assistants directly correlates with learning quality: timely, automated feedback boosts learner motivation and mitigates the "failure-spiral" effect typical of delayed manual grading [2]. Moreover, the versioning and change-tracking features in PKonfig furnish instructors with a detailed history of experimental configurations, enabling empirical identification of generation parameters that optimally support material mastery.

Deploying LLM assistants via a formalized CI/CD pipeline with PKonfig has demonstrated that adaptive tuning of generation parameters (temperature, response length, timeouts) and clear separation of sandbox and production environments not only ensure reliability and security but also simplify scaling without modifying the codebase. Versioned experimental environments allow teachers to empirically identify the best LLM parameters, which improves student participation by providing quick, high-quality feedback and lowering the hazards associated with grading delays.

V. Conclusion

The study confirmed the hypothesis that a minimalist, strictly typed approach to configuration management using PKonfig substantially enhances the reliability and reproducibility of multi-stage LLM pipelines. Comparative analysis showed that PKonfig outperforms pydantic-settings, ConfigMaps and Vault in cold-start speed and flexibility of configuration sources. Adopting PKonfig led to reduced cold-start latency, decreased mean time to recovery (MTTR), fewer incidents and improved SLA compliance. The presented methodology for assessing configuration-management impact on key metrics (MTTR, SLA adherence, throughput, accuracy) through comparative analysis and case studies provides a robust evaluation framework.

Future research directions include:

- Developing adapters for secret stores (AWS Secrets Manager, GCP Secret Manager, Vault)
- Exporting typed configuration schemas (OpenAPI) for self-documentation
- Creating a public latency/throughput benchmark for automated evaluation of configuration managers in LLM systems

References

[1] M. Mangaonkar and V. K. Penikalapati. "Enhancing production data pipeline monitoring and reliability through large language models (LLMs)," Eduzone: International Peer Reviewed/Refereed Multidisciplinary Journal, vol. 13, issue 1, pp. 51–56, 2024.

[2] Z. Ye, T. H. M. Le, and M. A. Babar. "LLMSecConfig: An LLM-Based Approach for Fixing Software Container Misconfigurations," pp. 1–13, 2025. DOI: 10.48550/arXiv.2502.02009

[3] S. Sultan, I. Ahmad, and T. Dimitriou. "Container security: Issues, challenges, and the road ahead," IEEE Access, vol. 7, pp. 52976–52996, 2019. DOI: 10.1109/ACCESS.2019.2911732

[4] L. Al Mashta. "Containers: Security Challenges and Mitigation Strategies: A Systematic Literature Review," pp.16-35, 2024.

[5] S. Q. Yan et al. "Corrective retrieval augmented generation," pp. 1–8, 2024.

[6] Z. Ye, T. H. M. Le, and M. A. Babar. "LLMSecConfig: An LLM-Based Approach for Fixing Software Container Misconfigurations," 2025.DOI:10.48550/arXiv.2502.02009

[7] T. H. M. Le and M. A. Babar. "Automatic Data Labeling for Software Vulnerability Prediction Models: How Far Are We?," Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 131–142, 2024. DOI:10.1145/3674805.3686675

[8] Y. Wang et al. "Reasoning with Large Language Models on Graph Tasks: The Influence of Temperature," 2024 5th International Conference on Computer Engineering and Application (ICCEA), IEEE, pp. 630–634, 2024. DOI: 10.1109/ICCEA62105.2024.10603677

[9] "7 Lessons from Building a Small-Scale AI Application," Thelis Blog. [Online]. Available: https://www.thelis.org/blog/lessons-from-ai (Accessed: May 6, 2025).

[10] "AI's Privilege Expansion," Digital Native Tech. [Online]. Available: https://www.digitalnative.tech/p/ais-privilege-expansion (Accessed: May 6, 2025).

[11] P. Calçado. "Building AI Products—Part I: Back-end Architecture," Philcalcado.com, Dec. 14, 2024. [Online]. Available: https://philcalcado.com/2024/12/14/building-ai-products-part-i.html (Accessed: May 6, 2025).

[12] A. Maloo. "The Bitter Lesson: Rethinking How We Build AI Systems," AnkitMaloo.com. [Online]. Available: https://ankitmaloo.com/bitter-lesson/ (Accessed: May 6, 2025).