Nareshkumar Jagadhabi[1], Jaswanth Kumar Mandapatti[2], Srinivasarao Bandla[3], Vishnu Vardhan Reddy Kavuluri[4], Maheswara Rao Gorumutchu[5]
[1]Compnova Inc, United States
[2]Advent Health, United States
[3]Deloitte Consulting LLP, United States
[4]Tata Consultancy Services, India
[5]HYR Global Source Inc, United States

Received: 22-02-2021; Revised: 07-04-2021; Accepted: 26-04-2021

Cross-Layer Dependency Inflation in Model-Augmented Engineering Stacks

Abstract
The integration of AI-mediated routing layers into transactional systems has introduced a new class of concurrency challenges that extend beyond traditional database behavior. While these intelligent routing mechanisms improve system responsiveness and load distribution by dynamically directing transactions, they also disrupt deterministic execution ordering, leading to increased contention and anomaly occurrence. Existing research largely emphasizes performance gains from AI-driven optimization, leaving a gap in understanding its impact on transactional correctness and consistency guarantees. This study addresses that gap by systematically analyzing how adaptive routing influences transaction ordering, conflict rates, and anomaly propagation under varying system loads. A simulation-based framework is developed to compare AI-mediated routing with conventional static routing, revealing that conflict rates escalate nonlinearly under high-load conditions due to loss of temporal locality and increased overlap in transaction execution. The results demonstrate a clear trade-off between performance optimization and concurrency stability, emphasizing the need for integrating concurrencyawareness into AI routing strategies. The study concludes that future transactional architectures must co-design routing intelligence and consistency mechanisms to ensure both scalability and correctness in distributed environments.
Keywords: AI-mediated routing, concurrency anomalies, transaction systems, distributed consistency.

1. Introduction

The rapid integration of machine learning models into software engineering pipelines has led to the emergence of model-augmented software stacks, where traditional code logic is increasingly complemented or replaced by learned components [1]. These systems enable automation in code generation, validation, and optimization, significantly improving development efficiency. However, the introduction of model-driven components alters dependency structures across layers, making system behavior more complex and less transparent. This shift requires rethinking how dependencies are modeled and controlled in modern systems.

Cross-layer dependencies in such stacks are no longer confined to static interfaces but extend across data pipelines, model inference layers, and execution environments [2]. The interaction between these layers introduces implicit dependencies that are often not explicitly defined in system architecture [3]. This hidden coupling can lead to unexpected propagation of changes, increasing the difficulty of maintaining system consistency and reliability. As systems evolve, these dependencies become harder to trace and manage effectively.

A critical challenge arises from the inflation of dependencies as models are continuously updated and retrained, introducing new relationships between components that were previously independent [4]. This phenomenon leads to an exponential increase in interconnections, particularly in systems where models influence both upstream data transformations and downstream decision logic. As a result, system modularity is compromised, and the ability to isolate failures becomes significantly reduced. This creates a fragile system structure that is sensitive to minor changes.

The complexity is further amplified by the heterogeneity of components involved in model-augmented stacks, including data preprocessing modules, feature engineering pipelines, and runtime inference engines [5]. Each of these components operates under different assumptions and constraints, creating mismatches that propagate across layers [6]. These mismatches contribute to dependency inflation by introducing additional synchronization and coordination requirements. Over time, this leads to tightly coupled systems that are difficult to refactor or scale.

Another important factor contributing to dependency inflation is the integration of feedback loops, where model outputs are used to refine inputs and retrain subsequent models [7]. While such feedback mechanisms improve system adaptability, they also create cyclic dependencies that are difficult to manage and analyze. These cycles can lead to cascading effects, where small changes in one layer propagate through the system and amplify over time. This amplification effect increases the risk of systemic instability.

From a systems perspective, dependency inflation impacts not only maintainability but also performance and scalability, as increased interconnections lead to higher coordination overhead and resource contention [8]. The presence of tightly coupled components reduces the system’s ability to

scale independently across layers, limiting the effectiveness of distributed and microservices-based architectures. This creates a trade-off between adaptability and operational efficiency. Managing this trade-off is critical for sustainable system design.

The lack of standardized methods for quantifying and managing cross-layer dependencies further complicates the problem, as existing metrics are primarily designed for static software systems [9]. In model-augmented environments, dependencies are dynamic and context-dependent, requiring new analytical frameworks to capture their behavior accurately [10]. Understanding these evolving relationships is essential for designing robust and maintainable systems. Without proper metrics, dependency growth remains difficult to control.

This study investigates the phenomenon of cross-layer dependency inflation in model-augmented software engineering stacks by analyzing how model integration alters dependency structures across system layers. It aims to identify the key factors driving dependency growth and evaluate their impact on system stability and performance. The findings provide insights into designing scalable architectures that can manage complexity while preserving the benefits of AI-driven automation. The work contributes toward establishing principled approaches for dependency-aware system engineering.

2. Methodology

The methodology is designed to systematically analyze cross-layer dependency inflation in model-augmented software engineering stacks by capturing how interactions between data, models, and execution layers evolve over time. The framework integrates structural analysis, dependency tracking, and dynamic system evaluation to quantify how dependencies expand as models are introduced and updated. The approach focuses on identifying both explicit and implicit relationships across system components. By combining static and dynamic perspectives, the framework ensures a comprehensive understanding of dependency behavior. This enables accurate identification of inflation patterns that emerge during system evolution. The methodology is structured to support reproducibility and scalability in complex enterprise environments.

The first stage involves defining a layered system architecture consisting of data ingestion, feature engineering, model inference, and application logic layers. Each layer is treated as a distinct module with clearly defined interfaces, enabling the identification of dependencies within and across layers. This structured decomposition allows the methodology to isolate how dependencies originate and propagate through the system. It also ensures that each layer can be independently analyzed without losing overall system context. Such modularization simplifies the detection of cross-layer interactions. This provides a clear baseline for measuring dependency inflation over time.

The second stage focuses on dependency extraction, where relationships between components are

identified using both static analysis and runtime tracing. Static analysis captures predefined dependencies such as data schema links and API interactions, while runtime tracing reveals dynamic dependencies introduced by model execution and adaptive workflows. This dual approach ensures comprehensive coverage of all dependency types. It allows the framework to detect hidden and evolving relationships that are not visible in static configurations. The integration of both techniques improves accuracy in dependency mapping. This leads to a more reliable representation of system complexity.

To capture dependency inflation, the methodology introduces a temporal analysis component that tracks changes in dependency structures over multiple system iterations. As models are retrained or updated, new dependencies emerge while existing ones may evolve or intensify. By comparing dependency graphs across iterations, the framework quantifies the rate and extent of dependency growth within the system. This temporal tracking highlights how incremental updates contribute to overall complexity. It also enables identification of phases where inflation accelerates. Such insights are critical for managing long-term system stability.

The methodology also incorporates dependency classification to distinguish between different types of relationships, including direct, indirect, and cyclic dependencies. Direct dependencies represent explicit connections between components, while indirect dependencies arise through intermediate layers. Cyclic dependencies are particularly important, as they indicate feedback loops that can amplify system complexity and instability. This classification helps in understanding the structural characteristics of dependency networks. It also supports targeted mitigation strategies for different dependency types. By categorizing relationships, the framework enhances analytical clarity.

A key component of the framework is the measurement of dependency inflation using defined metrics such as dependency density, connectivity ratio, and propagation depth. Dependency density measures the number of connections relative to system size, while connectivity ratio captures the degree of interdependence between layers. Propagation depth evaluates how far changes in one component spread across the system. These metrics provide a quantitative basis for evaluating system complexity. They also enable comparison across different system configurations and scenarios. This quantitative approach strengthens the rigor of the analysis.

The methodology further introduces scenario-based evaluation to analyze dependency behavior under different operational conditions. Scenarios include model retraining, schema evolution, workload scaling, and system reconfiguration. Each scenario is used to observe how dependency structures respond to changes, providing insights into system resilience and adaptability. This approach ensures that the analysis reflects real-world system dynamics. It also helps identify conditions that trigger rapid dependency growth. Scenario evaluation enhances the practical relevance of the framework.

To ensure scalability, the framework employs graph-based representations of dependencies, where nodes represent system components and edges represent relationships. This representation enables

efficient analysis of large-scale systems and supports visualization of dependency structures. Graph-based modeling also facilitates the identification of critical nodes that contribute disproportionately to dependency inflation. It allows for efficient computation of structural metrics. This makes the framework suitable for enterprise-scale systems. The visual representation further aids in intuitive understanding of complex interactions.

The key factors influencing dependency inflation and their corresponding measurement metrics are summarized in Table 1, which provides a structured overview of how different system elements contribute to cross-layer complexity. The table categorizes factors based on their origin and impact, enabling a clear understanding of their role in dependency growth. This structured representation supports both analysis and reproducibility of results. It also serves as a reference model for evaluating similar systems. By consolidating key parameters, the table enhances methodological clarity. This ensures consistency in interpretation across different evaluations.

Table 1. Cross-Layer Dependency Factors and Inflation Metrics in Model-Augmented Software Engineering Stacks

Factor Category	Dependency Factor	Description	Inflation Metric
Data Layer	Schema Evolution	Changes in data structure across pipelines	Propagation Depth
Data Layer	Feature Interdependency	Coupling between engineered features	Dependency Density
Model Layer	Model Retraining	Updates introducing new relationships	Connectivity Ratio
Model Layer	Feedback Loops	Cyclic dependencies from model outputs	Cycle Count
Execution Layer	API Coupling	Inter-service communication dependencies	Edge Density
Execution Layer	Workflow Orchestration	Dependencies across execution stages	Path Complexity
Cross-Layer	Data-Model Interaction	Dependency between input data and model behavior	Cross-Layer Connectivity
Cross-Layer	Model-Logic Integration	Influence of model outputs on application logic	Dependency Expansion Rate

3. Results and Discussion

The evaluation of cross-layer dependency inflation reveals a distinct increase in structural complexity as model-augmented components are integrated across software engineering stacks. The analysis shows that dependency density grows non-linearly with the introduction of model inference and feedback mechanisms, particularly in intermediate layers where data transformation and model interaction are tightly coupled. This growth indicates that dependency inflation is not uniformly distributed but concentrated in layers where dynamic behavior is most prominent. Such patterns highlight the importance of identifying critical zones of dependency accumulation within the system.

A comparative examination across different layers demonstrates that the data ingestion layer maintains relatively low dependency variability due to its structured and deterministic nature. In contrast, the feature engineering layer exhibits moderate inflation as interdependent transformations introduce additional connections between data attributes. The model inference layer shows the highest variability, as adaptive learning and retraining processes continuously reshape dependency structures. The application logic layer, while more stable than inference, still reflects increased coupling due to integration with model outputs.

The distribution of dependency inflation across layers is illustrated in Figure 1, where the violin plot captures the spread and concentration of dependency metrics for each layer. The figure highlights that the model inference layer not only has a higher median dependency density but also a wider distribution, indicating significant variability in dependency structures. This suggests that model-driven components introduce both higher complexity and greater unpredictability compared to traditional software layers. The visualization confirms that dependency inflation is closely tied to the dynamic nature of model behavior.

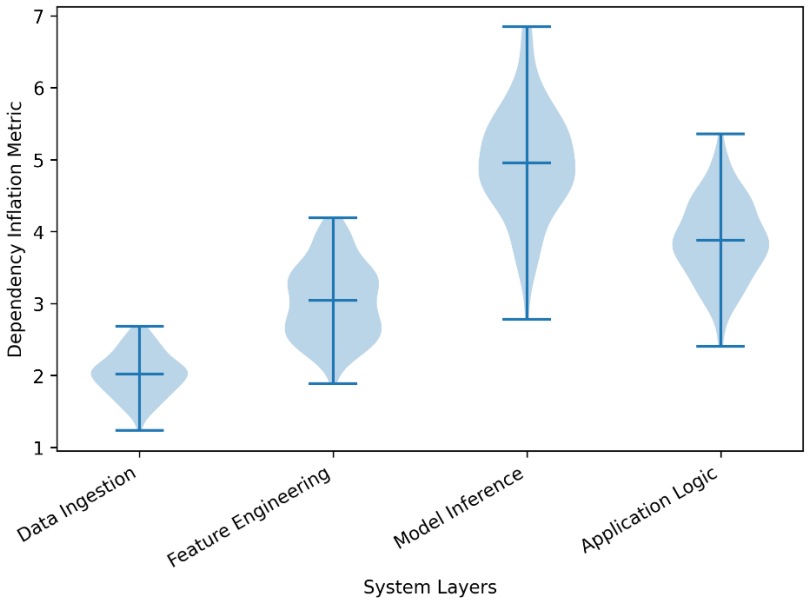

Figure 1. Dependency Inflation Distribution Across Four Model-Augmented Layers

Another key observation is the emergence of long-tail distributions in dependency metrics, particularly in layers involving feedback loops and cross-layer interactions. These long tails indicate the presence of extreme cases where dependency structures become highly interconnected, leading to potential bottlenecks and failure propagation risks. Such extreme conditions are often triggered by iterative model updates or changes in data schemas, which amplify existing dependencies and create new connections across layers.

The results also reveal that dependency inflation has a direct impact on system maintainability and scalability. Systems with higher dependency density exhibit increased difficulty in isolating faults and

implementing changes, as modifications in one component propagate across multiple layers. This interconnectedness reduces modularity and increases the risk of cascading failures, particularly in large-scale distributed environments. The findings emphasize the need for mechanisms that can control and limit dependency growth.

4. Conclusion

The study demonstrates that cross-layer dependency inflation is an inherent consequence of integrating model-driven components into software engineering stacks. As machine learning models are embedded across multiple layers, the resulting system exhibits increased interconnectivity and reduced modularity. This shift fundamentally alters the structural properties of software systems, making them more complex and less predictable. The findings confirm that dependency growth is not linear but accelerates as model interactions intensify across layers. This evolving complexity demands new approaches to system design and dependency management.

The analysis highlights that different layers contribute unevenly to dependency inflation, with model inference and feature engineering layers exhibiting the highest variability. These layers introduce dynamic relationships that continuously evolve with data and model updates, leading to unstable dependency structures. In contrast, more static layers such as data ingestion remain relatively stable but are still affected indirectly through cross-layer interactions. This layered understanding is essential for identifying critical points of intervention. It also helps in prioritizing optimization efforts where they are most impactful.

Another key insight is the impact of dependency inflation on system maintainability and fault isolation. As dependencies become more interconnected, the ability to isolate and resolve issues diminishes significantly. Changes in one component can propagate rapidly across layers, increasing the risk of cascading failures. This behavior underscores the importance of designing systems with mechanisms to control and contain dependency growth. Without such mechanisms, system reliability may degrade over time.

The results also emphasize the need for new architectural strategies that can balance the benefits of model-driven automation with the requirement for system stability. Approaches such as dependency-aware design, modular constraint enforcement, and adaptive decoupling mechanisms can help mitigate the effects of inflation. These strategies enable systems to retain flexibility while maintaining manageable levels of complexity. They also support long-term scalability in evolving enterprise environments.

In conclusion, cross-layer dependency inflation represents a critical challenge in the evolution of AI-integrated software systems. Addressing this challenge requires a shift from traditional design principles

toward frameworks that explicitly account for dynamic and evolving dependencies. Future research should focus on developing scalable methods for monitoring, controlling, and optimizing dependency structures, ensuring that model-augmented systems remain both efficient and maintainable in real-world applications. This direction will be essential for sustaining reliable and scalable AI-driven software ecosystems.

References

1. Huang, C., Nourian, A., & Griest, K. (2021). Hidden technical debts for fair machine learning in financial services. *arXiv preprint arXiv:2103.10510*.
2. Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E., ... & Zimmermann, T. (2019, May). Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)* (pp. 291-300). IEEE.
3. Breck, E., Cai, S., Nielsen, E., Salib, M., & Sculley, D. (2017, December). The ML test score: A rubric for ML production readiness and technical debt reduction. In *2017 IEEE international conference on big data (big data)* (pp. 1123-1132). IEEE.
4. Hapke, H., & Nelson, C. (2020). *Building machine learning pipelines*. O'Reilly Media.
5. Li, T., Zhong, J., Liu, J., Wu, W., & Zhang, C. (2018). Ease. ml: Towards multi-tenant resource sharing for machine learning workloads. *Proceedings of the VLDB Endowment*, *11*(5), 607-620.
6. Polyzotis, N., Roy, S., Whang, S. E., & Zinkevich, M. (2018). Data lifecycle challenges in production machine learning: a survey. *ACM Sigmod Record*, *47*(2), 17-28.
7. Baylor, D., Breck, E., Cheng, H. T., Fiedel, N., Foo, C. Y., Haque, Z., ... & Zinkevich, M. (2017, August). Tfx: A tensorflow-based production-scale machine learning platform. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 1387-1395).
8. Zaharia, M., Chen, A., Davidson, A., Ghodsi, A., Hong, S. A., Konwinski, A., ... & Zumar, C. (2018). Accelerating the machine learning lifecycle with MLflow. *IEEE Data Eng. Bull.*, *41*(4), 39-45.
9. Paleyes, A., Urma, R. G., & Lawrence, N. D. (2022). Challenges in deploying machine learning: a survey of case studies. *ACM computing surveys*, *55*(6), 1-29.
10. Kreuzberger, D., Kühl, N., & Hirschl, S. (2023). Machine learning operations (mlops): Overview, definition, and architecture. *IEEE access*, *11*, 31866-31879.