AI and Machine Learning Advances
Volume 2, Issue 2
ISSN: 3067-3216

A Responsibility-Allocation Framework for LLM-First and Hybrid Code-First Enterprise AI Architectures

[1]*Swapneswar Sundar Ray
[1]Independent Researcher, New Jersey, USA.

Abstract

Enterprise use of large language models creates architectural challenges involving control, traceability, state management, output conformance, and operational governance. This design-science paper introduces the Deterministic–Probabilistic Responsibility Allocation Framework, which distinguishes a model-first architecture from a hybrid code-first architecture by specifying where validation, routing, state management, contextual reasoning, output enforcement, recovery, and telemetry should reside. The framework contributes an explicit responsibility-assignment method, a boundary contract for each model invocation, and lifecycle control points that connect probabilistic generation to deterministic validation and escalation. It is examined through an application programming interface sandbox-generation illustration and a model-agnostic reference procedure covering prompt structure, OpenAPI parsing, schema validation, retry and fallback behavior, and output acceptance criteria. The analysis shows how the framework can structure fault isolation, observable execution paths, and enforceable output contracts without claiming measured improvements in reliability, scalability, latency, or cost. Controlled multi-model benchmarking remains necessary to quantify these properties.

Keywords: AI governance; API sandbox generation; deterministic validation; enterprise AI architecture; large language models; responsibility allocation; structured output.

1 Introduction

Large language models (LLMs) have changed the design of intelligent software systems by enabling natural-language understanding, contextual generation, and reasoning. Their use has accelerated innovation in conversational artificial intelligence (AI), knowledge retrieval, software engineering, and enterprise automation. At the same time, production deployments expose architectural problems involving nondeterministic behavior, hallucinations, limited observability, and difficulty controlling multi-step execution.

Early implementations often adopt an LLM-first architecture in which a model is responsible for validation, transformation, reasoning, and output generation within a single prompt-driven interaction. This approach can accelerate prototyping, but performance may degrade as task complexity and prompt ambiguity increase. In this study, LLM cognitive load refers to the number and diversity of validation, transformation, reasoning, routing, and formatting responsibilities assigned to a single model invocation. The concept is used as an architectural construct rather than as a directly measured psychological or computational variable.

Traditional software engineering instead emphasizes deterministic control, modular design, explicit interfaces, and separation of concerns. A code-first approach can use these principles to assign validation, routing, state management, and output enforcement to deterministic components while reserving LLMs for tasks that require probabilistic reasoning or contextual interpretation.

The scholarly and practical gap addressed in this study is the absence of a concise, application-level method for deciding which responsibilities should remain deterministic, which may be delegated to an LLM, and which require escalation when machine-verifiable controls are insufficient. The study therefore compares LLM-first and hybrid code-first paradigms, develops DPRAF, and examines its use through representative enterprise workflow categories and an API sandbox-generation illustration. The objectives are to characterize LLM cognitive load, define responsibility-assignment criteria and model-call boundary contracts, compare responsibility allocation between the two architectural approaches, and examine implications for fault isolation, observability, maintainability, governance,

Swapneswar Sundar Ray
Independent Researcher, New Jersey, USA.
Email: swapneswar.ray@gmail.com

Received: 4-Aug-2026 Revised: 24-Aug-2026 Accepted: 29-Aug-2026

AI and Machine Learning Advances
Volume 2, Issue 2
ISSN: 3067-3216

and API sandbox generation.

The study addresses the following research questions:

RQ1: How do LLM-first and hybrid code-first architectures differ in their allocation of validation, routing, state management, reasoning, and output-control responsibilities?

RQ2: How can deterministic validation, routing, and post-processing improve fault isolation, observability, and output control in multi-step enterprise AI workflows?

RQ3: What architectural implications, limitations, and production considerations arise when DPRAF is applied to enterprise API sandbox generation?

The principal contribution is not a new model, guardrail product, or orchestration runtime. DPRAF integrates five elements that are usually treated separately: (a) explicit responsibility-assignment criteria, (b) a boundary contract defining the permitted context, tools, output schema, and acceptance checks for every model call, (c) validator-driven retry and escalation paths, (d) stage-level evidence through state transitions, traces, and audit events, and (e) governance insertion points linked to the workflow lifecycle. The novelty therefore lies in the integrated allocation method and its application-level decision logic rather than in any individual software-engineering control.

2 Literature Review

The transformer architecture established the technical foundation for contemporary LLMs (Vaswani et al., 2017); subsequent work demonstrated few-shot behavior (Brown et al., 2020) and retrieval-augmented generation for grounded knowledge-intensive tasks (Lewis et al., 2020). A systematic review has documented the growing role and unresolved reliability concerns of LLMs in software engineering (Hou et al., 2024).

Peer-reviewed human–AI and orchestration research demonstrates the value of decomposing complex model interactions. AI Chains showed that exposing intermediate steps can improve transparency and controllability (Wu et al., 2022). AutoGen formalized multi-agent conversation patterns involving models, tools, code execution, and human input (Wu et al., 2024), while RouteLLM studied learned model routing to balance quality and inference cost (Ong et al., 2025). These approaches motivate decomposition and orchestration but do not provide an application-level rule for allocating validation, state, security-sensitive decisions, and output enforcement between code and models.

Guardrail and structured-output research addresses specific control points. Grammar-constrained decoding enforces formal output structures (Geng et al., 2023).

Table 1: Relationship between prior approaches and the distinct scope of DPRAF

Approach	Primary contribution	Control mechanism	Distinction from DPRAF
AI Chains (Wu et al., 2022)	Decomposed and inspectable LLM interactions	Chained prompts and editable intermediate outputs	Does not classify state, validation, security, and escalation responsibilities across an enterprise workflow
Systematic guardrails (Dong et al., 2024)	Context-sensitive guardrail design	Input/output safeguards and verification	Focuses on safeguarding requirements rather than end-to-end responsibility allocation
Structured generation (Geng et al., 2023)	Formally constrained model output	Grammar-constrained decoding	Controls generation syntax but not routing, state, recovery, audit, or governance placement
PROMPTEVALS (Viret al., 2025)	Assertions for production LLM pipelines	Developer-defined output criteria and retry	Concentrates on assertion discovery; DPRAF specifies the wider model-call boundary contract
AutoGen and RouteLLM (Wu et al., 2024; Ong et al., 2025)	Orchestration and model selection	Agent conversations, tools, and learned routing	Provide runtimes or routing methods rather than responsibility-assignment criteria
AIOpsLab (Chen et al., 2025)	Reproducible operational-agent evaluation	Workloads, fault injection, environments, and telemetry	Provides evaluation infrastructure; DPRAF defines an application architecture to be evaluated

Dong et al. (2024) argued that LLM guardrails require systematic, context-sensitive design rather than isolated filters. PROMPTEVALS provides peer-reviewed evidence that production LLM pipelines benefit from developer-defined assertions evaluated alongside generation (Vir et al., 2025). These studies establish the need for constraints and validation; DPRAF extends this concern across the full workflow boundary, including routing, state mutation, retry, escalation, and audit evidence.

Agent evaluation research further emphasizes explicit interfaces and realistic execution environments. AgentBench evaluates agents across interactive environments (Liu et al., 2024); ToolEmu uses an emulated sandbox to identify tool-use risks (Ruan et al., 2024); SWE-agent demonstrates the importance of structured agent–computer interfaces and execution feedback (Yang et al., 2024); and AIOpsLab combines workloads, fault injection, cloud orchestration, and telemetry for reproducible operational evaluation (Chen et al., 2025). Enterprise and software-development benchmarks similarly highlight access-aware tasks, alignment checks, and review mechanisms (Vishwakarma et al., 2025; Liu et al., 2025).

Governance requires lifecycle-level control rather than output filtering alone. The NIST Generative AI Profile organizes risk-management activities around governance, mapping,
measurement, and management across the AI lifecycle (Autio et al., 2024).

Table 1 distinguishes representative prior approaches from the integrated contribution of DPRAF.

The literature therefore supports individual mechanisms used by DPRAF but does not combine them into a single responsibility-assignment method. DPRAF's distinct scope is the application-level integration of assignment criteria, invocation contracts, deterministic acceptance checks, escalation paths, stage-level evidence, and governance control points.

3 Materials and Methods

3.1 Study Design

This research used a design-science approach to develop and examine an architectural framework for allocating responsibilities between deterministic software components and probabilistic LLM components. The framework was derived from separation-of-concerns, modularity, explicit-interface, fault-containment, and lifecycle-governance principles. The design artifact was examined through comparison with an LLM-first reference architecture, requirements traceability, and a synthetic API sandbox-generation walkthrough. The study is therefore a conceptual architecture paper rather than a prospectively designed controlled experiment.

The workflow categories considered in the architectural analysis included structured input processing, contextual reasoning, multi-step task decomposition, structured data transformation, and output generation. These categories were used to examine where deterministic controls and bounded model reasoning should be placed within an enterprise workflow.

Table 2: Conceptual responsibility allocation in LLM-first and hybrid architectures

Architectural element	LLM-first architecture	Hybrid architecture
Input validation	Assigned to the prompt/model invocation	Deterministic schema and rule validation
Routing	Assigned to the model invocation	Code-controlled routing
Task decomposition	Assigned within the monolithic interaction	Deterministic decomposition into bounded subtasks
State management	Prompt context	External state and explicit workflow state
Output validation	Prompt-level output instructions	Schema validation and deterministic normalization
Retry handling	Re-prompting within the model interaction	Policy-controlled retry, timeout, and fallback paths
Logging	Final-response-centered logging	Stage-level tracing and audit events

AI and Machine Learning Advances
Volume 2, Issue 2
ISSN: 3067-3216

3.2 LLM-First Reference Architecture

The LLM-first architecture assigns validation, transformation, reasoning, routing, and output generation primarily to a model-driven interaction. The hybrid code-first architecture instead places deterministic boundaries around model calls. Table 2 presents the conceptual responsibility allocation used in this comparison.

3.3 Deterministic–Probabilistic Responsibility Allocation Framework

DPRAF separates deterministic control flow from probabilistic inference. As shown in Figure 1, the system is organized as a multi-stage pipeline comprising input acquisition, validation, routing and task decomposition, bounded LLM reasoning, post-processing, and output generation.

DPRAF assigns a workflow responsibility by examining whether its outcome is rule-defined, state-sensitive, security-relevant, context-dependent, and machine-verifiable. A responsibility remains deterministic when it changes authoritative state, enforces a policy or schema, selects an execution path, or can be expressed as an executable acceptance test. A responsibility may be delegated to an LLM when it requires contextual interpretation or generation and its output can be bounded by an explicit contract. Tasks whose consequences are high and whose outputs cannot be reliably verified are routed to a fallback or human approval path rather than being assigned autonomous model authority.

Each LLM invocation is governed by a boundary contract containing six fields: task objective, permitted context, permitted tools, output schema, acceptance checks, and failure policy. This contract is the main unit that connects DPRAF's architectural allocation to implementation, testing, and governance. Table 3 summarizes the assignment logic.

Table 3: DPRAF responsibility-assignment criteria

Assignment	Decision conditions	Representative responsibilities
Deterministic component	Requires semantic interpretation or generation and produces an output that can be validated	Schema validation, routing, authorization, state mutation, retries, timeout enforcement, output normalization, audit events
Bounded LLM component	Requires semantic interpretation or generation and produces an output that can be validated	Intent interpretation, explanatory text, contextual mock values, summarization, classification under an explicit schema
Fallback or human approval	High-consequence decision, unresolved ambiguity, or no adequate executable acceptance test	Policy exceptions, sensitive actions, repeated validation failure, uncertain or conflicting specifications

The architecture follows four design principles. First, deterministic logic handles validation, routing, state management, and schema enforcement, while LLMs are used for contextual interpretation and generation. Second, each LLM invocation is constrained to a narrowly defined task and limited context. Third, independent processing stages support testing, debugging, and extension. Fourth, intermediate outputs are validated and monitored to improve observability and fault isolation.

The input layer receives a request R. The validation layer checks schemas, required fields, constraints, and business rules. The routing layer classifies the request, selects a workflow, and decomposes it into subtasks $\{T_1, T_2, \ldots, T_n\}$. For each reasoning task, the LLM receives a task-specific context C_i and generates an intermediate output L_i. Deterministic post-processing validates, normalizes, and aggregates intermediate outputs into the final response O.

Figure 2 presents a production-oriented reference architecture. Solid components represent the core DPRAF workflow, while dashed components represent optional production controls whose implementation depends on risk, policy, and deployment context.

3.4 Reproducible Reference Procedure

DPRAF is model- and implementation-language agnostic. Consequently, this conceptual study does not attribute results to a particular provider, model version, temperature, token limit, runtime, or programming language. Instead, it defines the configuration and evidence that any empirical implementation must record. Table 4 states the disclosed reference profile used for the API illustration.

AI and Machine Learning Advances
Volume 2, Issue 2
ISSN: 3067-3216

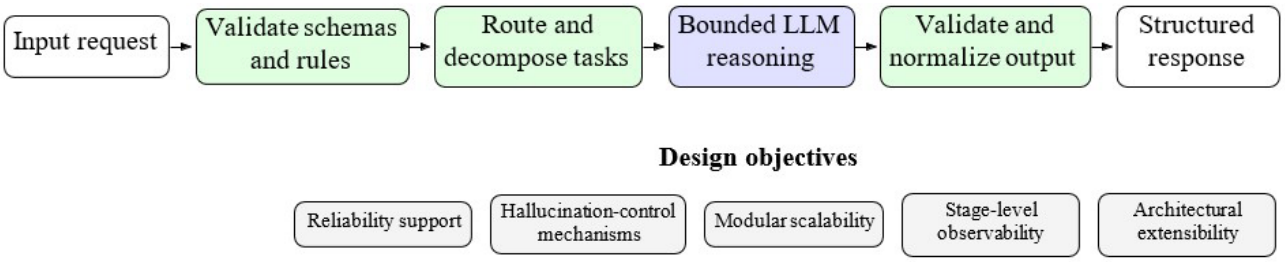

Figure 1: Deterministic–Probabilistic Responsibility Allocation Framework.

Note. Green stages represent deterministic operations, and the blue stage represents bounded probabilistic reasoning. Design objectives are architectural aims rather than independently measured outcomes.

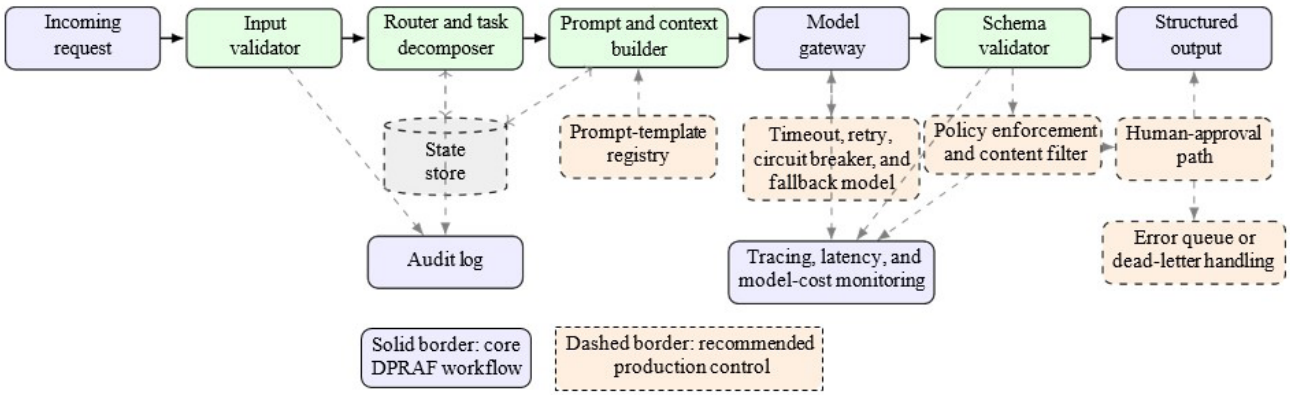

Figure 2: Reference architecture for production implementation.

Note. Solid components depict the core DPRAF workflow. Dashed components represent configurable production controls selected according to application risk and governance requirements.

Table 4: Model-agnostic reference profile for reproducible DPRAF implementations

Element	Reference specification
Model and provider	Not fixed by DPRAF; empirical studies must record the provider, model identifier, release or snapshot date, and endpoint version
Generation settings	External configuration containing temperature, top-□, input and output token limits, timeout, and seed where supported
Prompt-template structure	System role; bounded task objective; permitted endpoint context; prohibited behavior; required output schema; acceptance criteria; output-only instruction
Input specification	OpenAPI 3.x document processed by a version-compatible parser; unsupported versions fail validation before model invocation
Validation	OpenAPI structural validation, JSON syntax and JSON Schema validation, required-property checks, allowed-field checks, status-code and media-type checks, and application business rules
Retry and fallback	Policy-configured bounded retry using validator feedback; exhaustion routes to deterministic fallback, error queue, or human approval
Implementation interface	Language-neutral modules for parsing, routing, prompt construction, model access, validation, state, telemetry, and output publication
Output acceptance	Accepted only when parsing succeeds, the declared schema is satisfied, required fields are present, prohibited fields are absent, and no undefined operation is introduced

AI and Machine Learning Advances
Volume 2, Issue 2
ISSN: 3067-3216

Because this paper evaluates a conceptual architecture rather than a fixed implementation, provider-specific configurations are intentionally not instantiated; future empirical evaluations should populate every field in Table 4 before execution.

The reference procedure is as follows:

1. Parse and validate the incoming specification or request; terminate before model access if structural or policy checks fail.
2. Decompose the request into responsibilities and classify each responsibility as deterministic, bounded-LLM, or escalation according to Table 3.
3. Create a boundary contract for every bounded-LLM responsibility, including the minimum context, output schema, acceptance checks, and failure policy.
4. Invoke the configured model only for responsibilities assigned to bounded contextual interpretation or generation.
5. Parse and validate each generated output. If validation fails, provide only machine-generated validator feedback to the configured bounded retry policy.
6. On retry exhaustion or unresolved ambiguity, invoke the fallback, error-queue, or human-approval path; do not publish the invalid output.
7. Aggregate accepted outputs, apply deterministic normalization, update state, and publish the structured response.
8. Record the configuration identifier, prompt-template version, validation outcome, retry count, state transition, latency, and final disposition as stage-level evidence.

For a synthetic example, consider an OpenAPI operation GET /accounts/{accountId}. Deterministic components extract the path parameter, response status codes, media type, required properties, and field constraints. A bounded LLM may generate contextual mock values only for fields declared by the response schema. The validator rejects malformed JSON, missing required properties, additional prohibited fields, undefined status codes, or operations absent from the specification. This example discloses the workflow logic without using proprietary API definitions or customer data.

3.5 Evaluation Basis and Scope

The framework is evaluated through requirements traceability, architectural comparison, and a step-by-step walkthrough of the synthetic API example rather than through a controlled performance experiment. The analysis examines whether each responsibility has an explicit owner, machine-verifiable boundary, failure path, and evidence record. Quantitative claims about reliability, scalability, latency, cost, or setup effort are outside the scope of this paper.

3.6 Evaluation Dimensions

The framework defines six dimensions for future empirical evaluation:

- Accuracy: The proportion of outputs satisfying predefined task-correctness criteria.
- Consistency: The proportion of repeated executions producing semantically equivalent and schema-compliant outputs for the same input.
- Hallucination rate: The proportion of outputs containing unsupported claims, nonexistent fields, invalid operations, or information not grounded in the supplied context.
- Latency: Elapsed time from receipt of a complete request to production of a validated final response.
- Debuggability: The ability to localize failures through observable checkpoints, stage-level traces, and explicit error boundaries.
- Scalability: The ability to accommodate growth in workflow complexity, request concurrency, throughput, and independently deployable components.

These dimensions provide a preregisterable structure for subsequent controlled evaluation. An empirical study should report test cases, repetitions, model configuration, validator definitions, uncertainty, and statistical procedures before making comparative performance claims.

3.7 API Sandbox Application Illustration

The application illustration concerns generation of sandbox-ready API responses from OpenAPI specifications for integration domains such as payments, identity management, and data exchange. It is presented to demonstrate the allocation method, not as a measured case-study evaluation. No claim is made regarding the number of specifications, endpoints, runs, setup-time reduction, invalid-output rate, latency, or cost.

Under DPRAF, parsing, OpenAPI validation, operation discovery, routing, state, schema enforcement, retry policy, and publication remain deterministic. A bounded LLM may be used only for contextual mock-value generation when the values cannot be obtained through deterministic templates. Generated values are accepted only through the validation procedure in Table 4. This allocation prevents

the model from creating endpoints, fields, status codes, or state transitions outside the supplied specification.

3.8 Analytical Approach

RQ1 is addressed through the responsibility comparison in Table 2 and the allocation criteria in Table 3. RQ2 is addressed by tracing validation, routing, output control, and escalation through Figures 1 and 2. RQ3 is addressed through the synthetic API walkthrough and the production considerations in Table 4. No inferential statistical claims are made.

4 Results: Architectural Analysis and Case Illustration

4.1 RQ1-Responsibility Allocation Findings

The architectural comparison answers RQ1 by identifying a difference in authority rather than merely a difference in prompt size. In an LLM-first architecture, the model may interpret inputs, choose actions, transform state, and format outputs within one interaction. In DPRAF, rule-defined and state-sensitive authority remains in deterministic components; the model receives only a bounded interpretation or generation responsibility.

Finding 1: DPRAF moves authoritative workflow decisions outside the model boundary. Tables 2 and 3 show that validation, routing, state transitions, acceptance, retry exhaustion, and publication are assigned to deterministic components or explicit escalation paths. The model cannot independently change the workflow graph, introduce undeclared operations, or publish an unvalidated result.

4.2 RQ2-Fault Isolation, Observability, and Output Control

The architectural traces in Figures 1 and 2 answer RQ2 by locating validation, routing, model invocation, output acceptance, recovery, and escalation at explicit stage boundaries.

Finding 2: Boundary contracts create machine-testable acceptance and rejection points for model outputs. Each bounded invocation declares its objective, permitted context, output schema, acceptance checks, and failure policy. A generated output can therefore be parsed and tested before it affects state or reaches a downstream system. Outputs that violate executable constraints are rejected or routed through bounded recovery rather than being implicitly accepted by the next prompt.

Finding 3: Stage-specific state transitions and audit events permit failure localization. Validation outcomes, routing decisions, model-call identifiers, retry counts, state transitions, and final dispositions can be associated with distinct workflow stages. This evidence structure allows an operator to distinguish an input-validation failure from a routing error, generation failure, schema-rejection event, or post-processing failure. It supports fault isolation and observability without implying that the architecture has been quantitatively benchmarked.

4.3 RQ3-API Sandbox Application and Production Implications

The synthetic API walkthrough answers RQ3 by applying the responsibility-allocation method to GET /accounts/{accountId}. Deterministic components discover the operation, interpret the declared response schema, enforce required fields and allowed status codes, and control publication. A bounded LLM may generate contextual mock values only for fields already declared by the schema. Table 5 provides a compact example of the corresponding boundary contract.

The walkthrough demonstrates how endpoint discovery, schema interpretation, acceptance, and publication can be separated from contextual mock-value generation. It identifies production implications for validation, retry limits, audit evidence, fallback, and human approval, but it does not establish comparative improvement in setup effort, validity, consistency, latency, scalability, or cost.

5 Discussion

DPRAF complements rather than replaces prior work. AI Chains and multi-agent frameworks demonstrate decomposition and orchestration (Wu et al., 2022, 2024); structured decoding and production assertions demonstrate enforceable output constraints (Geng et al., 2023; Vir et al., 2025); and agent benchmarks demonstrate the need for realistic environments, interfaces, and telemetry (Liu et al., 2024; Ruan et al., 2024; Yang et al., 2024; Chen et al., 2025). DPRAF's contribution is to connect these mechanisms through an explicit allocation decision and a common boundary contract.

This integration also clarifies the limits of architectural controls. Deterministic validators can reject outputs that violate executable rules, but they cannot guarantee factual correctness when no authoritative source or acceptance test exists. Routing and stage-level tracing can expose execution paths, but they do not by themselves establish reliability or scalability. These remain system properties that must be measured under defined workloads.

AI and Machine Learning Advances
Volume 2, Issue 2
ISSN: 3067-3216

Table 5: Worked boundary contract for the synthetic API operation

Boundary-contract field	Example specification
Objective	Generate contextual mock response values
Permitted context	Declared response schema for GET /accounts/{accountId}
Permitted tools	None
Output schema	JSON conforming to the declared OpenAPI response schema
Acceptance checks	Valid JSON; required fields; declared data types; allowed fields; permitted status code and media type
Failure policy	One validator-guided retry, followed by deterministic fallback, error-queue handling, or human approval

5.1 Practical Implications

For enterprise implementation, the boundary contract creates identifiable locations for access control, prompt-injection defenses, sensitive-data filtering, secrets management, approval, retention, and audit. These control points align with the lifecycle orientation of the NIST Generative AI Profile (Autio et al., 2024). Their presence supports governance implementation, but effectiveness depends on organization-specific policies, testing, monitoring, and independent assurance.

5.2 Threats to Validity

Construct validity depends on whether DPRAF's three assignment categories and six boundary-contract fields adequately represent enterprise workflow responsibilities. Internal validity is limited because the paper evaluates a design artifact through comparison and walkthrough rather than a controlled experiment. External validity is constrained by the API-focused illustration and by the absence of implementations across multiple domains, models, and organizations. The synthetic example improves transparency but does not substitute for released code, test specifications, repeated runs, or independent replication.

6 Limitations and Future Work

DPRAF introduces additional interfaces, validators, state transitions, and operational controls that may increase implementation complexity and latency. The framework is intentionally model-and language-agnostic, which supports portability but leaves deployment-specific design choices unresolved. The paper does not provide empirical evidence for setup effort, output validity, reliability, scalability, latency, or cost.

Future research should implement the disclosed reference procedure, release non-confidential prompts and test specifications, and compare LLM-first and DPRAF conditions using identical models and inputs. Studies should report sample sizes, repeated runs, configuration snapshots, validator definitions, failure categories, latency and cost distributions, uncertainty, and statistical procedures. Cross-domain evaluation should also examine security, governance, human approval, and maintenance overhead.

7 Conclusion

This paper introduced DPRAF as an application-level method for assigning rule-defined, state-sensitive, and machine-verifiable responsibilities to deterministic components while bounding LLM use to contextual interpretation and generation. Its distinct contribution is the integration of assignment criteria, invocation boundary contracts, validation-driven recovery, stage-level evidence, and governance control points. The API sandbox illustration and reference procedure demonstrate how the method can be implemented without exposing proprietary data. The framework is designed to support controlled and observable enterprise LLM workflows; its effects on reliability, scalability, latency, cost, and maintenance remain questions for reproducible empirical evaluation.

Declarations

Ethics Approval and Consent to Participate

Not applicable. This study did not involve human participants, personal data, animals, or clinical research.

Consent for Publication

Not applicable. This manuscript does not contain identifiable personal information requiring consent for publication.

Availability of Data and Materials

No empirical dataset was generated or analyzed for this conceptual study. The non-confidential allocation rules, boundary-contract fields, reference procedure, acceptance criteria, and synthetic API walkthrough required to examine the framework are included in the manuscript. Proprietary implementation artifacts are not publicly available because they remain subject to organizational, contractual, security, and legal restrictions.

Conflicts of Interest

The author declares that there are no conflicts of interest related to this study.

Funding

This research received no external funding.

Authors' Contributions

Swapneswar Sundar Ray conceived and designed the study, developed DPRAF, conducted the comparative architectural analysis, developed the synthetic API walkthrough and reference procedure, interpreted the findings, and prepared and revised the manuscript.

Acknowledgements

The author acknowledges the professional experiences and enterprise engineering practices that informed this research. No confidential, proprietary, customer-identifying, or security-sensitive information is disclosed in this manuscript.

Author Independence Statement

The views expressed in this article are solely those of the author and do not represent the views of any current or former employer. No confidential, proprietary, customer-identifying, or security-sensitive information is disclosed.

References

Autio, C., Schwartz, R., Dunietz, J., Jain, S., Stanley, M., Tabassi, E., Hall, P., & Roberts, K. (2024). Artificial intelligence risk management framework: Generative artificial intelligence profile (NIST AI 600-1). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.AI.600-1

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., . . . Amodei, D. (2020). Language models are few-shot learners. Advances in Neural Information Processing Systems, 33, 1877–1901.

Chen, Y., Shetty, M., Somashekar, G., Ma, M., Simmhan, Y., Mace, J., Bansal, C., Wang, R., & Rajmohan, S. (2025). AIOpsLab: A holistic framework to evaluate AI agents for enabling autonomous clouds. Proceedings of Machine Learning and Systems, 7. https://proceedings.mlsys.org/paper_files/paper/2025/hash/d1f9e4a9f109b6e8b75ed362736f22ec-Abstract-Conference.html

Dong, Y., Mu, R., Jin, G., Qi, Y., Hu, J., Zhao, X., Meng, J., Ruan, W., & Huang, X. (2024). Position: Building guardrails for large language models requires systematic design. In Proceedings of the 41st International Conference on Machine Learning (Vol. 235, pp. 11375–11394). PMLR. https://proceedings.mlr.press/v235/dong24c.html

Geng, S., Josifoski, M., Peyrard, M., & West, R. (2023). Grammar-constrained decoding for structured NLP tasks without finetuning. In Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (pp. 10932–10952). Association for Computational Linguistics. https://doi.org/10.18653/v1/2023.emnlp-main.674

Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., Luo, X., Lo, D., Grundy, J., & Wang, H. (2024). Large language models for software engineering: A systematic literature review.

ACM Transactions on Software Engineering and Methodology, 33(8), Article 220. https://doi.org/10.1145/3695988

AI and Machine Learning Advances
Volume 2, Issue 2
ISSN: 3067-3216

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-T., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. Advances in Neural Information Processing Systems, 33, 9459–9474.

Liu, J., Wang, G., Yang, R., Zeng, J., Zhao, M., & Cai, Y. (2025). RTADev: Intention aligned multi-agent framework for software development. In Findings of the Association for Computational Linguistics: ACL 2025 (pp. 1548–1581). Association for Computational Linguistics. https://doi.org/10.18653/v1/2025.findings-acl.80

Liu, X., Yu, H., Zhang, H., Xu, Y., Lei, X., Lai, H., Gu, Y., Ding, H., Men, K., Yang, K., Zhang, S., Deng, X., Zeng, A., Du, Z., Zhang, C., Shen, S., Zhang, T., Su, Y., Sun, H., . . . Tang, J. (2024). AgentBench: Evaluating LLMs as agents. In The Twelfth International Conference on Learning Representations. https://openreview.net/forum?id=zAdUB0aCTQ

Ong, I., Almahairi, A., Wu, V., Chiang, W.-L., Wu, T., Gonzalez, J. E., Kadous, M. W., & Stoica, I. (2025). RouteLLM: Learning to route LLMs from preference data. In The Thirteenth International Conference on Learning Representations. https://proceedings.iclr.c c/paper_files/paper/2025/hash/5503a7c69d48a2f86fc00b3dc09de686-Abstr act-Conference.html

Ruan, Y., Dong, H., Wang, A., Pitis, S., Zhou, Y., Ba, J., Dubois, Y., Maddison, C. J., & Hashimoto, T. (2024). Identifying the risks of LM agents with an LM-emulated sandbox. In The Twelfth International Conference on Learning Representations. https://openreview.net/forum?id=GEcwtMk1uA

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. Advances in Neural Information Processing Systems, 30, 5998–6008.

Vir, R., Shankar, S., Chase, H., Hinthorn, W., & Parameswaran, A. G. (2025). PROMPTEVALS: A dataset of assertions and guardrails for custom production large language model pipelines. In Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Vol. 1, pp. 4225–4245). Association for Computational Linguistics. https://doi.org/10.18653/v1/2025.naacl-long.213

Vishwakarma, H., Agarwal, A., Patil, O., Devaguptapu, C., & Chandran, M. (2025). Can LLMs help you at work? A sandbox for evaluating LLM agents in enterprise environments. In Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (pp. 9167–9201). Association for Computational Linguistics. https://doi.org/10.186 53/v1/2025.emnlp-main.466

Wu, T., Terry, M., & Cai, C. J. (2022). AI chains: Transparent and controllable human–AI interaction by chaining large language model prompts. In Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems (Article 385, pp. 1–22). Association for Computing Machinery. https://doi.org/10.1145/3491102.3517582

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A. H., White, R. W., Burger, D., & Wang, C. (2024). AutoGen: Enabling next-gen LLM applications via multi-agent conversation. In Proceedings of the First Conference on Language Modeling. https://openreview.net/forum?id=tEAF9L Bdgu

Yang, J., Jimenez, C. E., Wettig, A., Lieret, K., Yao, S., Narasimhan, K., & Press, O. (2024). SWE-agent: Agent–computer interfaces enable automated software engineering. Advances in Neural Information Processing Systems, 37, 50528–50652. https://doi.org/10.52202/079017-1601