From Feature Variability to Agency Variability: A Software Product-Line Engineering Framework for Governed Agentic AI Systems

Riyadh Alsegier *

Posted Date: 12 May 2026

doi: 10.20944/preprints202605.0741.v1

Keywords: software product line engineering; agentic AI; agency variability; software variability; AI governance; AI agents; constraint solving; semantic observability; LLM-as-a-judge; human-in-the-loop

 doi:10.20944/preprints202605.0741.v1

Article

From Feature Variability to Agency Variability: A Software Product-Line Engineering Framework for Governed Agentic AI Systems

Riyadh Alsegier

Independent Researcher, Riyadh, Saudi Arabia; ralsegier@gmail.com

Abstract

Agentic artificial intelligence systems increasingly combine language models, memory, retrieval, tool use, orchestration, and human oversight. For software engineering, this creates a variability problem that feature-oriented product line methods only partly address: organizations are configuring not only functions or components, but permitted patterns of agency. Unlike MAS-SPL, which mainly structures families of agent roles and interactions, the proposed approach targets LLM-based agents whose prompts, retrieval sources, tool authority, runtime monitoring, and governance boundaries vary together. This paper proposes Agency-Centric Product Line Engineering for governed agentic AI systems. It defines agency variability as systematic variation in a system's capacity to pursue goals, perceive context, reason and plan, use tools, exercise authority, interact with humans or other agents, remain observable, and evolve under governance constraints. It also defines semantic observability as runtime monitoring of whether semantic behavior—goals, tool choices, evidence use, proposed actions, and escalation decisions—remains consistent with the granted agency profile. The paper contributes a nine-dimension Agency Variability Model, an Agency Configuration Schema, a constraint-based validation algorithm, and a prototype-style derivation of portfolio-management agent variants. The case shows that five variants can share assets while differing in perception, authority, autonomy, human control, and topology.

Keywords: software product line engineering; agentic AI; agency variability; software variability; AI governance; AI agents; constraint solving; semantic observability; LLM-as-a-judge; human-in-the-loop

1. Introduction

Software systems are increasingly assembled with foundation models, retrieval pipelines, memory mechanisms, tool interfaces, planners, and orchestration layers. The resulting systems do more than provide static functionality. In many settings they interpret goals, select information sources, use tools, draft or execute actions, and coordinate with people or other agents. This makes agentic AI a software engineering concern, not only an AI-model concern.

Software Product Line Engineering (SPLE) offers a mature response to a familiar problem: how to build families of related systems through reusable assets and managed variability. It asks what is common across a family, what varies, how variation is constrained, and how concrete products can be derived in a disciplined way. Those questions are directly relevant to organizations deploying multiple AI agents across functions and workflows.

The challenge is that many of the important differences among agentic systems are not ordinary features. A portfolio risk agent, an executive-summary agent, a compliance-checking agent, and a benefits-realization agent may all use retrieval, summarization, comparison, and drafting. Their meaningful differences lie in what they may observe, which tools they may invoke, how much planning they may perform, whether they may change records, when they must ask for approval, and how their actions are audited.

 doi:10.20944/preprints202605.0741.v1

This paper therefore proposes to complement feature variability with agency variability. The idea is not that features no longer matter. Rather, feature models need an additional layer that represents the form and degree of agency that a system variant may exercise. In other words, an agentic product line should not only configure what the system includes; it should configure what the system is allowed to do.

The paper contributes four elements: (1) a definition of agency variability for governed agentic AI systems; (2) a nine-dimension Agency Variability Model; (3) a configuration and validation method that connects agency choices to constraints, tests, and human review; and (4) an illustrative portfolio-management case with a quantitative comparison and derivation of five agentic variants.

The rest of the paper is organized as follows. Section 2 provides background. Section 3 positions the work against related literature. Section 4 explains the design-oriented research approach and validation strategy. Sections 5 and 6 present the framework, model, and derivation algorithm. Section 7 applies the framework to a portfolio-management product line. Section 8 evaluates the framework through analytical, prototype-style, and scenario-based validation. Sections 9 and 10 discuss implications, limitations, and future work.

2. Background

2.1. Software and Systems Product Line Engineering

Software Product Line Engineering develops families of related software-intensive systems through shared core assets and explicit variability. Instead of engineering each product as a separate project, SPLE separates domain engineering, where reusable assets and variation points are defined, from application engineering, where concrete variants are derived. Core ideas include product family scoping, commonality and variability analysis, feature modeling, product derivation, configuration constraints, traceability, and reuse across requirements, architecture, implementation, tests, and documentation.

Feature-based product line engineering, reflected in ISO/IEC 26580:2021, provides a practical way to represent selectable, mandatory, optional, alternative, and constrained product characteristics. This paper builds on that tradition but argues that agentic AI requires an additional variability layer, because the key difference among variants is often not whether a feature exists but how much agency is permitted when that feature is exercised.

2.2. AI-Enabled Software Systems and MLOps

AI-enabled systems combine conventional software with models, data pipelines, inference services, prompts, evaluation routines, and monitoring. Their behavior is probabilistic, data-sensitive, and context-dependent. MLOps and AI engineering practices help manage model development, deployment, monitoring, and evaluation. They are necessary for agentic systems, but they do not by themselves define a product line method for configuring tool authority, human approval, runtime action boundaries, and accountability across families of agents.

2.3. Agentic AI Systems

Agentic AI systems are composite software systems organized around goal-directed, tool-mediated behavior. A typical LLM-based agent includes a model interface, role instructions, retrieval or perception mechanisms, memory, tools, a planner or control loop, guardrails, and logging. Multi-agent systems may add role specialization, delegation, reviewer agents, manager-worker structures, shared state, and coordination protocols.

The current literature on LLM-based agents emphasizes planning, memory, tool use, reflection, coordination, evaluation, and safety. These concepts are highly relevant, but most agent frameworks treat them as implementation configurations rather than product-line variation points. The present

 doi:10.20944/preprints202605.0741.v1

paper asks how those configurations can be lifted into a software engineering framework that supports reuse, constraint checking, derivation, testing, and governance.

2.4. Governed Agency and Human Control

Governed agency means that an agent's capacity to act is bounded by explicit controls. These controls include human approval, role-based authority, data-access limits, policy checks, reversible action design, audit logging, shutdown controls, semantic monitoring, and post-deployment review. The goal is not to maximize autonomy. The goal is to calibrate agency to the task, risk, domain, and accountability requirements.

Governance frameworks such as the NIST AI Risk Management Framework, ISO/IEC 42001, and practical guidance on governing agentic AI systems emphasize accountability, lifecycle responsibilities, and human oversight. For agentic product lines, these concerns must be represented inside configuration and validation mechanisms rather than left as external policy statements.

3. Related Work and Research Gap

3.1. Software Product Lines and Variability Management

SPLE provides mature mechanisms for modeling features, constraints, common assets, and product derivation. These mechanisms are essential foundations for this paper. Their limitation in the present context is that feature models usually describe functional or quality differences among products. They are less direct when the central question is whether an agent may observe a source, plan across steps, invoke a tool, modify an external system, or escalate to a human.

3.2. Multi-Agent System Product Lines

Multi-Agent System Product Line (MAS-SPL) research is the closest historical predecessor. It showed that product line thinking can support families of agent systems rather than one-off agent-oriented applications. The difference is not that MAS-SPL is irrelevant; it is that it was developed before the current class of LLM-based, tool-using, enterprise-integrated agents became common.

A MAS-SPL approach can model agent roles, goals, protocols, and interaction patterns. It is less prepared to model several properties that are central to contemporary LLM-based agents: prompt and retrieval variability, tool-mediated authority, probabilistic runtime behavior, semantic observability, evaluator-model checks, and governance rules that must be enforced across configuration, execution, and audit. The sharper distinction is this: MAS-SPL can derive different kinds of agents; Agency Variability derives different permitted envelopes of agency for agents that may share the same functional features but differ in what they may observe, decide, invoke, execute, escalate, and record.

3.3. Machine-Learning-Enabled Product Lines

Recent ML-enabled product line work examines variability in model selection, training data, hyperparameters, performance metrics, versioning, notebooks, and deployment pipelines. Work on combining machine learning models through SPL techniques, enhancing SPLs with ML components, and feature-based versioning for ML-enabled product lines shows that AI artifacts can be brought into product-line reasoning. This literature is directly relevant, but it mainly treats ML as a component or pipeline family. It does not yet model the broader agency envelope of LLM-based systems: perception rights, tool authority, approval gates, semantic observability, and accountability for actions.

3.4. Dynamic Software Product Lines and Runtime Variability

Dynamic Software Product Lines shift some variability decisions from design time to runtime. This is highly relevant because agentic systems adapt plans, choose tools, request information, and

 doi:10.20944/preprints202605.0741.v1

respond to changing contexts while running. The proposed framework draws on this idea but narrows it toward governed agency: runtime adaptation is permitted only inside a validated agency envelope.

3.5. Generative AI, Agent Frameworks, and Agentic Software Engineering

Generative AI has also been explored as a tool for variability engineering, for example by helping infer feature models or automate migration tasks. The present paper addresses the complementary direction: using product line engineering to govern AI agents as the products being derived.

The LLM-agent literature provides a rich vocabulary around planning, memory, tool use, multi-agent coordination, evaluation, and security. Frameworks such as LangGraph, AutoGen, Microsoft Agent Framework, CrewAI, Semantic Kernel, and Pydantic AI provide implementation mechanisms for agents and workflows. The proposed framework is independent of these tools. It defines the product-line layer that should govern how any such tool is used when deriving families of agents.

Recent work on agentic software engineering further argues that software engineering may shift toward orchestration, verification, human-agent collaboration, and accountable oversight. Model-based software engineering discussions in 2026 also point to the need for traceable, reviewable artifacts when agentic AI participates in development activities. These streams complement the present work because they identify the need for new software engineering methods; this paper focuses on one such method: agency-centered product line engineering.

3.6. Research Gap

The gap can be summarized as follows. SPLE models product variability well, but agency is not yet a first-class variability construct. MAS-SPL models families of agent systems, but does not fully address LLM-based, tool-using, governed enterprise agents. ML-enabled SPL addresses model and data variability, but not action authority and human control. DSPL addresses runtime variability, but not semantic observability or agency-boundary enforcement. Agentic AI research studies agents and frameworks, but not systematic product-line derivation of governed agent variants.

The proposed contribution is therefore not a claim that agents, product lines, or governance are new. The contribution is their composition into a product-line model where agency is explicitly configured, constrained, derived, tested, monitored, and evolved. Table 1 summarizes this positioning in a neutral form.

Table 1. Adjacent research streams and the remaining gap for governed agentic product lines.

Adjacent stream	What it contributes	What remains insufficient for governed agentic product lines
Feature-based SPLE	Feature models, reusable assets, constraints, product derivation	Does not directly model authority, autonomy, semantic monitoring, or human approval as agency variables.
MAS product lines	Agent roles, interaction patterns, reusable MAS structures	Does not fully cover LLM behavior, prompt/RAG variability, tool-mediated authority, or enterprise governance controls.
ML-enabled SPL	Model, data, performance, pipeline, and deployment variability	Focuses on ML components more than action boundaries, tool authority, and accountability across agent variants.

 doi:10.20944/preprints202605.0741.v1

Dynamic SPL	Runtime reconfiguration and context-aware adaptation	Addresses runtime binding, but not whether an agent remains inside a granted agency envelope.
LLM-agent frameworks	Practical mechanisms for tools, memory, planning, and orchestration	Provide implementation configuration, but not product-line scoping, derivation, validation, and traceability.
AI governance	Risk principles, oversight, accountability, and human control	Often remains policy-level unless converted into executable constraints and variant-specific tests.

4. Materials and Methods

The paper uses a design-oriented conceptual research approach. Its purpose is to define a framework, model, and derivation method that can be examined and implemented in later empirical settings. The current version does not claim industrial validation. To avoid treating a checklist as evaluation, the paper includes three forms of validation: analytical coverage against design requirements, a prototype-style configuration and constraint-checking exercise, and a scenario-based comparison across derived variants.

4.1. Design Problem

How can software product line engineering be extended to support the systematic configuration, governance, validation, and evolution of families of agentic AI systems whose key differences concern forms and degrees of agency rather than only software features?

4.2. Design Requirements

The framework is guided by seven requirements: (DR1) model agency explicitly; (DR2) connect agency variability to feature variability; (DR3) embed governance in configuration; (DR4) support runtime adaptation within constraints; (DR5) represent human control as a configurable variable; (DR6) reuse non-code assets such as prompts, policies, tools, tests, and logs; and (DR7) manage evolution of models, prompts, tools, policies, and authority boundaries.

4.3. Validation Strategy

The validation is intentionally modest but more than descriptive. First, the framework is checked against the design requirements. Second, a prototype-style derivation algorithm is specified in Section 6. It is applied to five portfolio-management variants, including one detailed derivation path for the Risk Monitoring Agent. Third, a quantitative comparison table scores the variants across the nine agency dimensions. These scores are not empirical performance measurements; they are structured configuration measures that make variation visible and inspectable.

A Delphi study or industrial expert evaluation would strengthen the paper and is identified as a next step, but it is not presented as completed evidence. This distinction is important: the current paper establishes conceptual and configuration-level validity, not empirical effectiveness in production organizations.

5. Results: Agency-Centric Product Line Engineering Framework

5.1. Core Idea

Agency-Centric Product Line Engineering extends conventional product line thinking by adding an Agency Model beside the Feature Model. A feature model asks which capabilities or functions a

 doi:10.20944/preprints202605.0741.v1

variant includes. An agency model asks under which goals, boundaries, authority levels, and human-control rules those capabilities may be exercised.

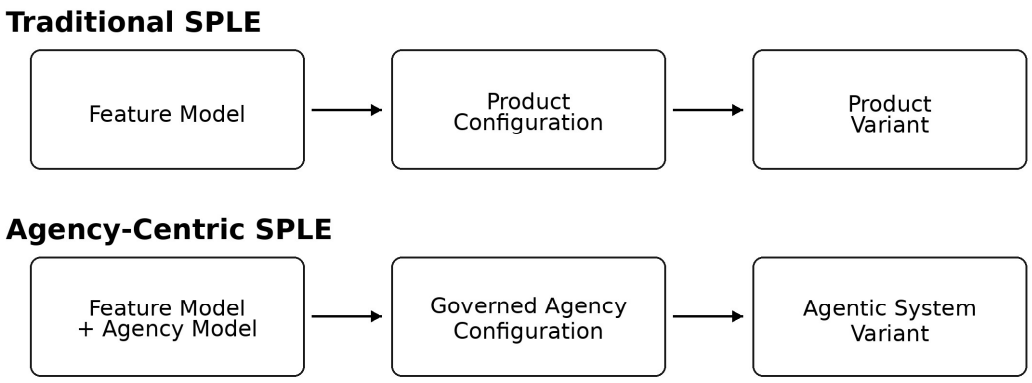

Figure 1. From feature variability to agency variability. Traditional SPLE centers on feature selection; the proposed extension adds agency configuration as a first-class product-line artifact.

5.2. Framework Layers

The framework has seven layers. The System Family Layer defines the product family and variant scope. The Capability Layer identifies reusable AI capabilities such as retrieval, summarization, extraction, classification, planning, recommendation, and tool use. The Agency Variability Layer models how agency differs across variants. The Core Assets Layer contains models, prompts, retrieval schemas, tool adapters, policy templates, evaluation suites, logs, and workflow components. The Governance Layer converts policy into constraints and required controls. The Runtime Adaptation Layer defines what may change during execution and how deviations are detected. The Evolution Layer manages changes to models, tools, policies, memory, tests, and authority profiles.

 doi:10.20944/preprints202605.0741.v1

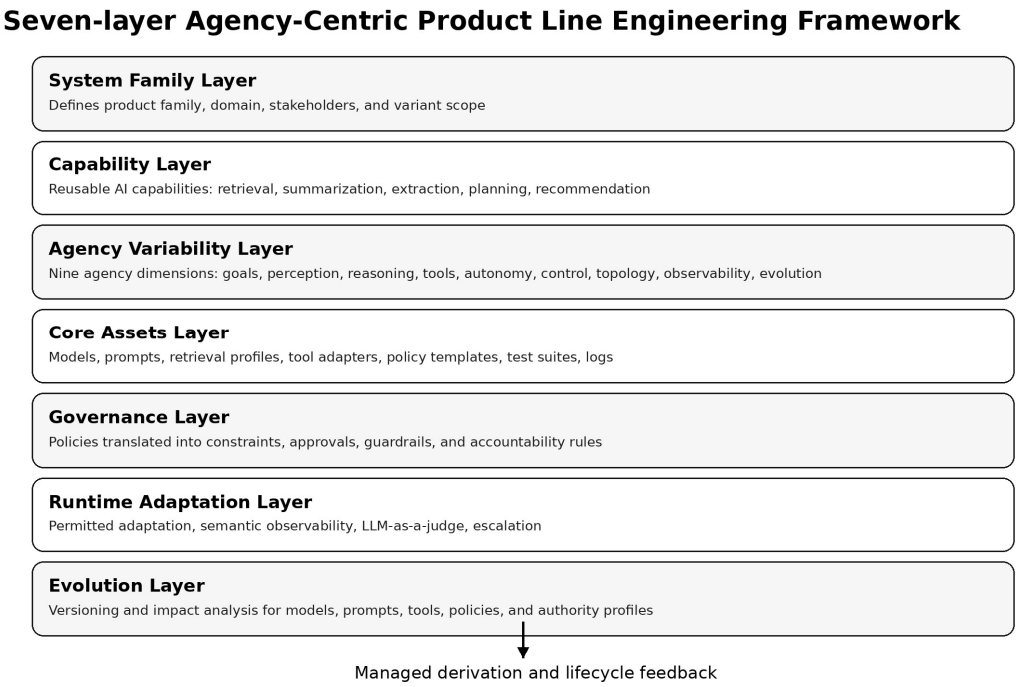

Figure 2. Seven-layer Agency-Centric Product Line Engineering framework. The framework connects product-family scoping, reusable AI capabilities, agency variability, core assets, governance, runtime adaptation, and lifecycle evolution.

5.3. Runtime Adaptation and Semantic Observability

Runtime adaptation is necessary because agentic systems may encounter missing data, conflicting instructions, tool failures, low-confidence outputs, or policy conflicts. The framework therefore separates permitted adaptation from prohibited self-expansion. An agent may select among approved tools, ask for more information, revise a plan, or escalate. It may not grant itself new permissions, access unapproved data, or execute irreversible actions outside its agency profile.

Semantic observability is the runtime mechanism that helps enforce this boundary. It monitors semantic events: selected goals, retrieved evidence, planned steps, tool calls, proposed actions, escalation decisions, blocked actions, and evaluator judgments. A practical pipeline has four stages. First, the agent emits a structured trace for each meaningful step, including goal, evidence, tool, proposed action, confidence signal where available, and policy context. Second, deterministic checks compare the trace with the agency configuration, for example whether the tool is allowed, whether the source is approved, and whether the action requires approval. Third, semantic checks use evaluator models, including LLM-as-a-judge where appropriate, to assess whether the proposed action is consistent with the variant's role, authority boundary, evidence base, and escalation rules. Fourth, the runtime controller either permits the step, blocks it, requests clarification, or escalates it to a human reviewer. The evaluator is therefore not a final decision-maker. It is a semantic monitor that supports audit, triage, and human control.

5.4. Evolution Layer

Agentic variants evolve when a model changes, a prompt is revised, a tool is added, a data source is removed, a policy changes, or an autonomy level is raised or lowered. The Evolution Layer requires versioning and impact analysis across all affected variants. For example, adding write access to a project-management tool may require new approval workflows, stronger logging, a revised security review, and regression tests for blocked actions.

 doi:10.20944/preprints202605.0741.v1

6. Results: Agency Variability Model and Derivation Method

6.1. Definition

Agency variability is the systematic variation in an AI-enabled system's capacity to pursue goals, perceive context, reason and plan, use tools, exercise authority, interact with humans or other agents, remain observable and secure, and evolve under explicit governance constraints.

6.2. Nine Agency Variability Dimensions

The earlier version of this framework separated agency into fourteen dimensions. This revision consolidates overlapping concerns into nine dimensions to improve clarity and reduce double counting. For example, reasoning and planning are grouped because planning depth is usually a specialized form of reasoning behavior; tool use and authority are grouped because tool access only matters once the allowed action boundary is specified; and escalation is grouped with human control because escalation is one form of human involvement.

Table 2. Nine agency variability dimensions and typical configuration questions.

Dimension	Definition	Typical configuration questions
D1 Goal and role scope	What objectives and organizational role the variant may pursue.	Is the agent task-specific, workflow-specific, role-specific, domain-specific, or portfolio-level?
D2 Perception and memory scope	What the agent may observe, retrieve, remember, and reuse.	Which repositories, dashboards, records, modalities, and memory boundaries are allowed?
D3 Reasoning and planning mode	How the agent analyzes information and decomposes goals.	May it summarize, classify, compare, recommend, plan multi-step actions, or revise plans after feedback?
D4 Tool use and action authority	Which tools the agent may use and what external changes it may cause.	May it read, draft, notify, create tasks, update records, approve, or trigger workflows?
D5 Autonomy and human control	How independently the agent may proceed and where humans intervene.	Is the agent advisory, interactive, approval-gated, bounded autonomous, or continuously supervised?
D6 Interaction and orchestration topology	How the agent interacts with users and other agents.	Is it user-initiated, proactive, reviewer-based, manager-worker, peer-to-peer, or multi-agent?
D7 Governance and guardrails	Which policy, security, and safety controls constrain behavior.	What data, privacy, identity, least-privilege, prompt-injection, and sandboxing controls apply?
D8 Accountability and semantic observability	How decisions, traces, evidence, and deviations are recorded and reviewed.	What logs, evidence links, LLM-judge checks, action ledgers, and audit trails are required?

 doi:10.20944/preprints202605.0741.v1

D9 Evolution and learning	How the variant changes over time.	Can models, prompts, memory, tools, policies, and autonomy levels change, and under what review?

6.3. Agency Configuration Schema

The schema below illustrates how the nine dimensions can be represented as a reviewable and machine-checkable configuration. It is deliberately implementation-neutral: it can be mapped to LangGraph, AutoGen, Microsoft Agent Framework, CrewAI, Semantic Kernel, Pydantic AI, or a custom orchestration layer.

```
agency_configuration:
 variant_id: portfolio_risk_agent_v2
 family: governed_portfolio_agents
 selected_features: [risk_monitoring, retrieval, summarization, recommendation]
 agency_dimensions:
 goal_role_scope: workflow_specific_risk_monitor
 perception_memory_scope:
 allowed_sources: [risk_register, status_reports, milestone_dashboard]
 prohibited_sources: [personal_hr_records, unapproved_finance_data]
 memory: project_level
 reasoning_planning_mode: bounded_multistep_analysis
 tool_action_authority:
 read_tools: [document_repository, project_dashboard]
 draft_tools: [mitigation_task_draft, risk_summary_draft]
 execute_tools: [internal_notification]
 prohibited_actions: [record_update_without_approval, external_message]
 autonomy_human_control: L3_approval_gated
 interaction_topology: single_agent_with_human_reviewer
 governance_guardrails: [least_privilege, source_filtering, prompt_injection_checks]
 accountability_observability: [evidence_log, tool_call_log, llm_judge_policy_check]
 evolution_learning: human_reviewed_updates_only
 required_tests:
 - source_access_test
 - tool_permission_test
 - blocked_action_test
 - evidence_traceability_test
 - semantic_observability_test
 - approval_path_test
```

6.4. Constraint-Based Derivation and Validation Algorithm

A configuration schema is useful only if it supports validation. The proposed derivation algorithm treats agency configuration as a constrained product-line derivation problem. It combines feature selections, agency dimensions, core-asset compatibility, governance rules, security policies, and required tests. SAT, SMT, or constraint satisfaction techniques can then reject invalid variants before deployment and can generate a human-reviewable explanation of the conflict.

Algorithm 1 is intentionally lightweight, but it is written as executable design logic rather than as a purely narrative procedure.

Algorithm 1. *Derive and validate an agentic product-line variant*

Input: FeatureSelection F, AgencyConfiguration A, CoreAssets C, GovernanceRules G, SecurityRules S, TestTemplates E

Output: DerivedVariant V, RequiredTests T, DerivationRecord R, or InvalidConfiguration

 doi:10.20944/preprints202605.0741.v1

1. Initialize an empty constraint set C_all and an empty derivation record R.
2. Declare variables from F and A: selected features, agency levels, tool permissions, data-source permissions, authority flags, human-control flags, topology choices, observability flags, and evolution rules.
3. For each selected feature f in F:
3.1 Add feature-model constraints for mandatory, optional, alternative, and excluded features.
3.2 Add asset constraints requiring at least one compatible prompt, retrieval profile, tool adapter, guardrail profile, and test template where applicable.
4. For each agency dimension d in A:
4.1 Encode the selected level of d as a finite-domain or Boolean variable.
4.2 Add cross-dimension rules, such as: write authority requires audit and rollback; multi-agent topology requires an accountability owner; broad perception requires source filtering.
5. For each governance or security rule r in G union S:
5.1 Translate r into a Boolean, finite-domain, or policy-as-code constraint.
5.2 Attach a human-readable rationale to R for later review.
6. Solve C_all using a SAT, SMT, or constraint satisfaction engine.
7. If C_all is unsatisfiable, return InvalidConfiguration with the unsat core, conflicting rules, and the smallest set of agency choices that must change.
8. If C_all is satisfiable:
8.1 Bind selected assets to a variant scaffold: prompt set, retrieval profile, tool-permission profile, approval workflow, guardrail profile, logging schema, and semantic-observability probes.
8.2 Generate RequiredTests T from the satisfied agency profile and TestTemplates E.
8.3 Add residual risks, human-review items, and version identifiers to R.
9. Return DerivedVariant V, RequiredTests T, and DerivationRecord R.

The constraint layer can be expressed through Boolean implications and finite-domain rules. Let ext_notify mean that external notification is enabled, approve mean that human approval is configured, write mean that the agent can write to a system of record, audit mean that audit logging is enabled, rollback mean that a rollback rule exists, L4 mean bounded autonomy, sem_obs mean semantic observability is enabled, swarm mean that a swarm-like topology is selected, owner mean that an accountability owner is defined, and conflict_rule mean that an inter-agent conflict rule exists. Example formulas include:

C1: ext_notify -> approve, encoded as (not ext_notify OR approve).

C2: write -> audit AND rollback, encoded as (not write OR audit) AND (not write OR rollback).

C3: L4 -> sem_obs AND audit, encoded as (not L4 OR sem_obs) AND (not L4 OR audit).

C4: swarm -> owner AND conflict_rule, encoded as (not swarm OR owner) AND (not swarm OR conflict_rule).

A concrete invalid assignment is ext_notify = true, approve = false. Under C1, (not true OR false) evaluates to false, so the solver rejects the configuration. A concrete valid Risk Monitoring assignment may set write = false, audit = true, rollback = true, L4 = false, sem_obs = true, swarm = false, owner = true. The same formulas therefore do two things: they reject unsafe configurations and document why a permitted configuration remains inside its agency boundary.

 doi:10.20944/preprints202605.0741.v1

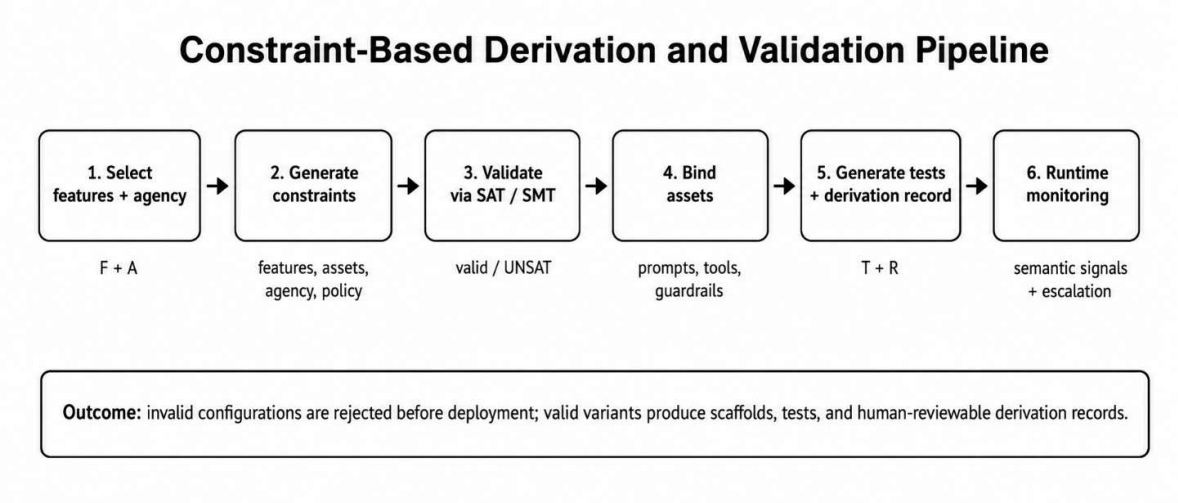

Figure 3. Constraint-based derivation and validation pipeline. Agency configurations are normalized, translated into constraints, checked, and used to generate variant scaffolds, tests, and human-review records.

7. Results: Illustrative Case Application

7.1. Case Context and Shared Assets

Portfolio and initiative management is suitable for illustration because it involves structured data, documents, governance routines, risk thresholds, executive reporting, stakeholder coordination, and decision support. The product line includes shared assets such as an approved LLM gateway, retrieval service, initiative status parser, risk register adapter, dashboard connector, executive-briefing template, governance checklist, evidence citation component, human approval workflow, audit log service, guardrail library, and evaluation suite.

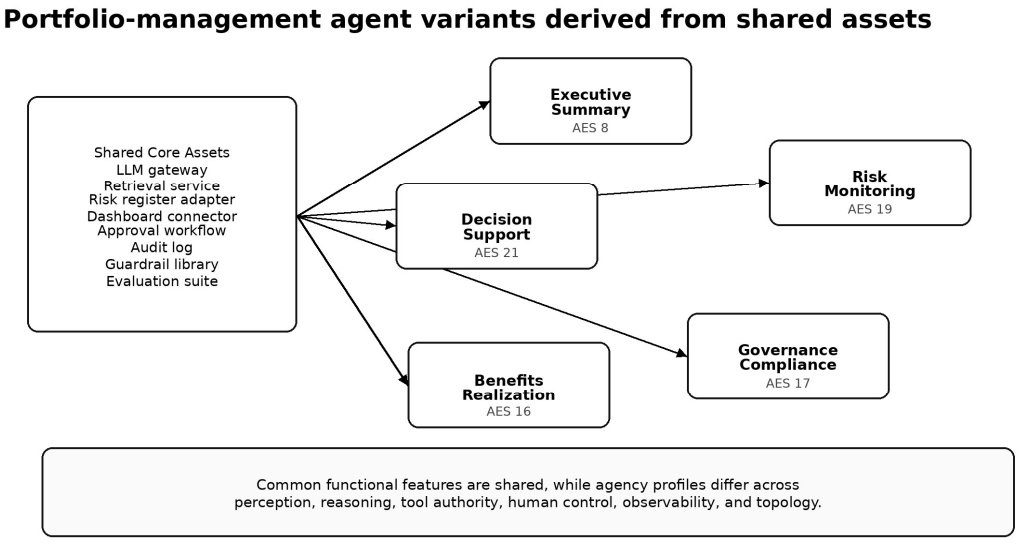

Figure 4. Portfolio-management agent variants derived from shared assets. Functionally similar agents share retrieval, logging, approval, and evaluation assets while differing in agency configuration.

7.2. Derived Variants

 doi:10.20944/preprints202605.0741.v1

Five variants are derived: an Executive Summary Agent, Risk Monitoring Agent, Governance Compliance Agent, Benefits Realization Agent, and Decision Support Agent. They share several functional features but differ in agency profile. The point of the case is to show that feature similarity can hide agency differences.

Table 3 uses a simple 0-3 scale for each dimension: 0 = not applicable or minimal, 1 = low, 2 = moderate, and 3 = high. The Agency Exposure Score (AES) is reported as an unweighted sum for transparency, not because all dimensions are assumed to carry equal risk. The unweighted version is intentionally easy for reviewers to audit by eye. For sensitivity analysis, organizations could replace it with a weighted score, AES_w = sum(w_i * d_i). For example, in a regulated workflow, authority, perception, and guardrails could be weighted more heavily: w_authority = 2.0, w_perception = 1.5, w_guardrails = 1.5, and all other weights = 1.0. The table's purpose is comparative rather than predictive: to show that functionally similar variants can differ substantially in agency exposure.

Table 3. Quantitative comparison of portfolio-management agent variants.

Variant	Shared functional features	D1-D9 score vector	AES	Interpretation
Executive Summary	Retrieval, summarization, comparison	[1,1,1,0,1,0,1,2,1]	8	Low agency exposure; read-only advisory use with evidence logging.
Risk Monitoring	Retrieval, summarization, comparison, drafting	[2,2,2,2,3,1,2,3,2]	19	Moderate-high exposure; approval-gated drafting and escalation are required.
Governance Compliance	Retrieval, checklist comparison, drafting	[2,2,2,1,2,1,2,3,2]	17	Moderate exposure; strong traceability and review prevent over-enforcement.
Benefits Realization	Retrieval, comparison, variance analysis	[2,2,2,1,2,1,2,3,2]	17	Moderate exposure; analytical autonomy is separated from record authority.
Decision Support	Retrieval, synthesis, comparison, decision drafting	[3,3,3,1,3,1,3,3,2]	22	Highest exposure; broad perception and decision influence require stronger controls.

Table 4. Comparison with an ad hoc agent-building approach.

Criterion	Ad hoc agent development	Agency-Centric Product Line Engineering
Reuse	Prompts, tools, and tests are often copied informally.	Reusable assets are selected from a governed product-line catalog.
Validity	Invalid authority or tool combinations may be discovered late.	Constraints reject invalid configurations before derivation.
Traceability	Design rationale is scattered across code, prompts, and documents.	A derivation record links features, agency choices, assets, and controls.
Testing	Tests are manually selected per agent.	Required tests are generated from the agency profile.
Governance	Human approval and audit rules vary by team.	Approval, audit, and guardrails are configuration rules.

Compared with the ad hoc baseline, the framework does not claim to prove superior runtime performance. Its immediate advantage is engineering discipline: it makes agency choices explicit, checkable, reproducible, and reviewable before implementation.

Table 5. Example derivation path for the Risk Monitoring Agent.

 doi:10.20944/preprints202605.0741.v1

Step	Input from product line	Agency decision	Generated artifact/control	Validation check
1. Scope variant	Portfolio family; risk-monitoring feature	Workflow-specific risk monitor; not a general decision maker	Risk-agent role template and prompt family	Scope-boundary check
2. Bind sources	Risk register, status reports, milestone dashboard	Allow project-level perception; exclude HR and unapproved finance data	Retrieval profile with denied-source rules	Source-access and denied-source tests
3. Configure reasoning	Risk classification, evidence comparison, mitigation templates	Bounded multi-step reasoning; no open-ended strategic planning	Planner scaffold and recommendation template	Classification and hallucination tests
4. Configure tools	Read dashboard; draft mitigation task; notify internally	Draft and notify only; no record updates without approval	Tool permission profile and blocked-action handler	Unauthorized-action test
5. Add human control	Approval workflow; risk threshold policy	Approval for status changes and mitigation tasks	Approval path and escalation rules	Approval-path and escalation-trigger tests
6. Add observability	Audit log; evaluator model; policy checker	Log tool calls, evidence, proposed actions; LLM judge flags deviations	Semantic observability dashboard	Policy-compliance and judge-calibration tests
7. Produce derivation record	Assets, constraints, tests, residual assumptions	Human reviewer confirms validity before deployment	Variant scaffold and review record	Human review sign-off

This derivation path is deliberately concrete. It shows how the same core assets are transformed into a governed variant through agency choices and validation steps. The Risk Monitoring Agent is not merely described; it is derived through scoped feature selection, source binding, reasoning configuration, tool authority, human control, semantic observability, and test generation.

8. Discussion and Preliminary Evaluation

8.1. Analytical Coverage

The analytical evaluation checks whether the framework satisfies the seven design requirements. Unlike a purely narrative checklist, the requirements are tied to specific framework mechanisms: the nine-dimension model, the configuration schema, the constraint-based derivation algorithm, the semantic observability pipeline, the quantitative comparison table, and the derivation record.

Table 6. Design requirement coverage.

Requirement	Mechanism in the framework	Evidence in the paper

 doi:10.20944/preprints202605.0741.v1

DR1 Explicit agency model	Nine agency dimensions	Section 6.2
DR2 Link to feature variability	Feature selection plus agency configuration	Sections 5.1, 6.3, 7.2
DR3 Governance in configuration	Governance and security constraints	Sections 6.3-6.4
DR4 Runtime adaptation	Semantic observability and allowed adaptation rules	Section 5.3
DR5 Human control	Approval, escalation, review, sign-off	Sections 6.2, 7.3
DR6 Non-code asset reuse	Prompts, tools, retrieval, guardrails, logs, tests	Sections 5.2, 7.1
DR7 Lifecycle evolution	Evolution dimension and impact analysis	Sections 5.4, 6.2

8.2. Prototype-Style Configuration Validation

The second validation applies Algorithm 1 to the portfolio-management variants. The validation checks whether each variant has a complete agency configuration, whether required controls are generated, and whether invalid combinations can be detected. The Risk Monitoring Agent derivation in Table 5 is the detailed example. A prototype implementation could represent the configuration in YAML and translate constraints into SAT or SMT clauses. The present paper does not report a deployed tool; it specifies the algorithm and demonstrates its application at configuration level.

8.3. Scenario-Based Comparison

The scenario comparison in Table 3 supports the central argument: functionally similar variants can have different agency profiles. Executive Summary and Decision Support both use retrieval and summarization, but the latter has broader perception, deeper reasoning, higher decision influence, and stronger accountability requirements. Risk Monitoring and Governance Compliance both draft follow-up artifacts, but their authority and escalation paths differ. This comparison does not prove organizational effectiveness; it demonstrates that agency variability captures distinctions that feature lists alone would obscure.

8.4. Evaluation Boundaries

The evaluation remains limited. It does not include a Delphi study, industrial deployment, user study, or measured productivity effect. Its value is methodological: it shows that the framework can express, validate, derive, and compare governed agentic variants in a structured way. A future Delphi study with software architects, AI engineers, governance specialists, and product line experts should assess completeness, usability, and external validity.

8.5. Delphi Validation Protocol for Future Empirical Refinement

No Delphi study has been conducted in this version, and the paper should not imply expert consensus. To make the empirical path concrete, a follow-up Delphi study should recruit 12-18 experts across SPLE, AI engineering, agent frameworks, AI governance, and software architecture. Round 1 would ask experts to rate the necessity, clarity, and overlap of the nine dimensions and to suggest missing constraints. Round 2 would present the revised model, aggregated feedback, and candidate weights for AES_w. Round 3 would test convergence on the dimension set, constraint categories, and validation outputs. Agreement could be summarized using interquartile ranges and Kendall's W, while qualitative comments would be coded for additions, merges, and boundary conditions. This protocol would turn the current conceptual validation into an expert-refined model without overstating the evidence available in this paper.

9. Discussion

 doi:10.20944/preprints202605.0741.v1

9.1. Implications for Software Architecture

Agentic systems require architectures that separate capability from authority. A model may be capable of proposing an action, but the architecture must decide whether the system may execute it, draft it, escalate it, or block it. Tool adapters, permission services, policy engines, approval workflows, semantic logs, and monitoring components therefore become central architectural assets.

9.2. Implications for Variability Modeling

The paper suggests that feature models should be complemented by agency models. A future modeling language could include constructs for agency dimensions, authority boundaries, human control, semantic observability, orchestration topology, and guardrail profiles. These constructs could be connected to existing feature-modeling tools and constraint solvers.

9.3. Implications for AI Governance

AI governance becomes stronger when it is translated into configuration rules. For example, a rule requiring human approval for external communication can prevent derivation of a variant that includes external messaging without an approval path. A rule requiring traceability for high-impact recommendations can automatically generate evidence-citation and audit tests.

9.4. Implications for Tooling

The framework points toward practical tools: agency variability editors, configuration validators, policy-to-constraint compilers, test generators, derivation-record generators, and runtime observability dashboards. These tools would not replace human judgment. They would make agency choices inspectable before deployment and traceable after deployment.

10. Limitations and Future Work

This paper is conceptual and prototype-style, not empirically validated. It proposes a framework, model, algorithm, illustrative scoring method, and case-based derivation, but it does not report a completed Delphi study, industrial case study, controlled experiment, or deployment measurement. The nine dimensions and AES scoring are therefore tentative. They should be treated as a structured starting point for expert review and field validation, not as a finalized standard.

The framework also assumes that organizations can express governance rules with enough precision to support configuration checks. Many organizations lack mature AI governance, data governance, or product line engineering practices. In such settings, the framework may initially serve as a diagnostic tool rather than a fully automated derivation method.

Future work should pursue four directions. First, conduct a Delphi study or expert evaluation to refine the dimensions and constraints. Second, implement a lightweight validator that translates agency configurations into SAT or SMT constraints. Third, test the framework in industrial or public-sector agent portfolios. Fourth, extend the model to multi-agent ecosystems where agency varies not only within agents but also across delegation, conflict resolution, shared memory, and accountability structures.

11. Conclusions

Agentic AI changes the variability problem facing software engineering. Organizations are not only choosing features or models; they are deciding how much agency each system variant may exercise. This paper proposed Agency-Centric Product Line Engineering as a way to configure, validate, derive, and monitor families of governed agentic AI systems. Its central construct, agency variability, reframes product-line reasoning around goals, perception, reasoning and planning, tool authority, human control, topology, guardrails, observability, and evolution. The portfolio-management case showed how functionally similar agents can differ meaningfully in agency profile,

 doi:10.20944/preprints202605.0741.v1

and how those differences can be made explicit through configuration, constraints, tests, and derivation records. The next step is empirical: expert review, prototype tooling, and industrial validation.

Supplementary Materials: The following supporting information can be downloaded at the website of this paper posted on Preprints.org, Supplementary File S1: Agency Configuration and Validation Examples, including a YAML agency configuration example, Boolean constraint examples, validation pseudocode, and a derivation trace for the Risk Monitoring Agent.

Funding: This research received no external funding.

Informed Consent Statement: Not applicable.

Data Availability Statement: No empirical datasets were generated or analyzed in this study. The conceptual models, illustrative configuration schema, and derivation logic are included within the manuscript and Supplementary File S1.

Conflicts of Interest: The author declares no conflicts of interest.

Software/Code Availability Statement: No executable software artifact was developed as part of this study. The validation logic and configuration schema are presented as conceptual artifacts. Future work will implement these artifacts as a prototype validator.

Abbreviations

The following abbreviations are used in this manuscript:

AI	Artificial Intelligence
AES	Agency Exposure Score
API	Application Programming Interface
DSPL	Dynamic Software Product Line
GenAI	Generative Artificial Intelligence
HITL	Human-in-the-Loop
LLM	Large Language Model
MAS	Multi-Agent System
MAS-SPL	Multi-Agent System Software Product Line
MDPI	Multidisciplinary Digital Publishing Institute
ML	Machine Learning
MLOps	Machine Learning Operations
RAG	Retrieval-Augmented Generation
SAT	Boolean Satisfiability
SMT	Satisfiability Modulo Theories
SPLE	Software Product Line Engineering
SPL	Software Product Line
UNSAT	Unsatisfiable
YAML	YAML Ain't Markup Language

Appendix A. Agency Configuration Checklist

1. What system family is being engineered?
2. Which variants are in scope?
3. Which features are shared?
4. Which agency dimensions vary?
5. What can each variant observe, remember, and use?
6. Which actions require approval?
7. Which constraints invalidate a configuration?
8. Which tests are generated?
9. Which runtime semantic signals are monitored?

 doi:10.20944/preprints202605.0741.v1

10. How are model, prompt, tool, policy, and authority changes versioned?

References

1. Kang, K.C.; Cohen, S.G.; Hess, J.A.; Novak, W.E.; Peterson, A.S. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Carnegie Mellon University, Software Engineering Institute, 1990.
2. Clements, P.; Northrop, L. Software Product Lines: Practices and Patterns. Addison-Wesley, 2001.
3. Pohl, K.; Boeckle, G.; van der Linden, F. Software Product Line Engineering: Foundations, Principles and Techniques. Springer, 2005.
4. Apel, S.; Batory, D.; Kaestner, C.; Saake, G. Feature-Oriented Software Product Lines: Concepts and Implementation. Springer, 2013.
5. ISO/IEC. ISO/IEC 26580:2021 Software and Systems Engineering -- Methods and Tools for the Feature-Based Approach to Software and Systems Product Line Engineering. International Organization for Standardization, 2021.
6. Pena, J.; Hinchey, M.G.; Ruiz-Cortes, A. Multi-Agent System Product Lines: Challenges and Benefits. Communications of the ACM 2006, 49, 82-84.
7. Bencomo, N.; Hallsteinsen, S.; de Almeida, E.S. A View of Dynamic Software Product Lines. In Dynamic Software Product Lines; Springer, 2012.
8. Bashari, M.; Bagheri, E.; Du, W. Dynamic Software Product Line Engineering: A Reference Framework. Software and Systems Modeling, 2017.
9. Aguayo, C.P.; Sepulveda, S. Variability Management in Self-Adaptive Systems through Deep Learning: A Dynamic Software Product Line Approach. Electronics 2024, 13, 905.
10. Gomez-Vazquez, M.; Cabot, J. Exploring the Use of Software Product Lines for the Combination of Machine Learning Models. In Proceedings of SPLC 2024, 2024.
11. Greiner, S.; Schmid, K.; Berger, T.; Krieter, S.; Meixner, K. Generative AI and Software Variability: A Research Vision. In Proceedings of VaMoS 2024, 2024.
12. Amershi, S.; Begel, A.; Bird, C.; DeLine, R.; Gall, H.; Kamar, E.; Nagappan, N.; Nushi, B.; Zimmermann, T. Software Engineering for Machine Learning: A Case Study. ICSE-SEIP, 2019.
13. Sculley, D.; Holt, G.; Golovin, D.; Davydov, E.; Phillips, T.; Ebner, D.; Chaudhary, V.; Young, M.; Crespo, J.F.; Dennison, D. Hidden Technical Debt in Machine Learning Systems. NeurIPS, 2015.
14. ISO/IEC. ISO/IEC 22989:2022 Information Technology -- Artificial Intelligence -- Artificial Intelligence Concepts and Terminology. International Organization for Standardization, 2022.
15. ISO/IEC. ISO/IEC 42001:2023 Information Technology -- Artificial Intelligence -- Management System. International Organization for Standardization, 2023.
16. National Institute of Standards and Technology. Artificial Intelligence Risk Management Framework (AI RMF 1.0). NIST AI 100-1, 2023.
17. National Institute of Standards and Technology. Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile. NIST AI 600-1, 2024.
18. OpenAI. Practices for Governing Agentic AI Systems. OpenAI, 2023/2025.
19. Russell, S.; Norvig, P. Artificial Intelligence: A Modern Approach, 4th ed. Pearson, 2020.
20. Wang, L.; Ma, C.; Feng, X.; Zhang, Z.; Yang, H.; Zhang, J.; Chen, Z.; Tang, J.; Chen, X.; Lin, Y.; et al. A Survey on Large Language Model Based Autonomous Agents. Frontiers of Computer Science, 2024.
21. Guo, T.; Chen, X.; Wang, Y.; Chang, R.; Pei, S.; Chawla, N.V.; Wiest, O.; Zhang, X. Large Language Model Based Multi-Agents: A Survey of Progress and Challenges. arXiv, 2024.
22. Shahriar, A.; Rahman, M.N.; Ahmed, S.; Sadeque, F.; Parvez, M.R. A Survey on Agentic Security: Applications, Threats and Defenses. arXiv:2510.06445, 2025.
23. Alenezi, M. Rethinking Software Engineering for Agentic AI Systems. arXiv:2604.10599, 2026.
24. Chechik, M.; Combemale, B.; Gray, J.; Rumpe, B. Agentic AI in the Next Frontier of Model-Based Software Engineering: The Arrival of AI-Hyper-Agile Software Engineering Methods? Software and Systems Modeling 2026, 25, 315-317.
25. LangChain. LangGraph Documentation. 2026.
26. Microsoft. AutoGen Documentation. 2026.

 doi:10.20944/preprints202605.0741.v1

27. Microsoft. Microsoft Agent Framework Documentation. 2026.
28. Microsoft. Semantic Kernel Documentation. 2026.
29. CrewAI. CrewAI Documentation. 2026.
30. Pydantic. Pydantic AI Documentation. 2026.
31. Mitchell, M.; Wu, S.; Zaldivar, A.; Barnes, P.; Vasserman, L.; Hutchinson, B.; Spitzer, E.; Raji, I.D.; Gebru, T. Model Cards for Model Reporting. FAT*, 2019.
32. Gebru, T.; Morgenstern, J.; Vecchione, B.; Vaughan, J.W.; Wallach, H.; Daume III, H.; Crawford, K. Datasheets for Datasets. Communications of the ACM 2021, 64, 86-92.
33. Raji, I.D.; Smart, A.; White, R.N.; Mitchell, M.; Gebru, T.; Hutchinson, B.; Smith-Loud, J.; Theron, D.; Barnes, P. Closing the AI Accountability Gap: Defining an End-to-End Framework for Internal Algorithmic Auditing. FAT*, 2020.
34. Bass, L.; Clements, P.; Kazman, R. Software Architecture in Practice, 4th ed. Addison-Wesley, 2021.
35. Cobaleda, L.-V.; Carvajal, J.; Vallejo, P.; Lopez, A.; Mazo, R. Enhancing Software Product Lines with Machine Learning Components. arXiv:2510.27640, 2025.
36. Preuner, M.; Grunbacher, P.; de Paula Filho, P.L.; Egyed, A. Feature-Based Versioning for ML-Enabled Product Lines. In Software Engineering and Advanced Applications: 51st Euromicro Conference, SEAA 2025 Proceedings; LNCS 16081; Springer, 2026; pp. 3-19. https://doi.org/10.1007/978-3-032-04190-6_1.
37. Kagal, L.; Noy, N.; Guha, R.; et al. Agentic AI Security: Threats, Defenses, Evaluation, and Open Challenges. arXiv:2510.23883, 2025.