International Journal of Research Publications in Engineering, Technology and Management (IJRPETM)

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 9, Issue 2, March - April 2026||

DOI:10.15662/IJRPETM.2026.0902022

Reimagining Enterprise SaaS Products with AI: An Architect's Perspective on Infusing AI Use Cases Without Breaking the Product

Viswanathan Tiruchindurai Srinivasan

Associate Vice President, Technology Consulting, Persistent Systems, Texas, USA

viswanathan_t@persistent.com

Publication History: Received: 19.03.2026; Revised: 11.04.2026; Accepted: 14.04. 2026; Published: 20.04.2026.

ABSTRACT: Most enterprise SaaS "AI" features are bolted onto existing screens rather than reasoned into the workflows they are meant to improve, producing a predictable pattern: a launch announcement followed by a quiet plateau in adoption. This article argues that durable value comes from reimagining specific, currently manual workflows end-to-end, not from adding AI surfaces, and that multi-tenant SaaS products carry a distinct risk profile - cross-tenant data leakage, autonomous actions inside a customer's live account, and erosion of the trust the subscription relationship depends on - that generic enterprise AI governance does not address. The article proposes a risk-tiered autonomy model scoped to tenant isolation, a guardrail architecture for actions taken inside customer accounts, a four-step implementation roadmap (tenant-aware knowledge base, retrieval-augmented generation pipeline, model testing, integration and deployment), and a consolidated checklist for product and engineering leaders reimagining SaaS workflows around AI.

KEYWORDS: Agentic AI, SaaS Product Design, Multi-Tenancy, Retrieval-Augmented Generation (RAG), AI Governance, Customer Trust, Enterprise Architecture

I. INTRODUCTION

Every enterprise SaaS roadmap in the last two years has an "AI" line item. Most of them describe the same pattern: a chat box bolted onto an existing screen, a "summarize" button added to a list view, an assistant panel that answers questions about data the product already displays. These features ship, get a launch announcement, and then quietly plateau in usage - because they were added to the product rather than reasoned into it.

The organizations getting durable value from AI are not the ones with the most AI buttons. They are the ones that treated AI as an opportunity to reimagine specific workflows end-to-end - replacing multi-step, manual processes with a system that understands intent, retrieves the right context, and either recommends or performs the next step - while protecting the two things that make a SaaS product sellable at all: the reliability customers depend on, and the trust that their data is handled correctly in a multi-tenant environment.

This article examines why AI infusion into enterprise SaaS products creates a distinct set of failure modes beyond the general challenges of enterprise AI and lays out an architectural and organizational approach for reimagining product workflows around AI use cases without introducing reliability, security, or trust issues that undermine the product itself.

Part 1: Retrofitting AI vs. Reimagining the Product

The bolt-on trap. The fastest way to ship an AI feature is to add a new surface to the existing product - a chat panel, a "generate" button, and a sidebar assistant. This is also the fastest way to produce a feature customers try once and abandon, because it treats AI as an add-on capability rather than a redesign of how a task gets done. The pattern is recognizable: the feature duplicates functionality the product already has; it lives in a UI surface disconnected from where the underlying task actually happens; usage is measured by clicks on the AI feature itself, not whether the task got faster; and no existing manual step was removed as a result of adding it - the old way remains the default path.

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 9, Issue 2, March - April 2026||

DOI:10.15662/IJRPETM.2026.0902022

Reimagining workflow, not decorating a screen. The alternative starts with a specific, already-painful workflow and asks what it looks like if the product could understand intent and act on the customer's behalf, not what button could be added next to it. A reimagined workflow replaces or compresses existing manual steps rather than adding new ones alongside them; its value is measurable in the same terms the product already reports on - time to complete a task, error rate, screens visited, support tickets generated; it has an explicit owner accountable for the workflow's outcome, not just the feature's existence; and it has a defined path to a more autonomous version of itself, with the roadmap specifying what evidence would justify the product taking the next step automatically.

The gap between the demo environment and the multi-tenant production environment. AI features are almost always demonstrated against a clean internal environment - a well-populated demo tenant, curated sample data, a single account with no customization history. Production is a different environment entirely: thousands of tenants with wildly different data volume and customization, tenant-specific fields and integrations never part of the training data, real data full of formatting inconsistencies and historical artifacts, and strict data isolation requirements where a retrieval step that blends context across tenants is not a quality bug - it is a security incident and, in most enterprise contracts, a breach of contractual obligations. The architectural response mirrors the two-lane pattern used elsewhere, adapted for multi-tenancy: an **experimentation lane** on internal or synthetic tenant data, and a **production lane** connected to live tenant data through the product's existing tenant-isolation boundaries, reinforced further and gated by a formal promotion process before any real customer is exposed.

Part 2: Why Agentic AI Features Fail Differently in a Multi-Tenant Product

Assistive AI features - suggestions, summaries, autofill - fail primarily on relevance and adoption. Agentic AI features, where the product plans and executes actions inside a customer's account with limited human review, fail for an additional, more serious reason: **an action taken incorrectly inside a customer's live SaaS environment is not a UX defect, it is a breach of the operational trust the entire subscription relationship depends on.**

The autonomy spectrum for in-product AI runs from **L0** manual, where the customer performs the workflow themselves, through **L1/L2** assisted and suggested actions the customer reviews and confirms, **L3** conditional autonomy where the product acts inside a bounded scope and the customer can see, undo, or override, **L4** high autonomy handling most instances with exception-based escalation, to **L5** full autonomy running end-to-end with only periodic review. Most initiatives fail because a compelling demo of L4-like behavior gets built and shown to leadership and prospective customers before the L2/L3 scaffolding exists - a bounded, per-tenant action catalog, a customer-visible audit trail, a tested undo mechanism, a track record under real tenant data - and the commercial pressure that follows outpaces the guardrail engineering the feature actually needs.

The missing guardrail layer in a multi-tenant context. An AI feature that can read a customer's data and act inside their account needs a guardrail layer that is easy to skip under launch pressure:

- **Strict tenant-scoped action catalogs**, with every action's scope constrained to the initiating tenant and enforced at the execution layer independently of what the reasoning step intended.
- **Pre-execution validation against the tenant's actual current state**, not just the general case the feature was designed around.
- **Rate limiting and circuit breakers per tenant**, since a runaway agent performing duplicate actions inside one account is a trust incident regardless of how well the feature performs elsewhere.
- **Customer-visible undo and audit trail**, so customers can see what the product did on their behalf and reverse it without contacting support.
- **Human-in-the-loop by default above Tier 1**, with autonomy treated as an ongoing, adjustable customer setting rather than a one-time onboarding checkbox.
- **Deterministic rollback for every autonomous action** - if an action cannot be reliably reversed, it stays a suggested action requiring explicit confirmation.

Without this layer, features either stay permanently in "suggest only" mode and get deprioritized as engagement plateaus, or an autonomous action goes wrong inside a customer's account, generating a churn risk and, in the worst case, a public trust incident that damages adoption of every other AI feature in the product.

Grounding, hallucination, and cross-tenant leakage risk. Beyond ordinary hallucination risk - addressed through live tenant grounding, constrained action generation from a validated catalog, and architectural separation of reasoning from execution - multi-tenant SaaS carries a more severe, distinct risk: **cross-tenant context leakage**, where a retrieval or caching layer inadvertently surfaces one customer's data in a response generated for another, through a shared

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 9, Issue 2, March - April 2026||

DOI:10.15662/IJRPETM.2026.0902022

embedding index that is not properly filtered, a shared prompt cache, or a fine-tuning process that pooled customer data without adequate isolation. Every retrieval call must be scoped by tenant ID at the query level, not filtered after retrieval; any caching layer must be partitioned per tenant with the same rigor as the product's core database isolation; customer data must never be pooled into shared training sets without explicit contractual and technical safeguards; and isolation testing should be an explicit, adversarial part of the test suite, not an assumption inherited from the product's existing tenancy model.

Part 3: The Data and Multi-Tenancy Bottleneck

Why SaaS product data is structurally harder than it looks. Data inside a multi-tenant product has characteristics that make AI infusion materially harder than building AI for a single internal dataset: **extreme variance across tenants**, where a feature that performs well for a data-rich enterprise account can behave unpredictably for a sparsely populated new one; **schema and customization drift**, since most products allow custom fields and workflows per tenant that an AI feature built against the default schema will not match; **freshness requirements that vary by workflow**, where some features need near-real-time data and others tolerate a batch refresh; and **label scarcity**, since ground truth for "was this suggestion correct" typically has to be derived indirectly from whether the customer accepted, edited, or reverted it.

The shared platform argument. Because these challenges recur across every AI feature in the product, they should be solved once at the platform level - a tenant-aware retrieval layer, a governed feature/embedding store with enforced isolation, a schema-normalization layer mapping each tenant's customizations into a common representation, and a labeled interaction corpus built from acceptance/edit/revert signals - rather than rebuilt inside every feature team. This investment does not show a return from any single feature, which is exactly why it tends to be underfunded; the case has to be made explicitly at the leadership level, since every feature built on a shared, tenant-isolated platform reaches production faster with a materially lower risk of a cross-tenant incident than one built from scratch.

Environment parity and MLOps/LLMOps for a multi-tenant product. An AI feature that behaves differently in staging than in production is a customer-trust issue, since customers experience it directly. This requires staging environments with representative tenant diversity, not a single clean demo tenant; CI/CD for prompts, retrieval configuration, and action policies with the rigor of the rest of the product, including regression testing against a replay library of representative tenant interactions; continuous monitoring for data drift, behavior drift, and acceptance-rate trends per tenant segment, since a declining acceptance rate is often the earliest signal of a quality problem; and a retraining cadence tied to product releases, since a schema change or new integration can silently degrade a feature well before a customer complains.

Part 4: Governance That Matches Product and Customer Risk

Risk-tiering AI features inside the product. Generic AI governance is necessary but insufficient for features that act inside a customer's live environment. A tier structure based on what the feature is allowed to do is needed: **Tier 1** read-only insight, no write access; **Tier 2** suggest with mandatory customer confirmation; **Tier 3** bounded autonomous action with visible oversight and undo; **Tier 4** broader autonomous workflow execution, reserved for workflows with a strong Tier 3 track record and customers who have explicitly opted in. Each tier should carry predefined requirements for evaluation rigor, disclosure, default-on versus opt-in status, and rollback readiness - decided once at the governance level, not negotiated per feature.

Embedding Security, Privacy, and Legal as design partners. The most damaging governance failure is security, privacy, or contractual objection surfacing after a feature is built and scheduled, forcing a costly redesign or a rushed compromise. The corrective pattern brings these functions into the design phase of any Tier 2+ feature: Security reviews the tenant-isolation design and action catalog before implementation, with specific attention to cross-tenant leakage risk; Privacy and Legal assess implications for existing contracts, data processing agreements, and regional data-residency requirements early, since these can determine whether a feature can be offered globally or requires a contract amendment; and Legal and Product together determine how AI usage is disclosed in terms of service and in-product messaging.

An AI product governance function. A standing, clearly accountable function spanning Product, Platform Engineering, Security, and Legal maintains the shared platform and tiering standard, reviews every Tier 2+ feature before broader rollout, runs a formal evaluation function testing new features against representative tenant diversity and adversarial cases, and serves as the point where lessons from one feature are captured and applied to the next.

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 9, Issue 2, March - April 2026||

DOI:10.15662/IJRPETM.2026.0902022

Part 5: Change Management for Customers and Internal Teams

Why a technically sound AI feature still fails to adopt. Customers who feel an AI feature is making decisions on their behalf without adequate visibility or control tend to disable it or downgrade their trust in the product overall, even when the feature is accurate - the underlying concern is control and predictability, not correctness. Support and customer success teams unprepared for a new autonomous behavior end up fielding confused customer questions they cannot answer and quietly advise customers to turn the feature off. Questions that need honest answers before a Tier 2+ launch: does the customer understand why the product acted and what data it used; can an administrator audit and adjust the feature's behavior without contacting support; have support teams been trained on failure modes; is rollout paired with in-product education rather than just a release note?

Building customer trust incrementally. Trust in an AI feature that acts inside a customer's account is built the way trust in a new integration partner is built - through a visible, low-stakes track record before broader autonomy is granted:

- **Design-partner and opt-in beta phases as a mandatory stage**, with close monitoring and direct feedback channels, not merely a marketing "early access" label.
- **Default-to-suggest, opt-in-to-autonomous** as the standard rollout pattern - a feature launches at Tier 2 for all customers and only becomes available at Tier 3 after explicit administrator opt-in.
- **Transparent, in-product explanation of every suggestion or action**, so customers can develop calibrated trust rather than blind acceptance or blanket rejection.
- **A visible, low-friction feedback and override mechanism**, where a customer's edit or rejection is captured and used to improve the feature.

Part 6: Fragmented Ownership Across Product, Engineering, and Go-to-Market

AI infusion initiatives touch Product Management, Engineering, the AI/ML platform team, Security, Legal, Customer Success, and often Sales and Marketing, who are eager to announce a capability well before it is operationally ready. Without a single owner, the familiar failure sequence follows: engineering ships a technically working feature, product has not defined what successful adoption means in measurable terms, customer success was not briefed and cannot support it, and marketing has already announced a capability the product organization is still hardening.

A more resilient pattern separates roles explicitly: an **AI Product Owner** for each reimagined workflow, accountable for the adoption and quality outcome and responsible for defining the tier and rollout plan; a **standing cross-functional squad** - product, applied AI/ML engineering, platform engineering, security, and a customer success representative - with genuine decision rights over scope and pacing, not a group merely informed after decisions are made; and a **defined escalation path**, with a named owner and target resolution time, for the recurring trade-offs - speed to market versus evaluation rigor, feature differentiation versus platform consistency, marketing timelines versus governance readiness. The durable fix is treating AI-infused workflows as a permanent capability area with dedicated staffing, not a series of individually staffed feature projects that disband once each feature ships.

Part 7: Choosing the Right First Use Cases to Reimagine

Many SaaS AI roadmaps are set with a mandate to "put AI everywhere," and the first wave selected is either too trivial to change customer behavior or too ambitious for the product's current data and governance maturity. A disciplined approach favors workflows that are **high-frequency and currently manual**, so even a suggest-only version saves meaningful time on something customers do often; **well-bounded**, with a clear, measurable outcome; **data-feasible today**, using data already in the product for most customers; and **reversible**, so any autonomous action can be undone without engineering intervention.

Representative early candidates include drafting or pre-filling structured content the customer currently writes from scratch, surfacing proactive recommendations inside an existing high-traffic screen rather than a new one, and automating a bounded administrative step - tagging, routing, prioritizing - customers currently perform manually. Each has a clear time-saving, a low blast radius if imperfect, and a natural home inside an existing workflow, properties well suited to proving out the autonomy ladder before reimagining higher-stakes workflows. A phased discipline - a Minimum Viable Reimagined Workflow at Tier 2 first, with an evidence-based path to Tier 3 for opted-in tenants - builds the adoption data and trust that later, more ambitious efforts depend on.

Part 8: The Implementation Roadmap

Step 1 - Create an Enterprise Knowledge Base. Assemble product documentation and workflow definitions kept current with actual behavior; common support-case resolutions and known-issue patterns, often the highest-value and

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 9, Issue 2, March - April 2026||

DOI:10.15662/IJRPETM.2026.0902022

least structured asset; a normalized, live representation of each tenant's own data accessed through the product's existing isolation model, never duplicated into a shared pool; schema and customization metadata per tenant; and a labeled interaction corpus built from acceptance, edit, and revert signals. Enforce tenant partitioning as a structural property of the knowledge base, not an access-control layer applied afterward - every piece of tenant-specific content should be structurally incapable of being retrieved for a different tenant, independent of any query-time filter that could be misconfigured. Tag documentation by product area, feature version, and applicable tier; require product and support SME review before publication, with an explicit deprecation workflow for outdated guidance tied to retired features.

Step 2 - Build a RAG Pipeline. Enforce tenant-scoped retrieval as a hard filter applied at the query layer before results are returned, not as a post-processing step - the single most important control against cross-tenant leakage. Use structure-preserving chunking for both documentation and tenant data, so a retrieved chunk remains a coherent unit. Combine semantic search with exact-match filtering on identifiers, record types, and tenant-specific field names, since dense retrieval alone is unreliable for precise identifiers. Ground the pipeline against the tenant's current data, not a stale index, since customer data changes continuously. Re-rank retrieved content and attribute every AI-generated recommendation to the specific record or document it drew on, for both the customer-visible explanation and the internal audit trail. When the pipeline cannot retrieve sufficiently relevant, tenant-specific context, the feature should visibly degrade to a lower-confidence state rather than generating a specific-sounding but ungrounded answer. Evaluate retrieval precision and recall independently of output quality.

Step 3 - Test Models. Build a golden evaluation set from real or representative tenant scenarios with known correct outcomes for offline testing before any live tenant is exposed. Test explicitly against the range of tenant profiles the product has - data-rich and data-sparse, heavily customized and default-configuration, different industries and regions - rather than only a single well-populated demo tenant. Treat cross-tenant isolation testing as a standing, automated adversarial test class, deliberately attempting to surface one tenant's data in another's session, not a one-time manual security review. Run shadow and opt-in beta testing on live tenant data, logging and comparing generated suggestions against actual customer behavior before any customer-facing exposure. Test latency and reliability under production load, since a slow or unreliable AI feature embedded in a core workflow degrades the entire product experience. Require structured review with product, support, and design-partner customers before promotion to a higher autonomy tier or general availability.

Step 4 - Integration and Deployment. Surface the AI capability inside the existing screen and workflow where the task already happens, rather than a separate assistant panel, so the reimagined workflow is discovered naturally. Use the product's feature-flagging infrastructure to enable the capability for design-partner tenants first, then an expanding percentage of accounts segmented by profile, monitoring adoption and error signals at each stage. Apply CI/CD to prompts, retrieval configuration, and action policies alongside the rest of the product codebase. Route autonomous actions through the product's existing permission and audit model, so customers and administrators see a single, consistent record of what happened in their account. Monitor business and trust metrics - acceptance rate, edit rate, revert rate, and support tickets attributable to the feature - alongside latency and error rate, reviewed on a defined cadence as a standing part of operation. Feed every customer edit, rejection, or support escalation back into the Step 1 knowledge base and Step 3 evaluation set, so the feature improves through real usage.

This four-step sequence is iterative in practice: tenant-diversity testing in Step 3 routinely surfaces gaps in the Step 1 knowledge layer's tenant-normalization logic, and production feedback in Step 4 routinely reveals retrieval configuration issues that send the team back to Step 2.

Part 9: A Consolidated Checklist to Prevent Failure

Ownership and business case

1. Assign a named AI Product Owner for each reimagined workflow before development begins.
2. Define the target product metric and baseline it before the initiative starts.
3. Fund the initiative as an ongoing product capability, not a one-time feature project.

Workflow reimagining, not bolt-on design

1. Start from an existing, currently manual, high-frequency workflow rather than "what AI feature can we add."
2. Design the feature to remove or compress existing manual steps, not add a new, parallel surface.
3. Define upfront what evidence would justify moving beyond suggest-only.

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 9, Issue 2, March - April 2026||

DOI:10.15662/IJRPETM.2026.0902022

Architecture and multi-tenant data

1. Build or extend a shared, tenant-isolated retrieval and feature-store platform rather than a bespoke pipeline per feature.
2. Enforce tenant-scoped filtering at the query layer for every retrieval call, not as post-processing.
3. Establish an experimentation lane and a separate production lane, with a documented promotion checklist.
4. Ensure staging environments include representative tenant diversity, not a single clean demo tenant.
5. Implement CI/CD for prompts, retrieval configuration, and action policies with the rigor of the product codebase.
6. Monitor continuously for data drift, feature behavior drift, and acceptance-rate trends per tenant segment.

Knowledge base and RAG pipeline

1. Stand up a governed, tenant-partitioned knowledge base with SME sign-off and version control.
2. Build a RAG pipeline with hybrid retrieval and live grounding against each tenant's current data.
3. Require source attribution and confidence-based graceful degradation on every recommendation.
4. Track retrieval precision and recall as a metric independent of output quality.

Model testing and validation

1. Build a golden evaluation set from real or representative tenant scenarios.
2. Test explicitly against representative tenant diversity, not only a single well-populated account.
3. Treat cross-tenant isolation testing as a standing, automated adversarial test class.
4. Run shadow or opt-in beta testing on live tenant data before general availability.
5. Test latency and reliability under production load.

Agent specific guardrails

1. Define an explicit, tenant-scoped action catalog with declared blast radius, reversibility, and per-tenant rate limits.
2. Require pre-execution validation against the tenant's actual current state for any action above Tier 1.
3. Build deterministic, customer-visible rollback before promotion beyond suggest-only.
4. Separate the reasoning/planning component from the execution component architecturally.
5. Maintain a customer-visible audit trail integrated into the product's existing activity log.

Governance

1. Adopt a risk-tiering framework and apply it consistently across all AI features in the product.
2. Involve Security, Privacy, and Legal as design partners from the start of any Tier 2+ feature.
3. Stand up a standing AI product governance function to review Tier 2+ features before broader rollout.

Change management and adoption

1. Prepare customer success and support teams on feature behavior before general availability.
2. Default new capabilities to suggest-only, with autonomous tiers available only after tenant opt-in.
3. Provide transparent, in-product explanation of every suggestion, and capture customer feedback as a first-class input.

Integration, deployment, and sequencing

1. Surface AI capability inside the existing workflow, integrated into the product's existing permission model.
2. Roll out progressively through feature flags, monitoring adoption and error signals at each stage.
3. Feed every customer correction and support escalation back into the knowledge base and evaluation set.
4. Select initial workflows that are high-frequency, well-bounded, data-feasible, and reversible.
5. Treat AI-infused workflow capability as a permanent, cross-functionally staffed product area.

II. CONCLUSION

Enterprise SaaS products are an unusually demanding environment for AI infusion, not because the underlying AI techniques are harder to apply elsewhere, but because every feature operates inside a multi-tenant trust boundary where a single mistake - a cross-tenant data exposure, an autonomous action a customer did not expect or cannot undo - can damage the commercial relationship immediately and visibly, and because customers judge the AI feature by the same reliability bar they apply to the rest of the product they are paying for.

The organizations that succeed stop treating AI as a feature category to be added to the roadmap and start treating it as a lens for reimagining specific, currently manual workflows - built on a shared, rigorously tenant-isolated data and

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 9, Issue 2, March - April 2026||

DOI:10.15662/IJRPETM.2026.0902022

retrieval platform, governed by an explicit risk-tiering and guardrail architecture rather than an all-or-nothing leap to autonomous action, and rolled out with as much deliberate investment in customer trust and internal enablement as in the underlying model or pipeline.

None of this makes AI infusion into a SaaS product simple. What it does is convert the risk of a damaging, trust-eroding failure into a set of known, manageable design and governance decisions - made deliberately, at the right point in the workflow's reimagining, by the people who are accountable for the product customers rely on.

REFERENCES

1. National Institute of Standards and Technology (NIST). *AI Risk Management Framework (AI RMF 1.0)*. U.S. Department of Commerce, 2023.
2. International Organization for Standardization. *ISO/IEC 42001:2023 - Information Technology, Artificial Intelligence Management System*. ISO, 2023.
3. Lewis, P., Perez, E., Piktus, A., et al. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks." *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.
4. Cloud Security Alliance. *Security Guidance for Critical Areas of Focus in Cloud Computing* (multi-tenancy and data isolation guidance). CSA, 2017.
5. Sculley, D., Holt, G., Golovin, D., et al. "Hidden Technical Debt in Machine Learning Systems." *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.
6. OWASP Foundation. *OWASP Top 10 for Large Language Model Applications*. OWASP, 2023.

BIOGRAPHY

Viswanathan Tiruchindurai Srinivasan is an Associate Vice President in the Technology Consulting practice of Persistent Systems, where he partners with CIOs and C-suites to turn business problems into opportunities using AI, from pilots into P&L impact. He has led AI programs across FinTech, SaaS, and Data platforms, focusing on strategy, architecture, and operating models that scale safely.

LinkedIn: https://www.linkedin.com/in/viswanathants/