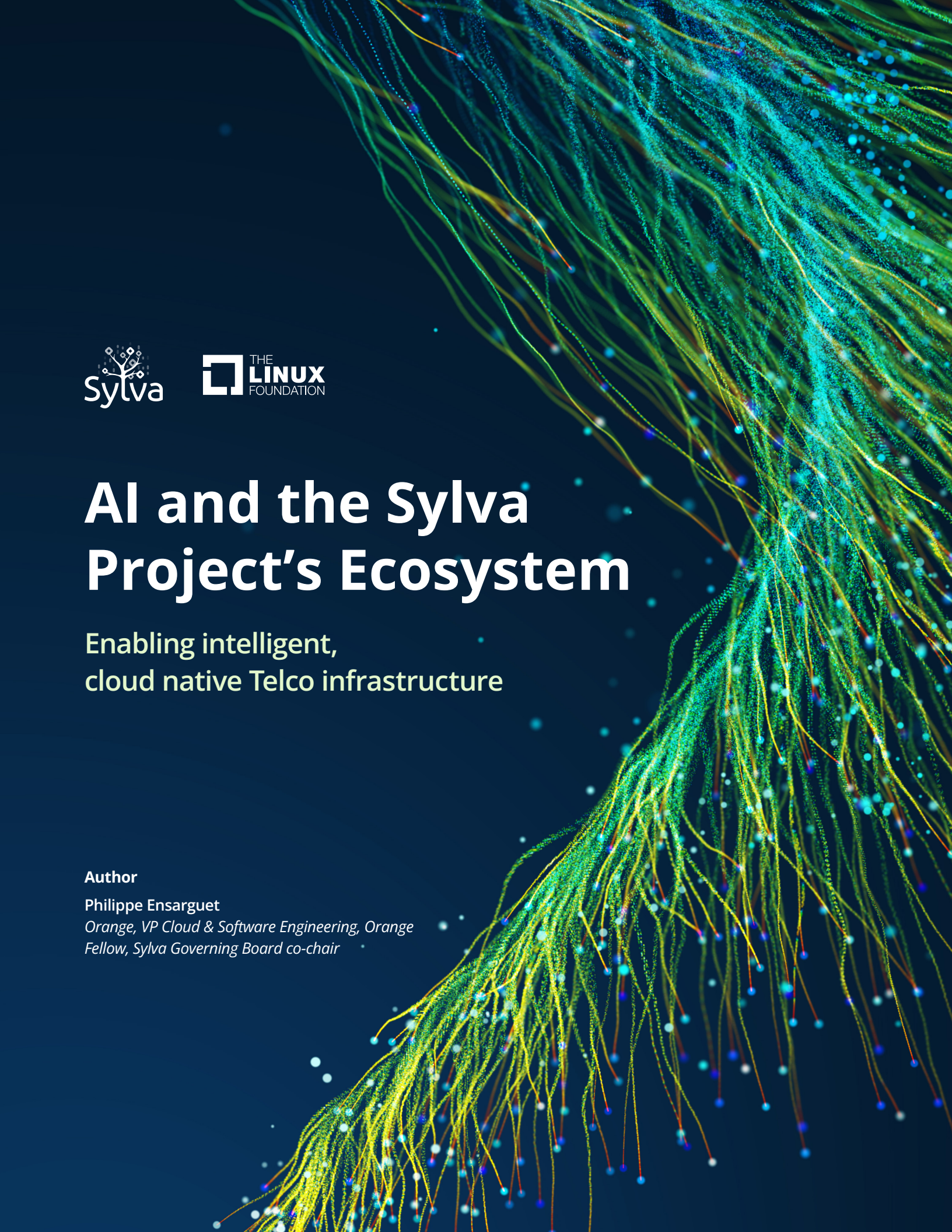


AI and the Sylva Project's Ecosystem

Enabling intelligent, cloud native Telco infrastructure

Author

Philippe Ensarguet
Orange, VP Cloud & Software Engineering, Orange Fellow, Sylva Governing Board co-chair

Table of Contents

Foreword

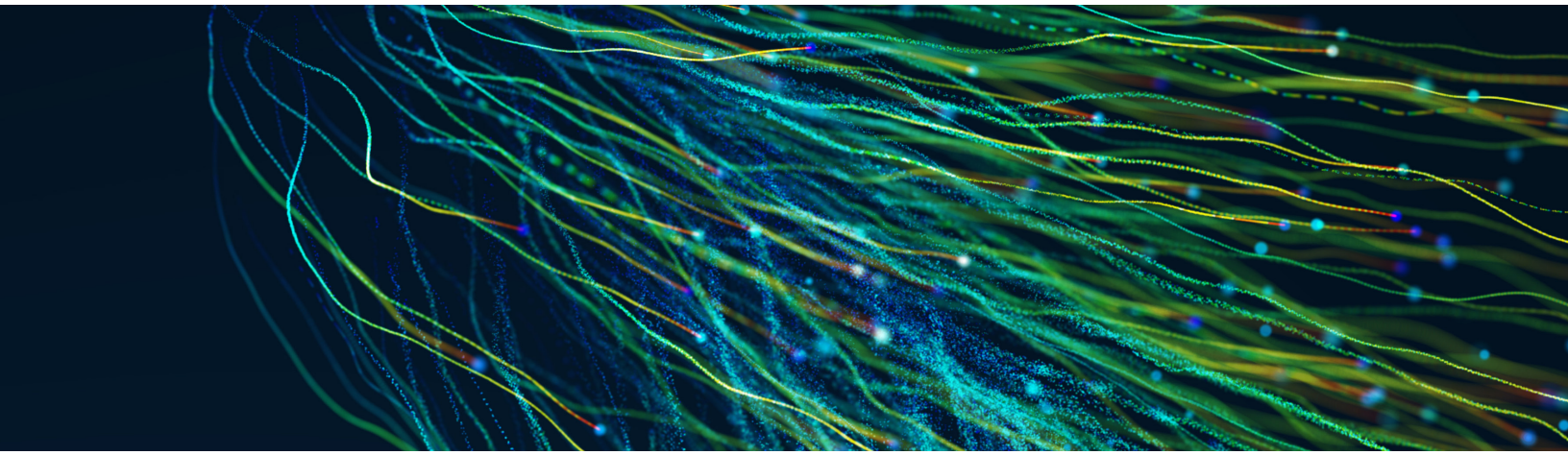
As artificial intelligence reshapes industries, telecommunications stands at a pivotal inflection point. The promise of AI in telecom is significant: more intelligent operations, more adaptive infrastructure, and ultimately, more autonomous networks. But realizing that promise at scale requires more than powerful models. It requires an open, cloud native foundation that can support AI consistently, securely, and across diverse environments.

That is what makes this paper so timely. It makes a compelling case that the future of AI in telecom depends on the maturity of the underlying platform, and that Project Sylva provides an important part of that foundation. By bringing together cloud native infrastructure, observability, automation, and open collaboration, Sylva helps create the conditions for AI to move beyond isolated experiments and into production-ready, operationally meaningful deployments.

Just as importantly, this paper highlights that the next phase of innovation will not be driven by any one organization alone. Advancing from machine learning to agentic AI in telecom will require open standards, shared governance, trusted operational frameworks, and deep ecosystem collaboration. These are areas where open source has consistently proven its value, and where the Linux Foundation is proud to help convene communities that can build the critical infrastructure for the future.

I'm grateful to the Sylva community and to Philippe Ensarguet for helping to articulate both the opportunity and the work ahead. This white paper is an important contribution to a growing conversation about how we build AI-ready infrastructure in the open—and how open collaboration can help ensure that the next generation of telecom networks is intelligent, interoperable, and trusted by design.

— **Mark Collier**
General Manager, AI & Infrastructure
The Linux Foundation



Executive summary

Telecommunications networks are shifting from manually operated, hardware-centric systems toward autonomous, software-defined platforms. This transformation is driven by two converging forces:

1. The **maturation of cloud native telco infrastructure** through open source initiatives like Project Sylva,
2. and the **rapid evolution of artificial intelligence,** from traditional machine learning through to agentic AI systems capable of autonomous reasoning and action.

This white paper argues that AI in telecom will remain fragmented and unable to scale beyond siloed proofs of concept without a standardised, open, cloud native substrate.

It's hard to shoot for the moon if you don't have a ground station!

The Sylva project[1] provides that substrate. Conversely, Sylva's full potential, particularly for closed-loop automation and autonomous operations, can only be realised when AI is deeply integrated across every layer of the stack.

Key arguments:

- Sylva's architecture is uniquely suited to host the full AI lifecycle, from data ingestion and model training to distributed inference at the edge, because of its Kubernetes-native design, built-in observability, and GitOps-driven automation.
- The emergence of agentic AI, autonomous systems that reason, plan, and execute multi-step operations, represents a step change beyond traditional ML-based automation, and Sylva's programmable infrastructure is a natural execution environment for these agents.
- To realise this vision, we must address the challenges of data governance, model lifecycle management, trust and cross-layer integration. These issues can be best solved through open source collaboration.

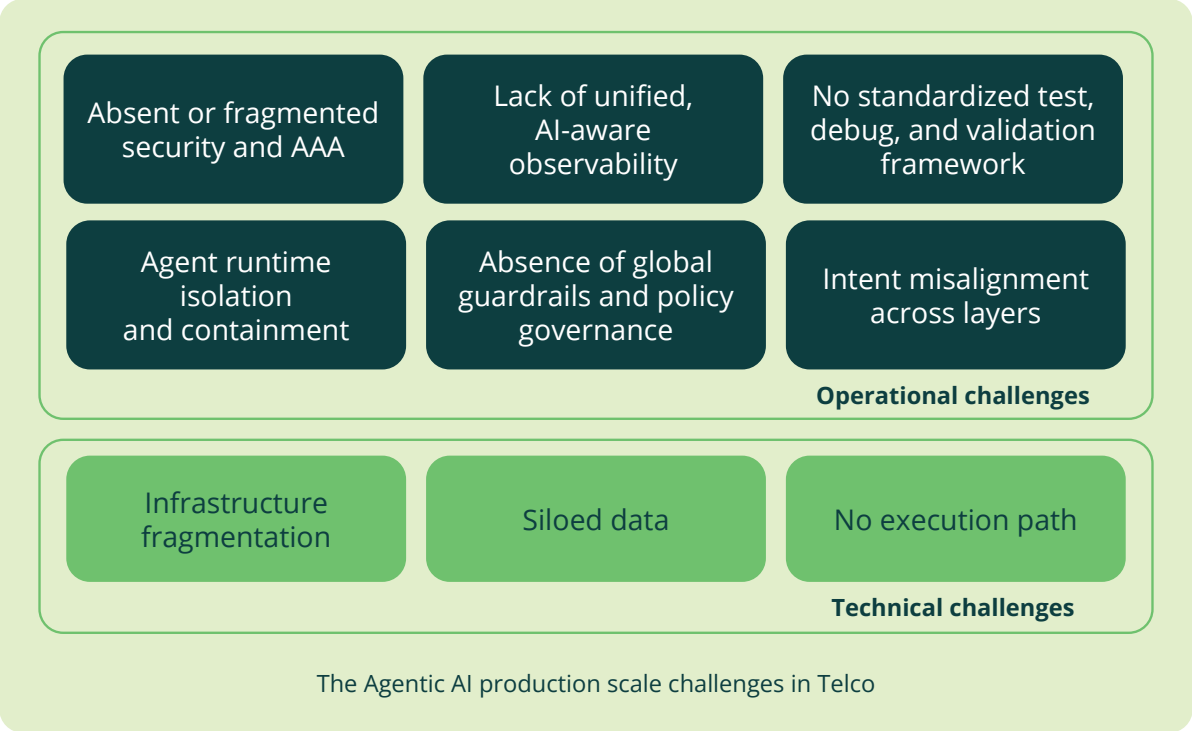
1. https://sylvaproject.org

1. Introduction: The thesis

1.1 AI in Telecom: easy on exploration mode, hard to grade to production

Telecom operators have invested heavily in AI over the past five years. Yet the industry-wide pattern is consistent: explorations and pilot projects succeed in isolated environments but fail to scale into production, often.

The reasons are structural and extend well beyond the commonly cited infrastructure gaps.



The Agentic AI production scale challenges in Telco

A. Technical substrate gaps

- **Infrastructure fragmentation:** AI workloads are deployed on bespoke platforms that differ across operators, vendors, and network domains (RAN, Core, Transport, Edge). There is no common runtime or deployment model.
- **Siloed data:** Network telemetry, performance counters, feature traces and operational logs are scattered across proprietary systems with incompatible schemas and access models. AI models trained on partial data produce partial insights.
- **No execution path:** Even when AI produces a valid recommendation, there could be no programmatic way to execute that recommendation back into the network. The “last mile” of automation is highly sensitive to the level of automation and to the management of network functions by software. This is compounded by a prerequisite that is often overlooked: network functions must themselves be truly cloud-native, exposing declarative APIs, supporting dynamic lifecycle management, and running as stateless, containerized workloads, for AI to programmatically act on them. Legacy or “lift-and-shift” virtualized NFs remain opaque to automation, regardless of how capable the AI system is.

B. Missing enterprise-grade operational fabric

Beyond infrastructure, AI in telecoms needs to operate within the corporate-wide operational and governance frameworks that any production system requires, and these are not the primary requirements in the exploration phase. These missing components are often the real blockers to scaling:

- **Absent or fragmented security and AAA (Authentication, Authorisation, Accounting):** AI systems, and especially autonomous agents, may interact with critical network infrastructure through multiple interfaces: direct API calls, but increasingly also through MCP servers that expose network capabilities as agent-callable tools, effectively opening a new door to the same underlying infrastructure. In exploration mode, security is typically handled ad hoc: hardcoded credentials, overly broad API permissions, no fine-grained authorization model, and MCP servers running without authentication. Production deployment requires a corporate-wide AAA framework that governs what each AI system is permitted to observe, decide, and execute, regardless of whether it reaches the network through an API call or an MCP tool invocation. This includes identity management for AI agents (not just human operators), role-based and attribute-based access control (RBAC/ABAC) for API calls and configuration changes, mutual TLS and token-based authentication between AI components and network functions, and comprehensive accounting, logging every AI action against a verifiable identity for compliance and forensic purposes. A key architectural choice that could mitigate risk is to route AI-driven changes through GitOps workflows rather than direct API calls or MCP requests: an agent commits a declarative change to a Git repository, making every action reproducible, version-controlled, reviewable by a human operator before it reaches the live network, and reversible by design. Without this, no operator's security team will approve production deployment, regardless of how well the AI model performs.
- **Lack of unified, AI-aware observability:** In exploration mode, you may typically monitor the AI model's outputs but not the full operational chain. Production AI requires end-to-end observability that spans the AI system itself, and covering the model performance, inference latency, or drift indicators, the data pipeline feeding the AI, including data freshness, schema conformance, or completeness, the actions taken by AI on the network, like "what was changed", "when", and "with what result", and the business-level impact, mainly around SLA metrics and customer experience indicators. Most operators have observability tools for their network infrastructure, but these are disconnected from AI-specific monitoring. There is no unified pane of glass that allows an operations team to trace from "the AI agent made a decision" through "this configuration change was applied" to "this was the impact on service quality". And it's a vital asset when you understand that the true shift with Agentic AI is related to autonomy.
- **No standardised test, debug, and validation framework:** AI models in telecom interact with complex, stateful, safety-critical systems. Yet the exploration phase typically may lack the rigorous testing frameworks expected of production network software. What is missing includes staging environments where AI decisions can be validated against real traffic patterns without impacting the live network, simulation and digital twin capabilities that allow agents to be tested against realistic failure scenarios before production deployment, regression testing for AI models, ensuring that a retrained model does not introduce regressions on previously handled scenarios, and debug tooling that allows operators to replay an agent's reasoning chain, inspect intermediate tool calls, and understand

why a particular decision was made. Without these, operators cannot satisfy the change management and validation processes that govern their production networks.

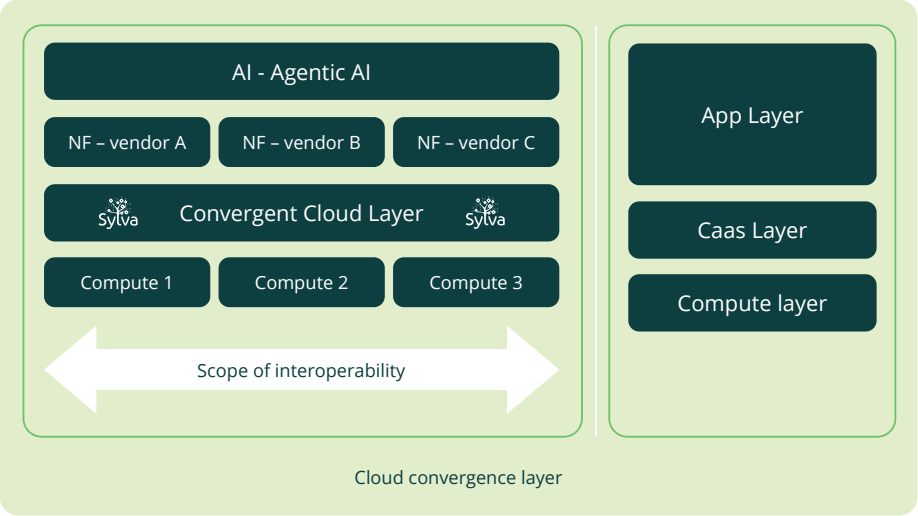
- **Agent runtime isolation and containment:** As you are on a fast track path, exploration steps typically run with implicit access to the full environment. In production, AI agents, particularly agentic systems that can reason and act autonomously, require strict containment at two distinct levels. At the **runtime level**, the agent's code and processes must be isolated, resource quotas, rate limiting, and secure multi-tenancy ensure that a malfunctioning or compromised agent cannot consume excessive resources or cascade failures beyond its designated scope. But runtime isolation alone is not sufficient. At the **protocol level**, policy-supervised execution boundaries must govern what each agent is permitted to reach through its integration interfaces (MCP, A2A): which network domains, which APIs, which data sources, and which actions are authorized, enforced not locally by the agent itself, but by centralized policies managed with a global view at upper orchestration layers. This separation matters: the agent's code may run in a container, but its access to critical network infrastructure is governed by protocol-level policies that no individual agent can bypass or renegotiate.
- **Absence of global guardrails and policy governance:** Individual AI models may have local safety checks, but production telecom AI requires a corporate-wide guardrail framework that operates above any single model or agent. This includes global constraints that no AI system can override (like maximum percentage of network resources modifiable in a single action window, mandatory redundancy preservation rules, blackout windows during which automated changes are prohibited), a policy engine that evaluates every AI-proposed action against operator-defined business rules, regulatory requirements, and safety boundaries before execution is permitted, cross-agent coordination policies that prevent multiple AI systems from issuing conflicting or compounding actions on the same network segment simultaneously, and escalation policies that define, at a corporate level, which categories of AI decisions require human approval, not left to the discretion of individual AI system developers.
- **Intent misalignment across layers:** An often-overlooked challenge is that AI agents in telecom operate under multiple, potentially conflicting layers of intent. A user intent (e.g., "optimize throughput for this customer") may conflict with an organizational intent (e.g., energy reduction policies or fair-use constraints), which may itself diverge from the developer's original intent for the agent's scope of action. In a telecom environment, these layers multiply: the network planning team, the operations team, the security team, and the business team each express intents that an agent must reconcile, and when they conflict, the agent must know which takes precedence. Without an explicit intent hierarchy and conflict resolution model, agents can deliver technically correct results that violate organizational policies, or conversely, refuse legitimate requests because of overly rigid constraints. This is not a theoretical risk; it is an operational reality that becomes acute the moment multiple agents from different vendors or teams operate on shared infrastructure, each carrying its own embedded assumptions about priorities.

1.2 The Sylva project as the AI-ready substrate

The Sylva project addresses these structural barriers by providing a **standardised, open, cloud native infrastructure stack** that unifies the runtime environment for telecom workloads, including AI workloads. When AI systems are envisaged to be built on top of Sylva:

- Models can be trained and served on the same Kubernetes clusters that run network functions.
- Observability telemetry feeds directly into AI pipelines without proprietary connectors.
- GitOps workflows provide the programmable execution layer that closes the loop between AI decisions and network configuration changes.
- Edge clusters managed by Sylva may bring inference closer to the data source, enabling real-time AI at the RAN and far edge.

This paper examines the architecture, capabilities, and practical use cases of AI integrated with Sylva, with particular focus on the transformative potential of agentic AI for autonomous network operations.



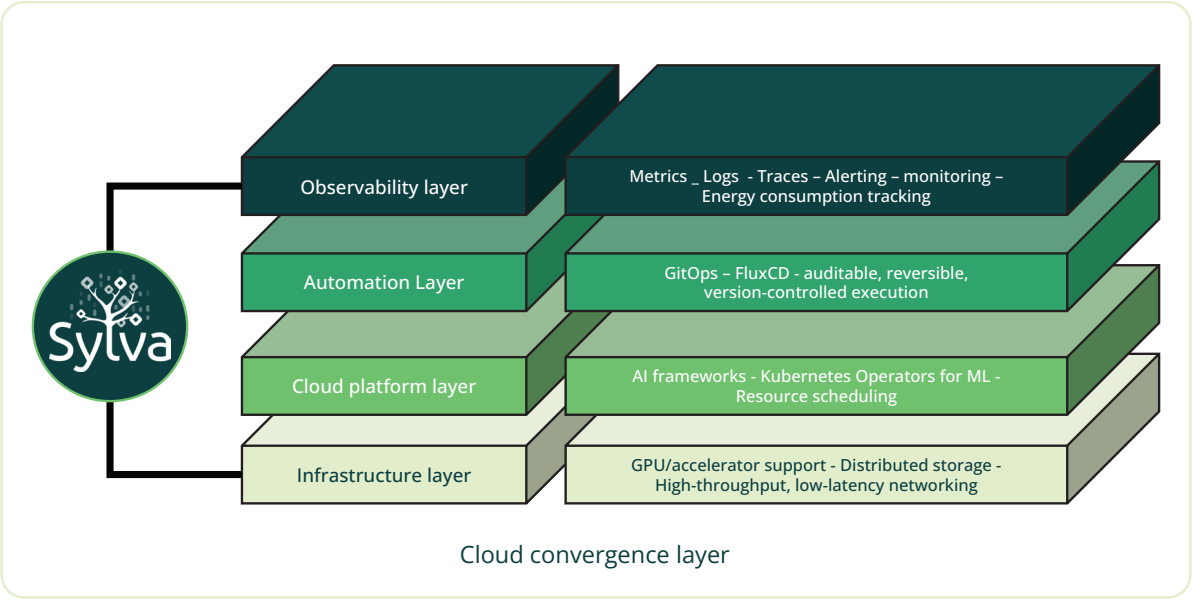
About Project Sylva

Launched in November 2022 and hosted by Linux Foundation Europe, the Sylva project was founded by leading European carriers (Orange, Deutsche Telekom, Telefónica, TIM, and Vodafone) together with network function providers Ericsson and Nokia. The project delivers three complementary pillars: a cloud software framework that defines common requirements and architecture for telco cloud infrastructure; a reference implementation providing a production-grade, open-source telco cloud stack built on Kubernetes, with GitOps-based lifecycle management; and a validation program with dedicated Validation Centers where workloads (e.g.: network functions) are tested and certified for compatibility with the Sylva stack. Three years after launch, the community has grown to 35 contributing companies and 16 member organisations, with 22 validated workloads and real deployments running in live operator networks across 10 countries. Sylva is now turning toward its next set of challenges, including edge federation, promoting cloud sovereignty, AI-infused operations, Open RAN and 6G-ready architectures.

2. Project Sylva: Architecture through an AI lens

This section examines the Sylva project's architecture not as a generic infrastructure overview, but specifically through the lens of what each component enables for AI workloads and agentic operations.

We walk through the four layers of the stack, infrastructure, cloud platform, automation, and observability, mapping each to its role in the AI lifecycle: from the compute resources that power model training and inference, through the Kubernetes runtime that hosts AI frameworks alongside network functions, to the GitOps execution path that makes AI decisions actionable, and the telemetry plane that feeds AI systems with the data they need to reason and learn.



2.1 Infrastructure layer: Compute for the full AI lifecycle

Sylva's infrastructure layer provisions compute, storage, and networking across centralised data centers, regional sites, and user-edge nodes. For AI, this means:

- **GPU/accelerator support** via Kubernetes device plugins and the NVIDIA GPU Operator, enabling model training in central clusters and inference at the edge. Sylva's architecture is not tied to a single hardware vendor, the project is open to contributions that extend accelerator support to other hardware platforms, ensuring operators retain choice as the AI hardware landscape evolves.
- **Distributed storage** capable of handling the datasets required for model training, network telemetry, traffic captures, and performance counters, with data locality awareness. Intensive AI training requires high-bandwidth GPU-to-storage links (such as GPUDirect Storage) to avoid data ingestion bottlenecks. Sylva's current storage capabilities are well suited for edge AI inference and moderate training workloads. However, large-scale training would require specialized storage fabrics that are not currently integrated or validated within the Sylva stack. As AI workloads on telecom infrastructure grow in ambition, this is an area where the community should evaluate and extend Sylva's infrastructure layer.

- **High-throughput, low-latency networking** (SR-IOV, DPDK, Multus CNI) is required for both AI inference serving and the real-time data ingestion pipelines that feed AI models, and supported since the beginning in Sylva. At scale, intensive AI training also demands GPU-to-GPU interconnects (such as GPUDirect RDMA) for efficient distributed training across nodes, another area where Sylva's networking layer would need to be extended for heavy AI workloads.

2.2 Cloud platform layer: Kubernetes as the AI runtime

Kubernetes is the de facto orchestration layer for modern AI/ML workloads. Within Sylva, we have the opportunity to expose:

- **AI frameworks** (PyTorch, TensorFlow, Langchain, ...) could run as containerised workloads alongside network functions (CNFs) on shared clusters, enabling resource efficiency.
- **Kubernetes Operators** for ML (Kubeflow Training Operator, KServe,...) manage the lifecycle of training jobs and inference endpoints.
- **Resource scheduling** with GPU affinity, topology-aware scheduling, and priority classes ensures AI workloads meet performance requirements without starving network functions.

2.3 Automation Layer: GitOps as the AI execution path

This is a critical and underappreciated link. Sylva's GitOps-based automation layer, based on FluxCD, provides the **programmable execution path** that allows AI decisions to be translated into infrastructure actions:

- An AI system that determines a configuration change (e.g., scaling a network slice, adjusting a RAN parameter) can express that change as a Git commit to a declarative configuration repository.
- The GitOps controller detects the change and reconciles the desired state with the running infrastructure.
- This creates an **auditable, reversible, version-controlled** execution path for AI-driven actions, essential for trust and compliance.
- Model versions themselves can be managed through GitOps workflows, enabling automated rollback if a new model degrades performance.

2.4 Observability layer: The AI data plane

Sylva's observability stack is not just a monitoring system; it is the primary data source for AI in the telecom stack. The current stack comprises:

- **Prometheus** exposes thousands of time-series metrics from every Kubernetes node, pod, and network function. These metrics are the raw material for anomaly detection, capacity forecasting, and performance optimization models. Thanos aggregates metrics from Prometheus instances across multiple clusters into the management cluster, providing AI systems with a unified, cross-cluster view of the infrastructure.

- **Loki**, fed by **Fluentd** and **Fluent Bit collectors**, centralises logs from all clusters into the management cluster. AI systems can query Loki's log streams for pattern detection, error classification, and correlation with metric-based anomalies.
- **Alertmanager** provides rule-based alerting with deduplication and grouping, which can serve as a trigger mechanism for AI-driven investigation and remediation workflows.
- **Kepler** (Kubernetes Efficient Power Level Exporter) exposes energy consumption and carbon metrics, a unique data source enabling AI-driven energy optimisation, a critical concern for telecom operators managing large distributed infrastructure.
- **Grafana** dashboards can be augmented with AI-generated annotations, predictions, and alerts, creating a human-in-the-loop interface for AI-driven operations.

The key architectural point: because these observability tools are part of the Sylva standard, AI models built for one Sylva deployment can be portable to another; the data schema and access patterns are consistent.

Note on distributed tracing: The current Sylva observability stack provides strong capabilities for metrics and logs, but does not yet include native support for OpenTelemetry distributed tracing. Agentic AI systems that need to correlate events across network domains require the kind of request-level, cross-service correlation that distributed tracing provides. OpenTelemetry, the CNCF standard for telemetry collection (metrics, logs, and traces), would be a natural evolution for Sylva's observability layer and a key enabler for advanced AI operations. It will be a major item of the Call To Action!

3. From ML to Agentic AI: The evolution of network intelligence

3.1 Three generations of AI in Telecom

The application of AI in telecommunications has evolved through three distinct phases, each building on, rather than replacing, the previous:

Generation 1- Rule-Based Automation (Pre-2018): Traditional network automation relied on predefined rules and thresholds. If CPU utilisation exceeds 80%, scale up. If packet loss exceeds 1%, reroute. These systems were rigid, reactive, and unable to handle the combinatorial complexity of modern networks.

Generation 2 - Machine Learning–Based Operations (2018–2024): The introduction of ML models enabled networks to move from reactive to predictive operations. Key capabilities included:

- **Time-series forecasting** for traffic prediction and capacity planning.
- **Anomaly detection** (autoencoders, isolation forests) on network telemetry to identify performance degradation before user impact.
- **Reinforcement learning** for RAN parameter optimisation, learning optimal handover thresholds, power levels, and scheduling policies through interaction with the live network.
- **Graph neural networks (GNNs)** for topology-aware analysis, understanding how failures propagate through interconnected network elements.

These models improved operations significantly but suffered from key limitations: they were narrow, one model per task, brittle, required retraining when conditions changed, and disconnected; they produced recommendations but could not act on them autonomously.

Generation 3 - Agentic AI (2025–present): The emergence of large language models (LLMs) and agentic AI frameworks has fundamentally changed the landscape. An AI agent is a system that can:

- **Reason** about complex, multi-domain problems using natural language understanding and structured logic.
- **Plan** multi-step sequences of actions to achieve a goal.
- **Use tools**, invoking APIs, querying databases, reading documentation, and calling specialised ML models.
- **Act** on the environment, executing configuration changes, triggering workflows, and escalating to human operators when uncertainty is high.
- **Learn and adapt** through feedback from the outcomes of its actions.

This is a qualitative leap beyond Generation 2. Where a traditional ML model might detect an anomaly and output a probability score, an agentic system can detect the anomaly, correlate it with recent configuration changes, hypothesise a root cause, consult runbook documentation,

determine a remediation plan, validate that plan against safety constraints, execute the fix through a GitOps commit, and verify the resolution, all autonomously.

These three generations coexist, they do not replace each other. Rule-based automation remains the right choice for well-understood, deterministic actions where predictability is paramount (threshold-based scaling, compliance checks). **ML-based models** excel at narrow, well-defined tasks where their deterministic, statistically grounded outputs outperform the probabilistic reasoning of an LLM, time-series forecasting, anomaly detection, and RL-based RAN optimization are not obsolete; they are mature and battle-tested. **Agentic AI** adds value where the task requires multi-domain reasoning, dynamic planning, and autonomous action across systems, precisely the scenarios where pre-coded workflows and single-purpose models fall short.

The art of designing AI-native telecom operations lies in choosing the right approach for each task: using an LLM-based agent where a simple ML model delivers equivalent results with deterministic behavior and lower cost is as much an anti-pattern as using a static rule where intelligent reasoning is genuinely needed.

The use cases in **Section 4** illustrate this coexistence: within a single scenario, an agentic orchestrator may delegate to RL-based xApps for near-real-time RAN optimization, invoke a lightweight autoencoder for anomaly detection, and reserve its own reasoning capabilities for cross-domain correlation and novel plan generation.

3.2 Sylva Agentic AI: Architecture and integration

Deploying agentic AI within a Sylva environment requires new specific architectural components, let's call it **Sylva Agentic AI** complementing **Sylva Core**, that address four distinct questions:

1. **where** does the agent run (**runtime environment**),
2. **how** does it interact with network tools and data sources (**tool integration via MCP**),
3. **which** model serves its reasoning for each task (**intelligent inference routing**),
4. and **what prevents** it from causing harm (**guardrails and safety**).

Let's examine each in turn.

Agent runtime environment

Agents are long-running, stateful processes that maintain context across interactions. **Containers** on Kubernetes are the non-negotiable control plane for AI agents: that's where orchestration, lifecycle management, and the broader agentic workflow lives, backed by the full CNCF ecosystem of service mesh, observability, and GitOps. For the "brain" of an agent, reasoning, LLM inference, decision logic, containers on Sylva's Kubernetes clusters are the proven path. But at the edge, **WASM** is the game-changer. When a micro-agent needs to make a decision at a cell site in sub-millisecond latency, dynamic spectrum allocation, real-time anomaly detection, a full container cold-start is too slow. WebAssembly offers near-native speed, a sandboxed security model by design, and binaries a fraction of the size, making it the natural "last mile" runtime for lightweight, security-sensitive inference at the far edge while also being

able to execute independently of the underlying CPU architecture. **VMs** still have a role, though it's narrowing: hard multi-tenancy isolation for sovereign AI workloads, GPU partitioning for training, or regulatory requirements demanding strong workload separation. **MicroVMs** may offer a middle path, combining VM-grade isolation with near-container startup times.

The architectural challenge for the Sylva community with Agentic AI for Telco is not choosing one runtime over another, but enabling the platform to schedule an AI agent across all three seamlessly, selecting the appropriate runtime based on deployment context, security requirements, and latency constraints.

Within Sylva, agents have access to:

- The Prometheus/OpenTelemetry data plane, for situational awareness
- The GitOps automation layer, for action execution
- Specialized ML models running as KServe inference endpoints, for domain-specific analysis
- Network function APIs and configuration interfaces, for direct interaction with the managed network

Tool integration layer

Agentic systems derive their power from tool use. The way agents discover and invoke tools is an area of rapid evolution, MCP, skills definitions, CLI wrappers, and other integration patterns are competing and coexisting, and what becomes the dominant standard may shift. **What matters for Sylva is that its architecture exposes the right capabilities** through well-defined, machine-consumable interfaces; the specific protocol through which agents consume them can evolve. That said, MCP has emerged as a strong candidate today, and the rest of this section focuses on it.

However, without a standardized integration model, each agent-to-tool connection becomes a bespoke implementation, recreating the fragmentation problem that Sylva was designed to solve at the infrastructure level. This is where the Model Context Protocol (MCP) becomes critical.

MCP is an open protocol, created by Anthropic and donated to the **Agentic AI Foundation** under the **Linux Foundation** in December 2025. It provides a universal, vendor-neutral standard for connecting AI agents to external tools and data sources, often described as "USB-C for AI." MCP follows a client-server architecture: the AI agent acts as an MCP client, while each tool or data source is exposed through an MCP server that declares its available capabilities (tools, resources, queries). Any MCP-compatible agent can discover and invoke any MCP server, eliminating the need for custom integrations per agent-tool pair.

The protocol has seen rapid industry adoption, with support from OpenAI, Google, Microsoft, and thousands of community-built servers. In a Sylva context, MCP provides the standardised integration layer between agentic AI systems and the Sylva stack's components.

The value of MCP in this architecture is threefold. First, **portability**: an AI agent built to operate on one Sylva deployment can work on any other, because the tool interfaces are standardised.

Second, **composability**: operators can add new tools, a new network domain, a new ML model, a new data source, by deploying an MCP server, without modifying the agent itself. Third, **governance**: because MCP servers explicitly declare their capabilities and can enforce authentication and authorisation, they provide a natural control point for the security and AAA requirements described in **Section 1.1**. Each MCP server can enforce fine-grained access policies on what an agent is permitted to observe and act upon.

Given that both MCP and Sylva are now hosted under the Linux Foundation, there is a natural opportunity for the Sylva community to develop and maintain a set of **reference MCP servers** for the Sylva stack, ensuring that the agent-to-infrastructure integration layer is as standardised and open as the infrastructure itself.

An early Sylva MCP implementation has been released in the Sylva Project, opening the door to agentic AI operation of the Kubernetes clusters. This first MCP server exposes read-oriented tools focused on cluster lifecycle observability, enabling an AI agent to discover and inspect the state of the Sylva-managed infrastructure. Currently exposed capabilities include: listing all Sylva clusters (management, workload, and generic clusters deployed through the GitOps workflow or the SylvaWorkloadCluster operator), querying the deployment status of individual clusters, inspecting the status of SylvaUnitsRelease resources (the Helm-based deployment units that compose each cluster), retrieving team configurations (the organizational construct that governs GitOps-based workload cluster provisioning, including namespace isolation, Git repository bindings, Vault secret storage, and RBAC policies), and reading the status of individual units (FluxCD Kustomizations or HelmReleases) running within a cluster namespace. While this initial scope is deliberately read-only, a sensible starting point that allows agents to observe and reason about infrastructure state without risk of unintended modifications, it already demonstrates the core pattern: Sylva's internal domain concepts (clusters, teams, units, releases) are semantically described and discoverable by any MCP-compatible agent. More can be found on the community add-ons[2] of the Sylva project.

2. https://gitlab.com/sylva-projects/community-addons/mcp-sylva

Skills: declarative behavior for AI agents

Traditional automation relies on workflows, imperative sequences of steps where every action, branch, and fallback is pre-coded. Agentic AI shifts this paradigm: instead of scripting how an agent should behave, operators define what the agent should achieve and which boundaries it must respect. This is what a skill provides.

A skill is a declarative specification that defines a specific agent behavior: the goal to accomplish, the tools the agent is authorized to use, the constraints and policies that apply, and the expected inputs and outputs. The agent receives the skill definition and autonomously reasons about how to fulfil it, selecting tools, planning steps, and adapting to context, rather than executing a pre-scripted sequence.

For a Sylva environment, the skill pattern is a natural extension of the intent-driven philosophy already advocated in this paper. Where MCP servers expose what tools are available, skills describe what the agent should do with them in a given context. A "diagnose UPF latency" skill would declare the goal (identify root cause of latency degradation), the authorized tools (Prometheus MCP, Loki MCP, Kubernetes API MCP), the constraints (read-only access, escalate if confidence is below threshold), and the expected output (structured root-cause report). The agent reasons about how to achieve the goal; it is not told the steps.

This opens a powerful community dimension: a shared library of Sylva-validated, telco-specific skills, declarative behavior specifications for common operational tasks (incident triage, slice provisioning, RAN conflict resolution, security containment), that any operator can deploy and any MCP-compatible agent can consume. Skills become the reusable domain expertise of the ecosystem, versioned through GitOps and tested in Validation Centers alongside network functions and MCP servers.

Intelligent inference routing

Agentic AI systems don't rely on a single model, they orchestrate multiple models, each suited to a different task: a lightweight SLM for simple classification, a reasoning model for complex root-cause analysis, a specialized fine-tuned model for RAN parameter optimization. Without intelligent routing, every query gets sent to a default model regardless of complexity, wasting expensive compute on trivial tasks and potentially violating data sovereignty constraints when traffic is routed to hosted endpoints outside the operator's jurisdiction.

Production agentic AI on Sylva may require an **inference gateway**[3], a Kubernetes-native policy enforcement point that dynamically routes agent queries to the most appropriate model backend based on real-time signals: cost, accuracy, latency, quota, and geo-location constraints. Emerging open-source projects such as AgentGateway (hosted under the Linux Foundation) demonstrate this pattern, using Kubernetes CRDs to declare routing policies and an agentic

3. https://gateway-api-inference-extension.sigs.k8s.io/

control plane that continuously adapts model selection. The gateway sits between the agent and its LLM backends, both local models (vLLM, Ollama) running on Sylva clusters and external hosted endpoints, providing a single control point for cost visibility, data residency enforcement, and adaptive model selection.

For the Sylva ecosystem, this means the community should evaluate and integrate an inference routing layer as part of the AI platform stack, ensuring that the question of which model serves each agent decision is governed with the same rigor as which tool the agent invokes, via MCP, and which action it executes, via GitOps. Note that the inference gateway and MCP serve different but complementary roles: MCP allows an agent to explicitly invoke a specific ML model as a tool (e.g., "call the anomaly detection model on these metrics"), while the inference gateway governs the routing of the agent's own reasoning, deciding which LLM backend handles a given prompt based on cost, complexity, and constraints. One governs tool-level model access; the other governs the agent's brain.

Cost visibility must come before cost optimization: you can't optimize what you can't measure.

Guardrails and safety

The three components above, runtime, tool integration, and inference routing, give an agent the ability to run, interact, and reason. But ability without boundaries is dangerous on a production telecom network. An agent that can commit GitOps changes, invoke network configuration APIs, and select its own model has the potential to cause significant harm if its actions are unconstrained. The guardrails layer is what makes the difference between a capable agent and a trustworthy one.

Autonomous action in a production telecom network demands robust safety mechanisms:

- **Action classification:** Actions are classified through two complementary mechanisms: a deterministic tier based on blast radius (scope, reversibility, affected domains) that serves as a safety baseline no agent can override, and a confidence-driven tier where the agent scores its own certainty for each decision. A low-confidence action gets escalated even if its blast radius is small; a high-confidence, low-scope action can be auto-executed. This interplay between hard rules and dynamic confidence is what enables the graduated autonomy described later on in **Section 5.3**.
- **Human-in-the-loop escalation:** High-impact actions require human approval before execution. The agent presents its reasoning, proposed action, and expected outcome to a human operator via the Grafana interface or an operations portal. Sylva's GitOps workflow provides a natural mechanism for this: the agent submits its proposed changes as a Git commit or merge request, which a human operator can review, validate, and approve before FluxCD reconciles the change into the live infrastructure, leveraging the same code-review patterns that engineering teams already use for infrastructure changes.
- **Action classification:** Actions are classified by blast radius (e.g., restarting a single pod vs. modifying a network-wide routing policy) with corresponding approval thresholds. This classification determines whether the action can be auto-executed, requires digital twin validation, or must be escalated to a human operator.

- **Rollback capability:** Because actions are expressed as GitOps commits, the same execution path described in the tool integration layer above, any AI-driven change can be automatically reverted if post-action monitoring detects degradation. This is not a separate rollback mechanism; it is an inherent property of the GitOps-driven architecture.
- **Audit trail:** Every agent decision, tool invocation (via MCP), model selection (via the inference gateway), and executed action is logged end-to-end, creating a complete audit trail for compliance and post-incident analysis. This traceability chain is what makes the entire architecture accountable, and it is where the OpenTelemetry distributed tracing gap identified in **Section 2.4** becomes most acute.

The guardrails layer does not operate in isolation, it depends on and completes the three preceding components. MCP servers enforce what the agent can access. The inference gateway governs which model it reasons with. Guardrails govern whether and how the agent is permitted to act on its conclusions.

Together, the four components form a coherent governance chain from observation through reasoning to action.

Note that with Project Salus[4], The Linux Foundation is proposing and exposing an AI framework to enhance trust and transparency in AI

4. https://project-salus.org

3.3 The closed-loop automation extended by Agentic AI

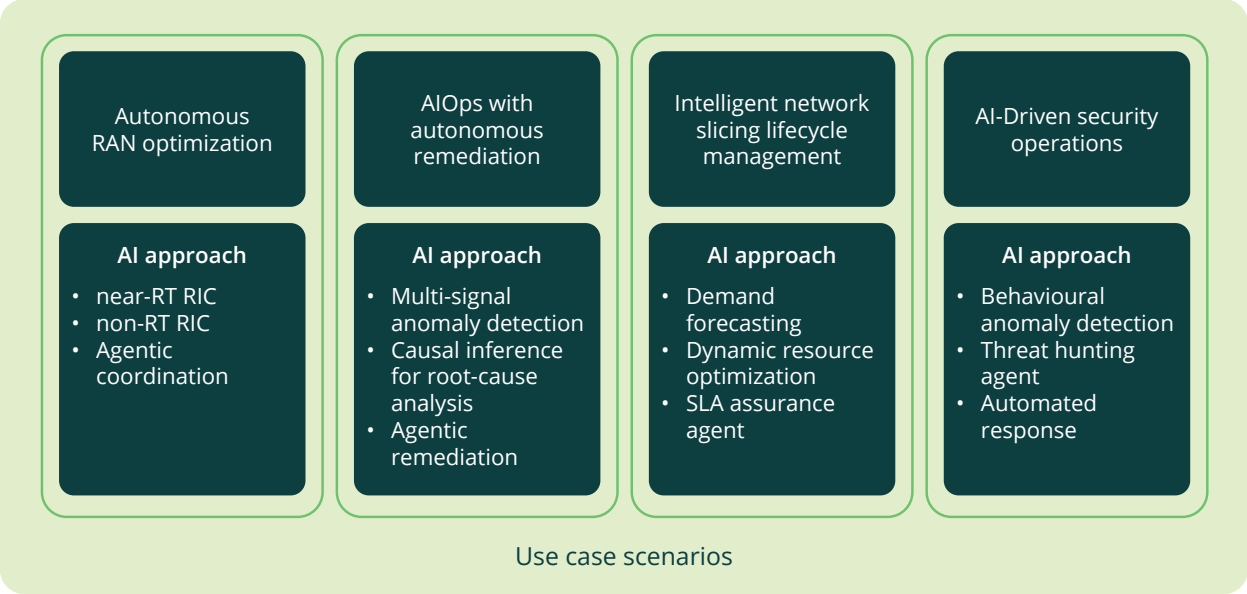
Traditional closed-loop automation in telecom follows the following main steps: Monitor, Analyse, Plan, Execute, and sometimes manage Knowledge. The importance of closed-loop automation is further reinforced by TM Forum's Autonomous Networks framework, which defines a widely adopted industry maturity scale from L0 (manual operations) to L5 (full autonomy), with most operators today sitting between L2 and L3.

Agentic AI transforms the established model into a more capable loop:

Closed-loop stages	Traditional Approach	Agentic AI Approach
Monitor	Fixed metric thresholds, static dashboards	Context-aware observation: the agent decides *what* to monitor based on current network state and recent events. As an example, use of declarative model (e.g.: K8s, GitOps)
Analyze	Single-purpose ML model (one model per anomaly type)	Multi-modal reasoning: the agent correlates metrics, logs, traces, topology, and even unstructured data (change tickets, vendor advisories) to form a holistic understanding.
Plan	Predefined playbook selection	Dynamic plan generation: the agent reasons about multiple possible remediations, evaluates trade-offs, and constructs a novel action plan if no playbook applies.
Execute	Script-triggered automation	Tool-mediated execution with verification: the agent executes each step, monitors the result, and adapts if the outcome differs from expectations.
Knowledge	Static knowledge base, periodic model retraining	Continuous learning: the agent updates its understanding based on action outcomes, and successful resolutions become new runbook entries.

4. Deep-Dive Use Case scenarios

For every following scenario, we will start by exposing the problem, setting the context, and then we will highlight the key points of the AI approach, before considering what the Sylva project is enabling. Each scenario description ends with an operational flow, presenting the main steps.



4.1 Autonomous RAN optimisation

Problem: Open RAN architectures decompose the radio access network into disaggregated components (O-RU, O-DU, O-CU) managed by the RAN Intelligent Controller (RIC). Optimising performance across thousands of cells requires continuous adjustment of parameters, handover thresholds, beamforming weights, power levels, scheduler configurations, in response to changing traffic patterns, user mobility, and interference conditions. Human operators cannot manage this complexity in real time.

AI Approach:

- **Near-real-time RIC (near-RT RIC):** Hosts xApps that use reinforcement learning to optimize RAN parameters on a per-cell, sub-second basis. A deep RL agent learns optimal policies for power control and scheduling by interacting with the live network through the E2 interface.
- **Non-real-time RIC (non-RT RIC):** Hosts rApps that perform longer-horizon optimisation, traffic forecasting using transformer-based time-series models, interference pattern analysis using GNNs, GenAI models for orchestration and human-machine dialogue, and policy guidance for the near-RT layer.
- **Agentic coordination:** An agentic AI system operating at the SMO level can orchestrate across multiple xApps, for example, detecting that a coverage optimization xApp and an energy-saving xApp are issuing conflicting directives, reasoning about the trade-off, and setting appropriate policy guardrails.

Sylva Enablement:

- RIC and SMO components run as containerised workloads on Sylva's Kubernetes clusters.
- GPU resources for RL training and inference are managed through Kubernetes scheduling with device plugins.
- Model versions are deployed and rolled back through GitOps pipelines. The same GitOps workflow manages network configuration changes, RAN parameter updates, policy adjustments, and infrastructure scaling, ensuring a single, auditable execution path for both AI and network operations.
- Prometheus collects RAN KPIs (throughput, latency, handover success rate) that feed directly into the RL training loop.
- OpenTelemetry is used to collect multiple agent traces for observability and service assurance

Operational Flow:

1. Prometheus ingests RAN KPIs from O-DU/O-CU via OpenTelemetry collectors.
2. A time-series forecasting model (rApp) predicts a traffic surge in a stadium area for an upcoming event.
3. The agentic orchestrator evaluates available capacity, interference constraints, and energy targets.
4. It commits updated policy configurations and RAN parameter targets to the GitOps repository. FluxCD reconciles the changes, directing the near-RT RIC xApps to pre-position beamforming configurations and adjust handover parameters.
5. During the event, these RL-based xApps perform constrained near-real-time fine-tuning.
6. Post-event, the agent evaluates performance, logs outcomes, and updates traffic prediction models.

4.2 AIOps with autonomous remediation

Problem: Modern telco cloud environments running on Sylva comprise thousands of Kubernetes pods, virtual machines, network functions, and physical infrastructure elements. Operational incidents, performance degradation, partial failures, misconfigurations generate cascading alerts across multiple systems. Operators face alert fatigue, slow root-cause identification, and manual remediation processes that extend service impact.

AI Approach:

- **Multi-signal anomaly detection:** An ensemble of models analyses metrics (Prometheus), logs (OpenTelemetry/Loki), and traces (Jaeger/Tempo) simultaneously. Autoencoders detect metric anomalies; NLP models classify log patterns; trace analysis identifies latency bottlenecks.
- **Causal inference for root-cause analysis:** Rather than simply correlating symptoms, graph-based causal models (Graph Neural Networks) reason about the dependency structure of

the infrastructure to identify the most likely root cause. If Pod A depends on Service B, which depends on Node C, and Node C shows disk I/O anomalies, the model traces the causation chain.

- **Agentic remediation:** Once an AI agent hypothesizes an incident's root cause, it consults a knowledge base for past issues, then generates a precise remediation plan. Before execution, it assesses the blast radius and determines if human approval is necessary. The agent executes the plan using GitOps. Finally, it monitors post-remediation metrics to either automatically close the incident ticket or escalate the issue with full context if resolution is not confirmed.

Sylva Enablement:

- The observability stack (Prometheus, OpenTelemetry) provides the unified data plane.
- Kubernetes APIs expose the full state of the managed infrastructure.
- GitOps (FluxCD) provides the auditable execution path for configuration changes.
- KServe hosts the specialised ML models (anomaly detection, causal inference) as inference endpoints callable by the agent.

Operational Flow:

1. Prometheus alerts trigger when a 5G Core UPF (User Plane Function) shows increased packet processing latency.
2. The AIOps agent is activated. It queries Prometheus for correlated metrics across the UPF pod, its host node, and upstream/downstream network functions.
3. OpenTelemetry traces reveal increased latency on the storage path. Log analysis identifies repeated I/O timeout errors on the host node.
4. The causal inference model confirms: root cause is degraded storage performance on Node X.
5. The agent generates a plan: cordon Node X, migrate affected pods to healthy nodes, alert the infrastructure team for physical inspection.
6. Blast radius is classified as "medium" (impacts one node, not network-wide). The agent's proposed GitOps commit triggers a CI pipeline that validates the change against a digital twin environment; only if tests pass green is the merge auto-approved.
7. The agent commits the node cordon and pod migration as a GitOps change. FluxCD reconciles the change within seconds.
8. Post-action monitoring confirms UPF latency has returned to normal. The incident is auto-closed.

4.3 Intelligent network slicing lifecycle management

Problem: 5G network slicing enables operators to create multiple logical networks on shared physical infrastructure, each tailored to specific service requirements (eMBB, URLLC, mMTC). Slice lifecycle management, creation, scaling, assurance, and decommissioning involves

coordinating resources across RAN, transport, and core domains. Static provisioning leads to over-allocation and waste; manual adjustment cannot keep pace with demand variability.

AI Approach:

- **Demand forecasting:** ML models predict per-slice traffic demand based on historical patterns, time-of-day, events, and external signals (e.g., event schedules, weather data for IoT slices). The choice of model architecture should be driven by the latency and accuracy trade-off required for each slice type: simpler models such as autoencoders or LSTMs are often faster to train, cheaper to serve, and sufficient for steady-state time-series forecasting, while more complex architectures like transformers may be warranted only for slices with highly irregular or multi-variate demand patterns. Over-engineering the model adds inference latency and operational cost without proportional benefit; a pragmatic approach that matches model complexity to the problem is essential.
- **Dynamic resource optimisation:** A multi-objective optimisation engine (potentially RL-based) allocates compute, network, and spectrum resources across slices to meet SLA targets while minimising cost and energy consumption.
- **SLA assurance agent:** An agentic AI system continuously monitors slice-level KPIs against SLA thresholds. When degradation is detected or predicted, the agent autonomously adjusts slice resources, scaling up compute for a high-demand period, rerouting traffic for a transport bottleneck, or triggering a network function scale-out. Critically, in a multi-slice environment where resources are shared, any remediation risks impact adjacent slices. The agent therefore leverages a **digital twin** of the slice environment to simulate proposed changes before execution, validating that a remediation will achieve its intended effect without triggering unforeseen degradation on co-located slices or downstream dependencies.

Sylva Enablement:

- Kubernetes namespaces and resource quotas provide the native isolation mechanism for slice workloads.
- Prometheus monitors per-namespace (per-slice) resource consumption and performance metrics.
- The GitOps layer manages slice configurations as declarative manifests, enabling automated scaling and reconfiguration.
- Edge clusters managed by Sylva enable slice resources to be distributed across central and edge sites.

Operational Flow:

1. A high-level service intent is submitted by an industrial automation customer: “Connect 500 robots across three factory floors with end-to-end latency below 10ms and 99.999% reliability.”
2. The agentic system receives the intent and reasons about how to fulfil it. It analyzes the latency and reliability requirements, correlates with the available network topology, and determines that a URLLC slice is the appropriate response, selecting the required RAN profile (dedicated scheduling, low-latency slot configuration), transport priorities (QoS class,

dedicated path reservation), and core network topology (local UPF breakout for minimal round-trip).

3. The demand forecasting model predicts expected load based on similar industrial deployments.
4. The slice is provisioned via GitOps: Kubernetes resources, network function instances, and network policies are committed to Git and reconciled by FluxCD.
5. During operation, the SLA assurance agent monitors latency, jitter, and reliability KPIs.
6. The agent detects that latency is approaching the SLA threshold due to increased background traffic on the shared infrastructure. Before acting, it submits its proposed remediation, QoS policy adjustment and UPF instance scaling to a digital twin of the slice environment. The simulation validates that the changes will restore latency margins without destabilising adjacent slices sharing the same infrastructure (e.g., confirming that reallocating compute to the URLLC slice does not starve a co-located eMBB slice below its own SLA floor). Once the simulation confirms no adverse side effects, the agent commits the changes as GitOps updates and FluxCD reconciles.
7. It autonomously adjusts QoS policies and scales the UPF instance, expressing these changes as GitOps commits.
8. When the customer contract ends, the agent orchestrates graceful decommissioning, draining traffic, releasing resources, and archiving telemetry.

4.4 AI-Driven security operations

Problem: Telco networks are high-value targets for cyberattacks. The attack surface has expanded with cloud native architectures, containerised workloads, API-driven interfaces, multi-tenant infrastructure, and distributed edge sites, all of which introduce new threat vectors. Traditional signature-based security tools cannot keep pace with evolving threats. At the same time, regulatory frameworks such as the EU's NIS2 directive impose stringent requirements on telecom operators, including mandatory rapid incident reporting, proactive risk management, and "security by design" principles, making automated, AI-driven security operations not just an efficiency gain but a compliance necessity.

AI Approach:

- **Behavioural anomaly detection:** ML models, ranging from statistical methods and tree-based ensembles to deep learning architectures, depending on the complexity and volume of the data, establish baseline behaviour profiles for network traffic, API calls, and user sessions. Deviations trigger investigation, not just simple threshold alerts, but contextual analysis that distinguishes a legitimate traffic spike from a DDoS attack, or normal API usage from credential stuffing.
- **Threat hunting agent:** An agentic AI system proactively searches for indicators of compromise across the infrastructure. It queries network flow data, correlates with threat intelligence feeds (via RAG), analyses container image provenance, and inspects Kubernetes RBAC configurations for anomalous privilege escalation.

- **Automated response:** For confirmed threats, the agent can execute containment actions, isolating compromised pods via network policies, rotating credentials, and blocking malicious IPs at the ingress, while preserving forensic evidence.

Sylva Enablement:

- Kubernetes network policies provide the enforcement mechanism for AI-driven containment.
- OpenTelemetry captures the API-level and network-level telemetry required for behavioural analysis.
- GitOps ensures security policy changes are version-controlled and reversible.
- Container image signing and admission controllers (part of Sylva's security baseline) provide the provenance data for supply chain analysis.

Operational Flow:

1. Prometheus metrics and Loki log streams detect an anomalous spike in API call volume, with patterns inconsistent with normal NF discovery traffic.
2. The behavioural anomaly detection model classifies the pattern as a potential credential-stuffing attack against the service-based interface.
3. The threat hunting agent is activated. It correlates the anomalous traffic with recent Kubernetes RBAC events, container image deployment logs, and external threat intelligence feeds (via RAG over threat databases).
4. The agent identifies that the source is a recently deployed pod running an image with an unsigned layer, flagged by the admission controller but permitted under a temporary exception policy.
5. The agent generates a containment plan: apply a Kubernetes NetworkPolicy isolating the suspect pod, rotate service account credentials, and revoke the image exception policy.
6. Blast radius is classified as "low" (single pod, no lateral movement detected). Automated containment is permitted per policy.
7. The agent commits the NetworkPolicy and credential rotation as GitOps changes. FluxCD reconciles within seconds. The exception policy revocation is committed separately with an audit annotation.
8. Post-containment, the agent continues monitoring for residual indicators of compromise across adjacent network functions. Finding none, it generates a forensic report preserving the full evidence chain, anomalous traffic captures, correlated events, reasoning trace, and actions taken, and closes the security incident.

5. Challenges, guardrails, and recommendations

The preceding sections have laid out a compelling vision: AI, and particularly agentic AI, integrated with Sylva's cloud native infrastructure, can transform telecom operations from reactive and manual to predictive and autonomous.

The architectural alignment is strong, the use cases are concrete, and early explorations have demonstrated feasibility. However, moving from proof of concept to production-grade deployment on live networks serving millions of subscribers is a fundamentally different challenge. The barriers are no longer about whether AI can work on Sylva; they are about whether it can work **safely, reliably, and at scale** within the operational, regulatory, and trust frameworks that govern telecom.

The challenges selected below reflect the most critical gaps that the whole ecosystem must address to make this transition. They span the full lifecycle: from the quality of data feeding AI models, through the operational discipline of keeping those models accurate over time, to the trust and governance frameworks required before any operator will allow an autonomous agent to modify a production network.

Ignoring any one of these would be sufficient to stall adoption; addressing them collectively is what separates a successful AI-native telco platform from another generation of stranded PoCs.

5.1 Data governance and quality

AI models are only as good as their training data. Operators deploying AI on Sylva must implement:

- **Data cataloguing:** A unified registry of available telemetry sources, schemas, and data quality metrics across the Sylva observability stack.
- **Data pipelines:** Scalable ETL/ELT pipelines (e.g., Apache Kafka, Apache Flink) running as Sylva workloads to prepare training data from raw telemetry.
- **Privacy and compliance:** Telemetry data may contain subscriber information subject to GDPR and sector-specific regulations. AI training pipelines must incorporate anonymisation and access controls.

5.2 Model lifecycle management (MLOps)

AI models in a live network degrade as conditions change (concept drift). A robust MLOps practice on Sylva includes:

- **Continuous monitoring** of model performance metrics (accuracy, drift indicators) via Prometheus.
- **Automated retraining** is triggered when performance degrades below thresholds.
- **Model versioning and rollback** through GitOps, treating models as declarative artefacts alongside infrastructure configuration.

- **A/B testing and canary deployment** of model updates, using Kubernetes traffic splitting to validate new models on a subset of traffic before full rollout.

5.3 Trust, explainability, and human oversight

The most significant barrier to agentic AI adoption in telecom is **trust**. Network operators will not cede control to autonomous systems without:

- **Explainability:** Agents must be able to articulate their reasoning, why they believe a root cause is X, why they chose remediation Y, what risks they considered. This is not optional; it is a regulatory and operational requirement.
- **Graduated autonomy:** Operators should progressively expand the scope of autonomous action as confidence grows, starting with "recommend only," progressing to "auto-execute low-impact actions," and eventually reaching full autonomy for well-understood scenarios.
- **Safety boundaries:** Hard constraints that AI agents cannot override, e.g., an agent must never simultaneously modify more than N% of network resources, must always preserve a minimum redundancy level, and must escalate if uncertainty exceeds a defined threshold.
- **Intent alignment and enforcement:** Agents increasingly operate from high-level intents rather than explicit instructions. This creates a new trust requirement: continuous verification that the agent's actions remain aligned with the original intent throughout the execution lifecycle. If a service intent specifies "<10ms latency for 500 robots," every decision the agent makes, slice type selection, resource allocation, remediation actions, must be traceable back to that intent, and the system must detect and flag intent drift when an agent's actions diverge from the stated objective.

5.4 Bridging the Agentic AI gap: What we need from ecosystem collaboration

The challenges described above cannot be solved by any single operator or vendor in isolation. They require open-source, community-driven solutions, shared building blocks that the entire ecosystem can adopt, contribute to, and trust. We identify four pillars where the ecosystem collaboration could have concrete capabilities:

- **Standardised Agent Safety & Governance.** How do we allow autonomous agents to act on critical network infrastructure without compromising safety? Today, every operator exploring agentic AI is inventing its own guardrails from scratch, its own policy schemas, its own approval workflows, its own audit formats. The industry needs open, shared frameworks for agent governance: zero-trust agentic architectures, a clear separation of reasoning and execution planes, auditable decision logs with causal traceability, and human-in-the-loop patterns as first-class primitives. This is not a competitive differentiator; it is common ground that must be built together.
- **Interoperable Agent Interfaces.** How do agents discover and interact with network capabilities across domains, vendors, and legacy systems? MCP and A2A (Agent-to-Agent) protocols provide the foundation, but their adoption must extend across all network domains and different standardisation bodies, RAN, transport, and core, with intent as the primary abstraction (declarative, not imperative), standard capability discovery and intent schemas, brownfield adapters for legacy OSS/BSS, and vendor-neutral data normalisation. Open source

is the only path to building these interfaces in a way that no single vendor controls.

- **Production-Grade Observability.** How do we observe not just the network, but the agents operating on it? Traditional monitoring tells us what happened to the infrastructure; it does not tell us *why an agent made a specific decision*. We need structured reasoning traces (not just logs), cross-agent correlation and dependency maps, compliance-ready explainability reports, and real-time anomaly detection on agent behaviour itself. This is where the OpenTelemetry gap identified in Sylva's stack becomes most acute, and where community collaboration can deliver a shared observability standard for agentic operations.
- **AI Result Confidence & Evaluation.** How do we know whether to trust an agent's output before it modifies a live network? Unlike traditional software, AI operates in a domain of probabilistic confidence, and the telecom industry lacks shared frameworks to evaluate it. We need open benchmarks for telco-specific agent tasks, confidence thresholds tied to action criticality (a recommendation to investigate is not the same as a decision to reroute live traffic), and continuous evaluation loops that run alongside agents in production, not just pre-deployment testing. Building these evaluation frameworks collaboratively avoids every operator reinventing the same trust infrastructure independently.

These pillars represent an open source agenda required to move agentic AI from demonstrations to production telecom operations. The Sylva community, with its position at the intersection of cloud-native infrastructure, telecom operations, and the Linux Foundation ecosystem, is uniquely placed to lead this effort.

6. Conclusion and call to action

This paper opened with a diagnosis: (Agentic) AI in telecom is easy in exploration mode, hard to graduate to production. The barriers are not primarily about model performance; they are about infrastructure fragmentation, missing enterprise-grade operational fabric, and the absence of shared governance frameworks. These are structural problems that require structural solutions.

The Sylva project provides the foundational answer. Its standardised, cloud-native stack, with Kubernetes as the runtime, GitOps as the auditable execution path, and a built-in observability layer, gives AI systems the substrate they need to move from isolated experiments to production-grade operations.

When AI is built on Sylva, models run where network functions run, telemetry feeds AI pipelines natively, and every AI-driven action is version-controlled, reversible, and traceable.

The emergence of agentic AI raises the stakes further. Autonomous agents that can reason, plan, use tools, and act on the network, integrated through standardised interfaces like MCP, represent the path from today's TM Forum Level 2–3 automation toward Level 4 and beyond: networks that don't just respond to human commands but genuinely self-optimise, self-heal, and self-protect. The four use cases explored in this paper, autonomous RAN optimisation, predictive AIOps, intelligent network slicing, and AI-driven security operations, demonstrate that this is not speculative; the architectural alignment between agentic AI and Sylva is concrete and actionable.

But technology alone is not sufficient. Moving to production demands that the ecosystem collectively address data governance, model lifecycle management, trust and explainability, and the four pillars identified in **Section 5.4**: standardised agent safety and governance, interoperable agent interfaces, production-grade observability for agentic operations, and shared frameworks for AI result confidence and evaluation. These are not competitive differentiators; they are common ground that must be built together, in the open.

Recommendations for operators

1. **Treat Sylva as the default runtime for telecom AI workloads:** avoid building parallel, siloed AI platforms that recreate the fragmentation this paper describes.
2. **Invest in data infrastructure first:** unified telemetry pipelines are the prerequisite for effective AI. Without high-quality, accessible data, no model will deliver production value.
3. **Begin agentic AI adoption with bounded, low-risk use cases** (e.g., automated incident triage, read-only infrastructure assessment) and expand scope incrementally through graduated autonomy.
4. **Participate in open-source standardisation** of AI interfaces, agent governance frameworks, and evaluation benchmarks; the ecosystem collaboration agenda outlined in **Section 5.4** needs operator involvement to reflect real operational requirements.
5. **Factor sustainability into AI operations:** Sylva's Kepler integration provides energy and carbon metrics that enable AI-driven optimisation of infrastructure efficiency. This responsibility should be embedded from the start, not retrofitted.

6. **Share AI-for-network architectures openly:** operators should contribute their use case architectures, integration patterns, and lessons learned to public open-source platforms such as the LF Networking 5G Super Blueprint initiative[5], which provides a community-driven framework for blueprinting end-to-end solutions across multiple open-source projects. Publishing validated AI + Sylva architectures as Super Blueprints accelerates ecosystem adoption, avoids every operator solving the same integration challenges independently, and creates a shared reference that vendors and integrators can build against.

For the Sylva project community

We identify three high-priority calls to action:

1. **Prioritise OpenTelemetry and distributed tracing as a core feature of the Sylva stack.** The current observability layer, while strong on metrics and logs, lacks the distributed tracing capability essential for advanced AI operations. OpenTelemetry, the CNCF standard already adopted across the cloud-native ecosystem, is the natural fit. Adding OpenTelemetry Collector support and distributed tracing to the Sylva reference implementation would close the most significant observability gap identified in this paper and unlock the full potential of AI-native telecom operations on Sylva.
2. **Complement and scale the development of production-grade Sylva MCP server with best-in-class security.** An early-stage Sylva MCP server has been open-sourced, proving the concept with read-only cluster lifecycle observability. The community should now build on this foundation to deliver a reference MCP server covering the full Sylva stack, with enterprise-grade security (authentication, authorisation, accountability), an MCP gateway pattern for centralised policy enforcement, and full audit traceability. This is the critical missing layer that transforms Sylva from an AI-ready substrate into a platform where agentic AI systems can operate safely and autonomously at production scale.
3. **Shape Sylva Agentic AI by packaging and integrating ML frameworks, inference runtimes, and intelligent inference routing into the Sylva Core reference stack.** Today, AI frameworks such as PyTorch, Ray, vLLM, KServe or LLM-d can run on Sylva's Kubernetes clusters, but they are not packaged, validated, or lifecycle-managed as part of the Sylva reference implementation.The same is true for inference routing, production agentic AI requires an intelligent gateway that dynamically routes agent queries to the right model backend based on cost, accuracy, latency, and data sovereignty constraints, a capability that does not exist in Sylva today. The community should work toward curated, Sylva-validated distributions of key AI components, model serving runtimes, training operators, agent frameworks, and an inference routing gateway, delivered as Sylva units with GitOps-managed lifecycle, tested in Validation Centers alongside network functions. This would lower the barrier for operators to deploy AI workloads on Sylva without each one assembling and maintaining its own AI toolchain independently, and would ensure that model selection governance is built into the platform from the start rather than bolted on after the fact.

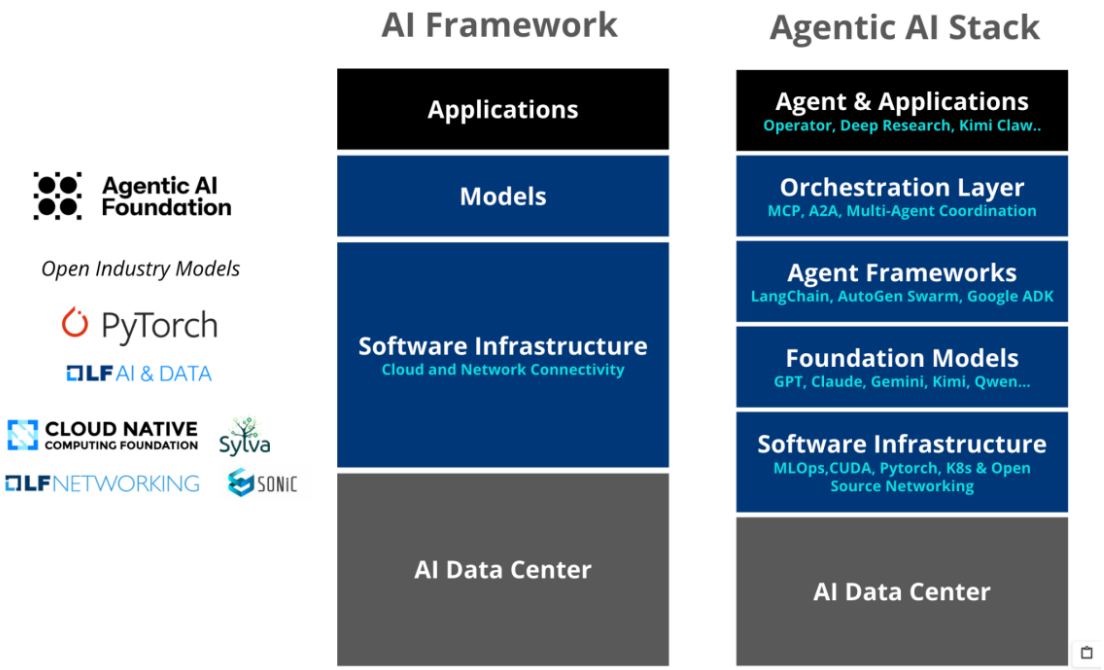
5. https://lf-networking.atlassian.net/wiki/spaces/LN/pages/15671228/5G+Super+Blueprints

For the broader ecosystem

The opportunity before us is to define, collectively, in the open, the reference architecture for AI-native telecom infrastructure. Not AI bolted onto the side of a cloud stack, but AI as a first-class architectural concern: integrated into observability, woven into automation, governed by shared standards, and evaluated by common benchmarks.

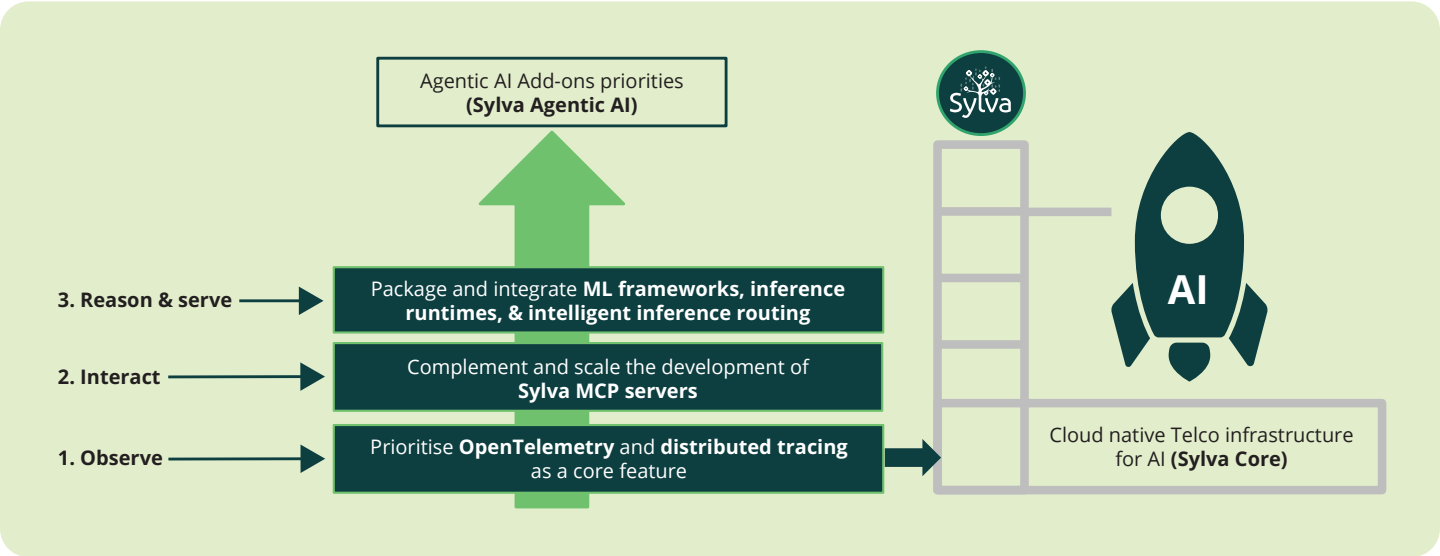
This effort is already underway. The LF Networking white paper "Architecting Autonomy: The Convergence of Agentic AI and Open Source Networking" (Joshipura, Tariq, Haiby, February 2026) proposes an Agentic AI Readiness Charter with ten architectural principles for building agentic-ready network applications, covering agent-native interfaces by design, intent as the primary abstraction, deterministic execution with governed autonomy, separation of reasoning and execution planes, first-class observability for agents, secure and auditable agent access, composable and discoverable capabilities, multi-domain coordination, backward compatibility, and open governance. These principles directly operationalise the four ecosystem collaboration pillars identified in **Section 5.4** of this paper, and they explicitly position Sylva alongside Nephio, ONAP, CAMARA, and SONiC as core LF Networking projects where agentic capabilities must be embedded. The paper also frames the dual transformation that Sylva enables: "AI for Networks" (agents that optimise and automate infrastructure) and "Networks for AI" (infrastructure optimised for distributed training and inference), both themes that run through the present document. The full paper is available at: https://lfnetworking.org/architecting-autonomy-convergence-of-agentic-ai-open-source-networking/

Both Sylva and MCP now live under the Linux Foundation. The standards bodies, ETSI, TM Forum, O-RAN Alliance, … are converging on closed-loop automation and intent-driven operations. The building blocks exist. What remains is the commitment to assemble them together, in the open, so that the next generation of autonomous telecom networks is built on shared foundations rather than proprietary silos.



Putting Sylva in perspective with Linux Foundation Strategy for AI, here is a simple overview of projects across the Agentic AI ecosystem.

The ground station is ready. It’s time to aim for the moon.



About the author

Philippe Ensarguet is an Orange Fellow, VP Cloud & Software Engineering, and 2021 award-winning CTO, championing at the crossroads of AI, Cloud Native, and platform innovation, currently driving telco-to-techco transformation. A three-time CTO with 25+ years' experience, he advises executives, VC funds, and tech startups. Board Advisor for Linux Foundation Europe, Sylva project governing board co-chair and active in open source communities, Philippe shapes technology strategy through advisory, mentorship, and thought leadership. Proud member of Tech.Rocks community.

Acknowledgement

The author wants to thank Vladimir Braquet (Orange), Carlo Cavazzoni (Telecom Italia), Eric Debeau (Orange), Taoufik En Najjary (Orange), Hanen Garcia (Red Hat), Ranny Haiby (LF Networking), Tim Irnish (Suse), Arpit Joshipura (LF Networking), Guillaume Nevicato (Orange), Mathieu Rohon (Orange), Olivier Simon (Orange), Kaï Steuernargel (Deutsche Telekom), Luis Velarde (Telefonica), Felipe Vicens (Telefonica) for their precious feedback and contributions during working on this white paper.

Thank you

https://sylvaproject.org