SVAIAM

A reference architecture and implementation of a Sovereign AI stack

Co-authored by contributors from BharatGen, IBM and NxtGen listed at the end of this article
27 Sep, 2026

Sovereign AI is an increasingly important desired architectural principle in India and across the world. However, it is not very well understood how to define and implement a Sovereign AI stack in practice. Most work in this area has been limited to specific components or capabilities, such as Sovereign AI models, or Sovereign infrastructure, or data residency policies, etc. In this paper, we present a reference architecture and realized implementation of a full Sovereign AI stack that has all the required technology layers and shows how to achieve functional and non-functional requirements that are critical for adoption and scaling. We call this reference architecture as SVAIAM – SoVereign AI Architecture Manifest. SVAIAM written in Devanagari script as **स्वयं** means "self".

1 Characteristics of a Sovereign AI Stack

It is worth defining recognizing some of the key characteristics that define the sovereignty aspects of Sovereign AI:

- **Data residency and control.** Data, metadata, and derived knowledge remain under the control of the organization or jurisdiction, with configurable guarantees around residency, privacy, access, lineage, and compliance. This enables adherence to regulatory requirements while preserving ownership of sensitive information.
- **Operational sovereignty.** Organizations retain independent control over the deployment, operation, monitoring, lifecycle management, and security of the AI platform, with no required dependence on an external operator or service provider.
- **AI sovereignty and choice.** Users should have the flexibility to select, combine, replace, and fine-tune AI models from multiple providers—including sovereign models, open-source, commercial, and domain-specific models—without architectural lock-in. This ensures that AI capabilities can evolve with changing requirements, performance characteristics, and policy considerations.
- **Technology and ecosystem sovereignty.** A sovereign AI stack should be built on open standards, interoperable interfaces, and modular components that minimize vendor lock-in and promote portability across infrastructure and also integration with other digital public infrastructure. Equally important is the ability to cultivate local innovation through extensible platform services, enabling governments, enterprises, academia, and independent software vendors to contribute domain-specific models, applications, and AI services.
- **Security and defence compatibility.** The stack should be operable in sensitive nationally controlled environments including air-gapped operations if needed.

2 Requirements

Sovereignty characteristics are certainly critical, but they should be achieved without compromising on non-functional requirements that are important for AI solution builders to obtain value from the stack. These requirements include:

1. **Quality.** The stack should provide access to state-of-the-art AI capabilities across foundation models, retrieval systems, unstructured data processing, orchestration frameworks, and infrastructure, ensuring that sovereign deployments can achieve accuracy, reliability, latency,

and user experience comparable to leading global AI platforms, especially in the context of regional/national linguistic, cultural and user requirements.

2. **Cost.** The architecture should optimize the total cost of ownership by enabling efficient utilization of compute, storage, networking, and AI accelerators. Support for heterogeneous infrastructure, model optimization, caching, and intelligent workload placement allows organizations to balance performance with economic efficiency.
3. **Developer agility.** The stack should provide AI developer tools that speed up AI solution development to reduce time to value and cost to build for developers. Given AI models and technologies evolve rapidly, these tools should allow for experimentation with new models and technologies without lock-in.
4. **Scalability.** The platform should scale seamlessly from small proof-of-concept deployments to enterprise and national-scale AI workloads. This includes elastic infrastructure, distributed inference and training, multi-tenancy, high availability, resilience, and operational automation across diverse deployment environments.
5. **AI solution governance.** Beyond infrastructure governance, the stack should provide comprehensive lifecycle governance for AI applications, including evaluation, observability, guardrails, explainability, lineage, policy enforcement, security, and compliance. These capabilities ensure that AI solutions remain trustworthy, auditable, and continuously improve throughout their operational lifecycle while meeting organizational and regulatory requirements.

3 Architecture

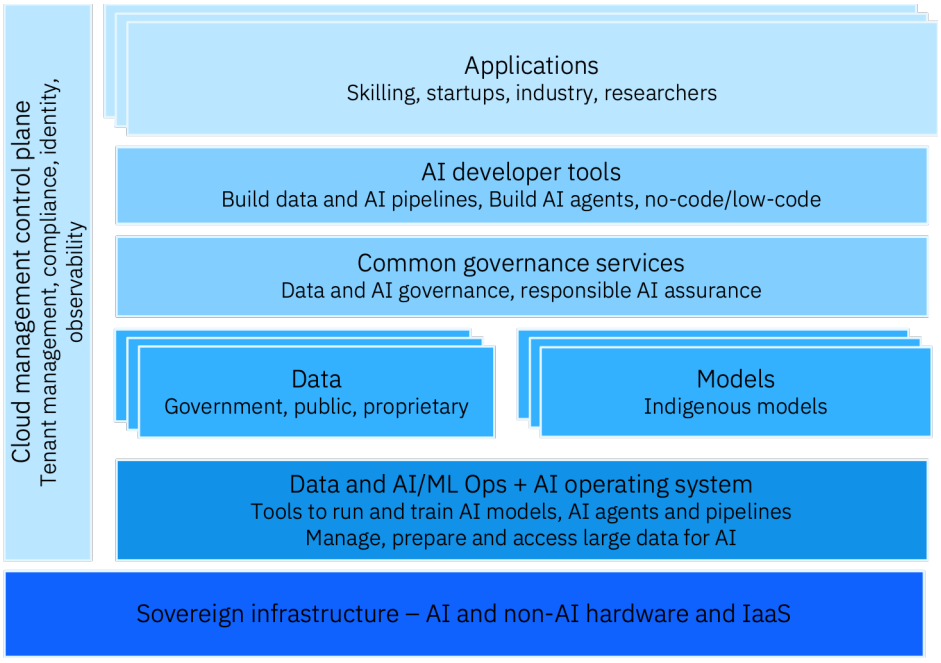


Figure 1 The technology layers of SVAIAM

The architecture of SVAIAM, illustrated in Figure 1 includes multiple layers across hardware infrastructure, software, data and models, designed in a way to address the requirements and characteristics listed in Sections 1 and 2:

1. **Sovereign Infrastructure** – This layer provides compute, storage and networking in the form of Infrastructure as a Service (IaaS), including GPU and/or non-GPU AI accelerators. The entire infrastructure and its operations are situated within the sovereign boundary.

2. **Data and AI/MP Ops + AI Operating system** – This layer provides the AI and data platform as a service. This includes tools to efficiently run and train AI models, and to run AI agents and AI pipelines. It also includes tools and services to manage, prepare and provide governed access to large structure and unstructured data for use in training or providing context to the AI agents. The architecture of the layer supports multiple tenants with appropriate tenant management.
3. **Data** – The architecture of the stack needs to allow for integration of varied sources of data, including structured and unstructured, and also across static (e.g. trusted knowledge documents) and dynamic (e.g. REST APIs to data services with changing data). There should be templates and accelerators for integration of such data with AI pipelines and agentic loops.
4. **Models** – The stack should allow for bringing in AI models from any provider, including open weight, sovereign and proprietary models, for deployment and serving in the sovereign boundary.
5. **Common governance services** – This layer provides standardized tools to monitor various aspects of model and data governance - across quality, performance, lineage, trustworthiness, versioning. The layer also provides standardized policies and metrics that can be reused across different applications, accelerating the time to quality, trustworthiness and compliance for application developers.
6. **Cloud management control plane** – This cross-cutting capability provides the "control plane" to observe, manage, secure and optimize across all the other layers, while also providing the tenant management and services management expected from a fully capable cloud-native environment, while meeting the characteristics of a sovereign AI Stack. It provides all these capabilities within the sovereign boundary with full operational control with the central IT team managing the AI stack deployment, without any dependence on external organizations or jurisdictions. This includes also a single pane of glass for continuous compliance with regulatory requirements (e.g. DPDP, IT Act, GDPR, etc) and with any additional organization and sovereignty controls relevant.

4 Reference Solution for Demonstrating SVAIAM

Let us look at a reference implementation of a full agentic AI solution based on SVAIAM. This reference solution is a voice-based AI agent for farmers that can converse in Telugu, Hindi and English and uses several structured and unstructured data sources to address the user's questions. While this is not the only solution template for AI applications, this is a very common solution archetype across a wide variety of end user personas and use cases.

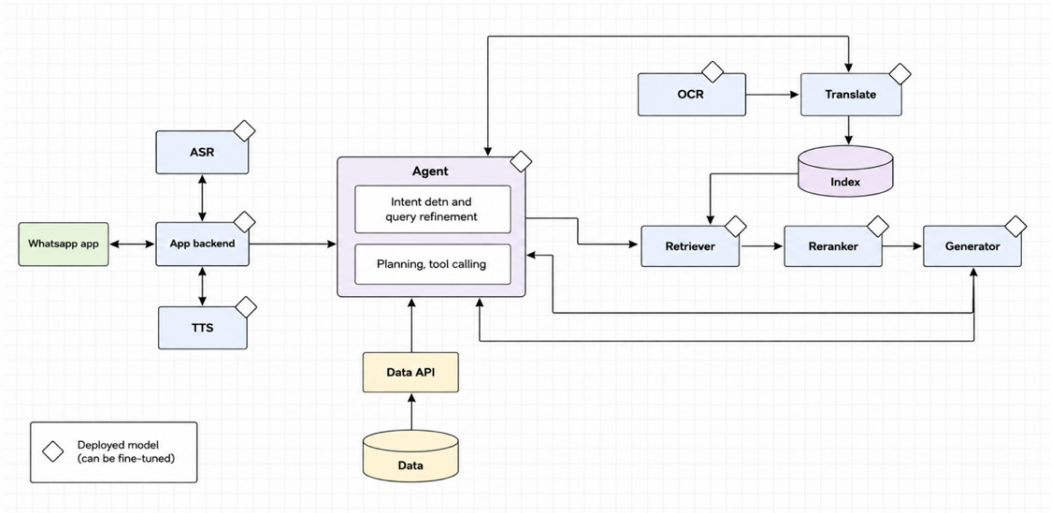


Figure 2 Solution Architecture: Component View

The reference implementation (Figure 2) demonstrates an end-to-end voice-enabled agentic AI system with the following key components:

WhatsApp App: The user-facing interface that enables farmers to interact with the AI agent through voice messages in their preferred language (Telugu, Hindi, or English), providing accessibility through a familiar platform with widespread adoption in rural India. The WhatsApp Business Account (WABA), the registered business phone number and the associated business portfolio through which this channel operates are owned, provisioned and managed by BharatGen.

App Backend: Orchestrates the overall user interaction flow, managing conversation state and coordinating between voice processing, agent reasoning, and response generation components.

ASR (Automatic Speech Recognition): Converts incoming voice messages into text using Shrutam, an indigenous model from BharatGen specifically optimized for Indian users, handling multiple Indian languages and regional accents with high accuracy.

TTS (Text-to-Speech): Synthesizes natural-sounding voice responses using Sooktam, also an indigenous model from BharatGen specifically suited for Indian users, enabling complete voice-based interaction with natural prosody appropriate for Indian languages.

Agent: The core reasoning engine powered by Gemma4-26B-A4B-it MoE model, kept lightweight mimicking a ReACT setup with comprehensive tools for RAG and API access. It performs intent detection, query refinement, planning, and tool calling. The agent includes both simple dense retrieval and HippoRAG, an advanced GraphRAG solution that excels in multi-hop reasoning for complex queries requiring information synthesis across multiple sources.

Data API and Data: Provides structured access to external data sources. While the AP deployment integrates government agricultural data, weather information, and market prices, this component is designed generically to support any set of APIs for external data access, ensuring data residency compliance within sovereign infrastructure boundaries.

OCR (Optical Character Recognition): Processes scanned documents using Docling integrated with Chandra OCR, one of the state-of-the-art indigenous OCR models suitable for Indic context. Chandra OCR accurately parses Telugu content and other Indian scripts including conjunct characters and diacritical marks, enabling digitization of regional language agricultural knowledge.

Translate: While the architecture supports translation, the AP deployment does not use a separate translator, instead relying on the multilingual capabilities of the retriever (bge-m3) and generator (Gemma4-26B-A4B-it) models to handle queries and content across Telugu, Hindi, and English natively.

Index: A Milvus vector database storing embeddings of processed documents, enabling efficient semantic search and retrieval at scale with support for multiple index types and distance metrics.

Retriever: Implements the RAG retrieval component using bge-m3, a lightweight encoder-based retriever selected for its strong multilingual ability. The model handles Telugu, Hindi, and English content effectively and can be easily replaced with any other retriever of choice.

Reranker: While the architecture supports reranking to refine retrieved results, the AP deployment has not implemented a reranker. However, the modular design allows easy integration if needed for improved retrieval quality in future iterations.

Generator: Synthesizes retrieved information with agent reasoning to produce accurate, contextual responses. Uses the same Gemma4-26B-A4B-it MoE model as the agent, ensuring efficient resource utilization through shared model deployment while maintaining high conversational quality.

AI governance tool: Provides comprehensive monitoring and evaluation capabilities, measuring agentic RAG-specific metrics including retrieval quality, tool usage effectiveness, reasoning accuracy, and end-to-end response quality. Enables live monitoring of deployed systems, tracking performance trends, and supporting continuous improvement through detailed trace analysis and custom KPI definition. In this implementation we use IBM watsonx.governance as the AI governance tool.

4.1 Design Philosophy

The reference implementation embodies a design philosophy that purposefully aligns technical choices with SVAIAM's sovereign core principles, balancing state-of-the-art capabilities with practical considerations of cost, flexibility, and continuous improvement:

Performance-driven model selection with architectural flexibility

- The stack is equipped with SOTA model choices and recipes, with selections motivated by rigorous performance evaluation on datasets curated specifically for the agricultural use-case.
- For Generation models, we considered Open models such as Gemma4, Gemma3, Granite4 and also indigenous models from Sarvam and BharatGen.
- The architecture enables plug-and-play replacement of each model component, ensuring that as better sovereign or open models become available, they can be seamlessly integrated without architectural changes, preventing vendor lock-in.

Cost-optimized design decisions

- Model choices explicitly consider cost trade-offs, balancing performance requirements with infrastructure economics.
- Strategic sharing of the Gemma4-26B-A4B-it MoE model for both agent reasoning and response generation ensures only a single model deployment, significantly reducing infrastructure costs while maintaining high performance.
- Selection of multilingual models (bge-m3 for retrieval, Gemma4 for generation) eliminates the need for a separate translation model, further reducing operational complexity and cost.

Advanced recipes for enhanced capabilities

- HippoRAG: A highly effective GraphRAG recipe for multi-hop reasoning and question answering, significantly boosting performance for complex queries that require synthesizing information across multiple sources.
- Intelligent synthetic data generation (SDG) recipes: Enable curation of high-quality synthetic data for fine-tuning retriever and generator models specifically for the use-case, supporting continuous improvement without extensive manual data labelling.

Data and context lifecycle

Data persistence and reuse are enabled through common document and vector stores that retain Docling outputs and indexed documents, respectively, while external API specifications are persistently stored for seamless reuse across applications. Context management is intentionally lightweight, with the agentic layer responsible for retrieving, retaining, and engineering task-specific context as part of its reasoning. The modular design makes this readily extensible to a dedicated semantic and memory layer by adding tools for creating, maintaining, and engineering persistent client context and/or historical behavioural data, similar to an LLM-Wiki-style architecture. This provides a generic foundation that allows developers to introduce progressively richer context services as application needs evolve, without requiring fundamental architectural changes

Developer-centric governance and control

- IBM watsonx.governance as the monitoring platform puts developers in control, allowing them to reuse standard metrics and/or define custom metrics such as use-case-specific KPIs.
- Enables continuous monitoring of deployed systems with real-time visibility into production behavior, empowering teams to make data-driven decisions about model updates and system optimizations.

5 Reference Implementation

The characteristics of the Sovereign AI stack, listed in Section 1, demand certain implementation requirements, which are listed in Table 1 along with the components that help satisfy the requirements.

Similarly, Table 2 illustrates capabilities that are implemented to address the different requirements for an effective Sovereign AI stack, along with the components used to implement the capabilities.

Table 1 Implementation requirements that enable the characteristics of a Sovereign AI stack and associated components used in the reference implementation.

Characteristics	Implementation requirement	Components used in the reference implementation
Data residency and control	All data kept and processed within India on Infrastructure deployed and controlled by Indian entity	NxtGen infrastructure
Operational sovereignty	The control planes deployed within sovereign boundary and managed by Indian entity	NxtGen managed PaaS with Red Hat OpenShift Container Platform
AI sovereignty and choice	Indian models deployed and used in the solution. Only open models with no use of models deployed outside India and no lock-in to any specific model provider.	BharatGen (Param2, Shrutam, Sooktam, Patram), Sarvam and open models
Technology and ecosystem sovereignty	Solution implementation components and models are open source.	Docling, OpenRAG, CUGA, SDG_Hub, vLLM
Security and defence compatibility	Except for the data integrations, admin and user access no external connections.	IBM Sovereign core deployed on organization's air-gapped infrastructure can allow fully air-gapped operations (not demonstrated in this implementation)

Table 2 Capabilities needed to address the requirements posed on an effective Sovereign AI stack, and associated components used in the reference implementation.

Requirements	Capability	Components used in the reference implementation
Quality	Co-optimize developer tools with sovereign and open models for regional linguistic and cultural requirements	Docling + Chandra.OCR CUGA + open models like Gemma 4 Sooktam and Shrutam

Cost	Optimize inference costs with distributed inferencing and heterogenous arch Multiple models on single GPU	llm-d on heterogenous hardware MIG + KServe
Developer agility	An India AI Kit for rapidly building high-quality AI solutions for India while meeting all the other requirements	Docling, CUGA, OpenRAG, vLLM, IBM watsonx.governance, SDG_Hub, VoiceERA integration, Indian and open models
Scalability	Runtime enablement of models Distributed inferencing	Runtime – Sooktam and Shrutam. llm-d
AI solution governance	Track agent performance to enable iterative improvement with benchmark synthesis and performance monitoring	IBM watsonx.governance SDG_Hub + Indic tasks

Let us look in some detail at the different functional capabilities of the reference implementation of SVAIAM in the context of the reference solution of Section 4.

5.1 Cost-effective and scalable model serving

vLLM is an open-source project with a vibrant community as part of the PyTorch foundation, for memory-efficient inferencing that is hardware agnostic (Nvidia, AMD and other accelerators). It can serve embedding models (for RAG), decoder-only models, as well as vision-language models (for use with docling as well as for document understanding tasks). Its paged attention implementation provides efficient kv-cache management, and together with continuous batching enables users to run large models and workloads on fewer GPUs while keeping throughput high. vLLM achieves 14-24x higher throughput compared to HuggingFace Transformers based inference[1] under high concurrent requests. Stated differently, vLLM can serve the same workload with fewer GPUs enabling cost-efficiency.

llm-d builds on vLLM to solve the distributed side of the inference serving problem. It enables a fleet of vLLM instances to behave as a single optimized inference service, with capabilities such as prefill-decode disaggregation, distributed kv-cache management, efficient load-balanced and kv-cache-aware routing for improved throughput and time-to-first-token. A recent benchmarking exercise[2] jointly conducted by IBM, NxtGen and Red Hat on a mixed 3-vendor 20-GPU OpenShift cluster is a strong data point here: llm-d's prefix-cache-aware routing improved throughput and time-to-first-token over a kubernetes round-robin baseline across every configuration tested, single-vendor and heterogeneous alike, with the improvement growing with pool size and heterogeneity. In the largest mixed-vendor test, a 20-GPU pool spanning three vendors saw a 91% throughput improvement at 85 queries per second and 5.4x better time-to-first-token compared to the Kubernetes baseline, driven by the fact that round-robin scheduling spreads requests evenly regardless of current pod capacity, causing lower-capacity or saturated pods to accumulate queue depth and reduce aggregate throughput. In contrast, llm-d's routing accounts for saturation and concentrates cache hits on warm pods so heterogeneous capacity gets used more effectively. Together, vLLM's efficient single-node serving and llm-d's fleet-wide routing let sovereign deployments reclaim stranded capacity across staggered hardware refreshes, keep data and inference on-premise for regulatory compliance, and enable cost-efficient scaling for large enterprise and population-scale workloads.

[1] https://vllm.ai/blog/2023-06-20-vllm

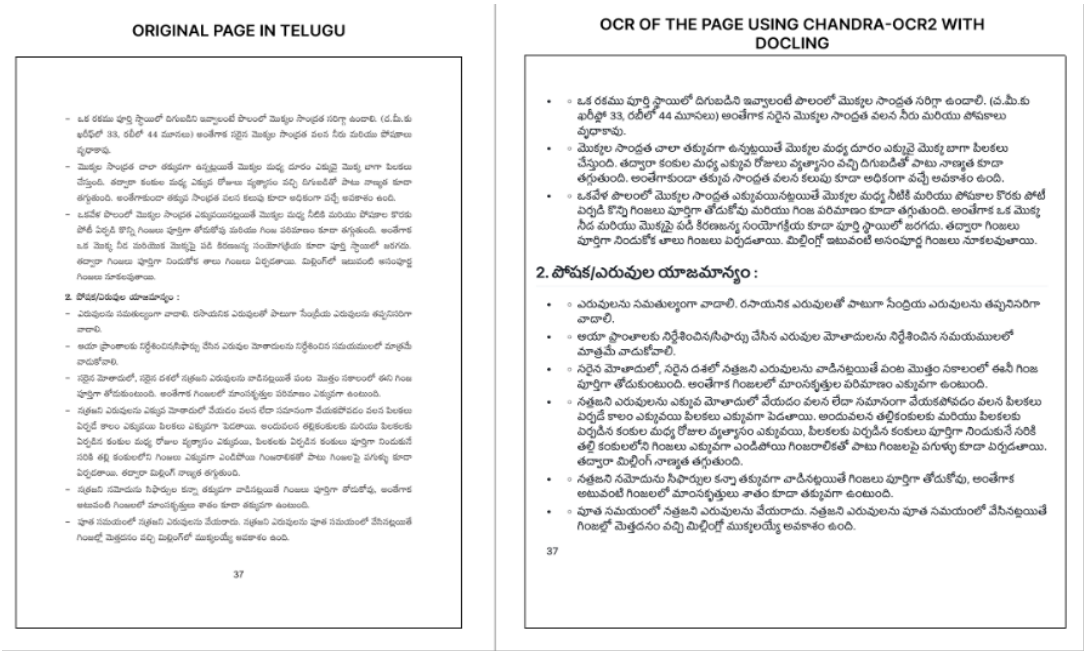
[2] https://llm-d.ai/blog/heterogeneous-inference-3-vendor-sovereign-cluster

5.2 Regional language document ingestion

OCR Selection: The document ingestion pipeline demonstrates a rigorous, performance-driven approach to processing regional language content while maintaining developer agility. At its core is Chandra-OCR-2, selected as one of the state-of-the-art models for Indic languages including Telugu through systematic evaluation on the Telugu corpus documents used for RAG. The model selection process involved sampling OCR outputs and validating them against native Telugu readers, with comparative analysis against alternative models to ensure optimal accuracy for the use-case.

Avg. Scores (0-5) (Human Eval)	Sarvam OCR	Dots. OCR	Gemma-3-27B	Deepseek-OCR	Surya-OCR	Chandra-OCR-2
Accuracy	4.89	4.61	2.45	2.89	4.4	4.94
Formatting	4.15	4.22	3.2	3.10	4.1	4.83

Docling Support: Chandra-OCR-2 is integrated with Docling,[3] a popular open-source document parsing pipeline that provides rich built-in support for interoperability between document structures and layouts by representing documents in a unified format. We integrated Chandra-OCR-2 support into Docling, enabling the same uniform parsing framework to seamlessly handle diverse document types and layouts. This integration makes the solution ready for processing diverse document types and extensible to other Indic languages as well. Docling also provides built-in configurations for different chunking strategies that directly feed into the indexing process, allowing developers to optimize chunk sizes and overlap based on their retrieval requirements.



Low code ingestion: The entire ingestion process in the toolkit is designed as a low-code experience where any developer can reuse built-in classes and demonstration notebooks with fewer than 10 lines of code to go from raw documents to a populated vector database for RAG use-cases. This combination of performance-validated models, flexible document processing, and developer-friendly abstractions exemplifies SVAIAM's principle of balancing quality with agility, enabling rapid

[3] https://github.com/docling-project/docling

deployment of sovereign AI solutions without compromising on accuracy for regional language content.

5.3 Robust RAG recipes

Generator Model Selection: The Generator model is the "brain" of the Agent and needs careful evaluation and benchmarking to test its robustness for the given language and task at hand. Our choice of Gemma-4-26B-A4B-it as the core LLM is driven by thorough benchmarking against other open-source and sovereign models, including BharatGen's Param2-17B-A2.4B, Gemma-3-31B-it, Sarvam-30B and GPT-OSS-20b, on our in-house Agri-QA Benchmark in Telugu. The benchmark is constructed using synthetic data generated with SDG_hub[4], followed by multiple layers of LLM-based verification and pruning and finally human-level verification. Gemma-4-26B-A4B-it topped the benchmark with the highest LLM-as-Judge score, underscoring its ability to generate accurate answers when handed the correct document. BharatGen's sovereign Param2-17B-A2.4B was rigorously evaluated on the same benchmark and held its own in the top tier of open and sovereign models: its score of 52.2 places it close behind the leading open models while activating only 2.4B parameters per token (compared with roughly 4B for Gemma-4-26B-A4B-it and 31B for the dense Gemma-3-31B-it), and well ahead of Sarvam-30B (4.2) and GPT-OSS-20B (8.4). This makes Param2 a strong, compute-efficient sovereign option for the generator role, and a natural candidate for the use-case-specific fine-tuning enabled by the SDG recipes in this stack.

	Gemma-4-26B-A4B-it	Sarvam-30B	Param2-17B-A2.4B	GPT-OSS-20B	Gemma-3-31B-it
LLMaJ (GPT-OSS-120B)	62.0	4.2	52.2	8.4	61.2

Choice of RAG Recipe: A RAG system needs to be robust enough to handle queries of varying complexity. A naive system might not be well suited to answer queries for which information is spread across the document in various different inter-connected chunks (multi-hop queries). In order to address this concern, we test various RAG recipes using the Gemma-4-26B-A4B-it model as the generator on our in-house Agri-RAG benchmark. We evaluate open-source RAG recipes including IRCoT,[5] LinearRAG,[6] and HippoRAG,[7] with HippoRAG-v2 reused and extended for our use case. We concluded that HippoRAG-2 is the preferred choice owing to its robust RAG ability, showcased by strong results on the RAG benchmark, while retaining the general reasoning and tool calling ability of the language model. To facilitate reuse and extension of these open-source RAG recipes, we use OpenRAG [8]and Langflow [9]for model swaps, hyper-parameter changes, and prompt alignment required for the Indic AP-Agri use case.

	Naive RAG	IR-CoT	LinearRAG	HippoRAG-2
LLMaJ (GPT-Oss-120B)	25.77	38.40	39.96	44.48

[4] https://github.com/Red-Hat-AI-Innovation-Team/sdg_hub
[5] https://github.com/StonyBrookNLP/ircot
[6] https://github.com/DEEP-PolyU/LinearRAG
[7] https://github.com/osu-nlp-group/hipporag
[8] https://github.com/langflow-ai/openrag
[9] https://github.com/langflow-ai/langflow

5.4 Government data API integration with agent

Source-aware data integration: The reference implementation treats government and domain data sources as governed tools rather than giving the language model unrestricted access to upstream systems. e-NAM supplies live and historical APMC trade data; Soil Health Card services supply soil-health, crop-registry, scheme and administrative-geography information; AgriWatch supplies price forecasts, market zones and intervention advisories; and the Vyavasaya Panchagam knowledge base supplies regional cultivation and farming-practice knowledge through RAG. Source-specific specialist agents retain the semantics of each REST, GraphQL, database or retrieval interface, resolve the required entities and return a common, source-labelled result to the agentic layer. The identification, API discovery, testing and provisioning of these data sources — the AgriWatch, e-NAM and Soil Health Card services and the Vyavasaya Panchagam knowledge base — were carried out by BharatGen, establishing the data foundation on which these specialist agents operate.

Governed tool publication through CUGA: NxtGen exposes these capabilities to the open-source Configurable Generalist Agent's (CUGA)[10] as five high-level operations covering market prices, highest-value commodity discovery, forecasts and advisories, soil and crop suitability, and farming practices. The operations are made available through an authenticated internal OpenAPI gateway and a CUGA Manager and Registry configuration. Tool definitions and the policies governing their use are activated from the same published configuration, ensuring that the runtime uses an approved and internally consistent set of capabilities. The planner sees only the approved operation schemas; credentials and source-specific access details remain within the trusted integration layer.

The gateway provides a bounded control point around the specialist services. It applies authentication, request validation, limits on input and returned data, concurrency and queue controls, execution timeouts, and sanitisation of credential-shaped fields. It also distinguishes successful, no-data and clarification outcomes before returning an observation to CUGA. The existing specialist agents continue to own the underlying business logic, allowing the integration contract to evolve without duplicating the implementation of the government and domain-data services.

Source-aware execution and traceability: The application first identifies the candidate information domains required for a question. A request requiring one independent source can use a direct fast path. Questions requiring multiple sources, or where the output from one source becomes the input to another, are passed to CUGA for coordinated tool execution. CUGA applies published tool guides and playbooks to select the appropriate operations, order dependent calls and combine the returned evidence into one response. Upstream API provenance, specialist-tool execution and response timing are captured in the application trace, supporting operational visibility and the investigation of incomplete or unexpected answers. This separation between governed data access and language-model reasoning is important for data control, auditability and the future addition of new public-data services.

5.5 Agentic loop with open and sovereign models

Hybrid agentic loop: Each interaction begins with the application loading the relevant conversation history and resolving follow-up references so that expressions such as "this crop" or "that market" become self-contained questions. Language detection, intent classification and dependency detection are then performed to determine the appropriate execution path. Straightforward questions requiring a single specialist are sent directly to that specialist. Multi-specialist questions and dependent workflows are routed through the CUGA orchestration layer, which we extended with policies and guidance for the AP-Agri use case, while general or out-of-domain questions can be evaluated against the published agriculture intent guard.

[10] https://github.com/cuga-project/cuga-agent

Within the CUGA path, the planner identifies the required tools, executes them in the appropriate sequence and uses intermediate results to construct subsequent calls. For example, the agent may first establish a suitable crop from soil information, obtain its market outlook and then retrieve the relevant cultivation practice. Tool guides describe when each capability should be used, while playbooks encode repeatable procedures for questions such as whether to grow or sell a crop, market-price checks and pest or disease response. CUGA then synthesises the evidence into a response appropriate to the user's language and channel. Bounded execution and a retained ReAct fallback provide continuity if the CUGA path does not complete, without weakening the separation between the planner and the underlying data services.

Open and sovereign model choice: The reasoning components communicate with models through an OpenAI-compatible interface rather than a provider-specific application contract. The reference implementation uses the performance-selected Gemma4-26B-A4B-it model for agent reasoning and response generation. BharatGen's Param2-17B-A2.4B, which showed strong performance relative to the top tier of open and sovereign models of similar class on the Agri-QA benchmark (Section 5.3), can be substituted directly through the same interface, as can other sovereign or open models, as their quality, language coverage, latency and cost characteristics evolve. Policies, playbooks, tool definitions and source integrations remain outside the model, allowing a model to be replaced or fine-tuned without redesigning the complete application.

Reference application on a multi-tenant sovereign platform: The farmer-advisory solution is one reference application on the NxtGen sovereign heterogeneous AI platform; it is not the limit of the platform architecture. The environment is designed to host multiple independently governed applications on a Red Hat OpenShift cluster running on an OpenStack sovereign cloud.

This design allows a range of applications to coexist on the same platform and to adopt the consumption mechanism appropriate to their users. The reference farmer-advisory application uses web, WhatsApp and voice interfaces, while other applications may use web channels, MCP clients, APIs and SDKs, or enterprise and event-driven integrations. OpenShift performs workload placement, scaling and service operations over the OpenStack infrastructure, while compatible workloads can be placed across heterogeneous accelerator capacity from AMD, NVIDIA and Intel. The result is a common sovereign platform capable of supporting Application 2, Application 3 and subsequent applications without requiring each solution to recreate its own infrastructure and operational foundation.

Distributed inference with llm-d: At the model-serving layer, llm-d complements the application-level role of CUGA. CUGA determines what reasoning and tool work is required; llm-d helps determine how model-inference requests are served efficiently across a pool of vLLM instances. Its prefix-cache-aware routing directs repeated prompt prefixes towards instances with reusable KV-cache state and takes current capacity into account, avoiding the limitations of simple round-robin placement. The architecture can also separate the compute-intensive prefill stage from token decoding so that each stage can be optimised across suitable resource pools.

In joint experiments conducted by NxtGen, IBM and Red Hat and on the NxtGen sovereign cloud, llm-d consistently improved throughput and time-to-first-token (TTFT) on both single-vendor and heterogeneous configurations. In the three-vendor, 20-pod experiment described, the llm-d configuration reached approximately 14,200 tokens per second under heavy traffic, compared with approximately 7,500 tokens per second for the conventional Kubernetes configuration, while reducing response time by nearly half a minute. Further details are covered in Section 5.1. The broader result indicated the potential to serve approximately twice as many users three to five times faster for the tested workloads. These results provide an important reference point for using heterogeneous capacity more effectively; realised gains will depend on the model, request pattern, cache reuse and serving configuration.

Together, CUGA at the application-orchestration layer, llm-d at the distributed model-serving layer, OpenShift at the workload-orchestration layer and OpenStack at the infrastructure layer create a modular separation of concerns. This allows the reference application to scale while also enabling multiple tenant applications to share the sovereign platform under defined isolation, policy and service requirements.

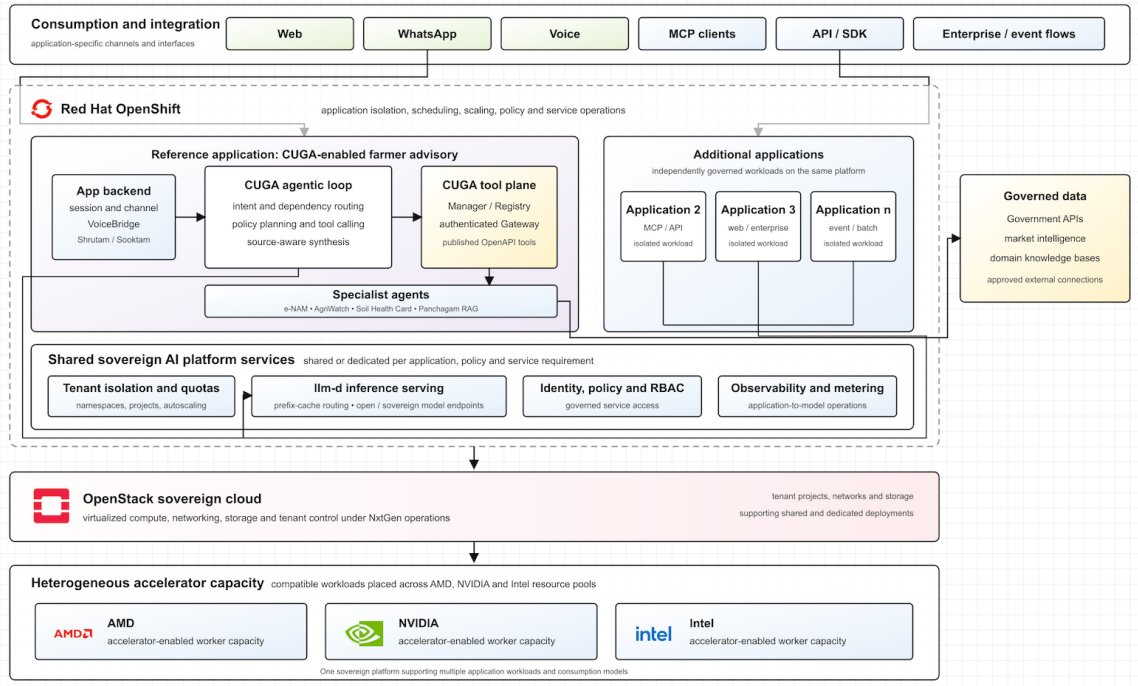


Figure 3 NxtGen sovereign heterogeneous AI platform used in this reference implementation of SVAIAM.

5.6 Voice integration and sovereign models

End-to-end voice integration: The reference implementation integrates voice as a first-class channel rather than as a separate agent workflow. Audio received through the web interface or WhatsApp is converted to a consistent 16 kHz mono PCM stream and passed through the VoiceBridge integration to Shrutam for automatic speech recognition. The resulting transcript enters the same context-resolution, classification, policy and CUGA orchestration path as a text request, ensuring that voice and text interactions use the same data sources, tools and guardrails.

Once the agent has produced its answer, the response is reformatted for spoken delivery and normalised to remove visual formatting and expand numbers, units and abbreviations that may otherwise be pronounced incorrectly. Sooktam, delivered through the BharatGen text-to-speech service, then produces the voice response. Longer answers are divided at suitable punctuation boundaries for synthesis and recombined into a single audio result. The web API can return the generated audio with the textual response, while the WhatsApp integration uploads and returns it as a voice message.

Sovereign voice choice and extensibility: Shrutam and Sooktam provide an indigenous voice path suited to Indian users and regional-language interaction, while NxtGen provides the channel integration, session handling and connection to the governed agentic loop. The demonstrated path supports Telugu and Indian-English interaction, and the service-oriented boundary allows additional Indian languages or improved sovereign models to be introduced as they are validated. VoiceEra, or another approved voice service, may therefore be evaluated as an alternate ASR or TTS provider through the same integration boundary without changing the CUGA policies, specialist agents or government-data connections.

Voice remains one of several application-consumption mechanisms supported by the platform. Each tenant application may independently choose voice, WhatsApp, web, MCP, API or enterprise integration according to its users and service requirements, while continuing to use the common sovereign infrastructure, model-serving and governance layers. This preserves channel flexibility at the application edge while maintaining consistent policy enforcement, source traceability and operational control across the platform.

5.7 BharatGen sovereign model suite: language, voice and vision

Across the stack, BharatGen contributes the indigenous model layer: a coherent family of sovereign models that together cover language, speech and vision for Indian users. In the reference solution these models sit at both ends of every interaction — Shrutam hears the farmer's question and Sooktam speaks the answer — with Param2 evaluated in the top tier for the reasoning core and Patram extending the same sovereign coverage to documents and images. Each is exposed through a standard service or OpenAI-compatible boundary, so it can be adopted, upgraded or replaced independently, consistent with the AI sovereignty and choice characteristic in Section 1. Table 3 summarises the suite.

Model	Modality	Role in SVAIAM	Status in reference implementation
Param2-17B-A2.4B	Language (MoE LLM)	Agent reasoning and response generation	Evaluated on Agri-QA; top-tier score of 52.2 (Section 5.3); drop-in via OpenAI-compatible interface
Shrutam	Speech-to-text (ASR)	Voice input from WhatsApp and web	Deployed in the voice path (Section 5.6)
Sooktam	Text-to-speech (TTS)	Spoken responses in the user's language	Deployed in the voice path (Section 5.6)
Patram	Vision-language (document VLM)	Document and image understanding over Indic content	Potential to be used and deployed on stack

Table 3 BharatGen sovereign models and their roles in the reference implementation of SVAIAM.

Voice — Shrutam and Sooktam: Shrutam is an automatic speech recognition model built for Indian languages and accents, and Sooktam is a text-to-speech model that produces natural prosody for Indian languages. Together they make voice the primary channel for farmers who are more comfortable speaking than typing.

Vision — Patram: Patram, BharatGen's vision-language model for document understanding, extends the sovereign model family from audio and text to vision. Where Chandra-OCR-2 converts scanned pages into text for indexing (Section 5.2), a document VLM can reason directly over page images — tables, forms, soil-health cards, advisories and mixed-script layouts — and can be served on the same vLLM runtime already used in this stack for vision-language workloads (Section 5.1). Integrated through the Docling pipeline, Patram gives SVAIAM an indigenous option for document understanding alongside OCR, and together with Shrutam and Sooktam provides sovereign multimodal coverage across vision, speech and language.

Data and channel foundations: BharatGen's contribution also extends beyond models: it carried out the discovery, testing and provisioning of the government data sources used by the agent (Section 5.4) and provides the WhatsApp Business channel infrastructure through which farmers reach the solution (Section 4). The result is a single indigenous thread through the reference solution — from the data the agent draws on, to the channel the farmer uses, to the models that listen, reason, read and speak.

5.8 Governance of AI solution for continuous improvement

The reference implementation employs IBM watsonx.governance as a governance and evaluation layer for systematic benchmarking, continuous evaluation, and iterative improvement, since it can be

deployed on any infrastructure including on-premise. The platform provides pre-built metrics such as recall and MRR, while enabling developers to define custom metrics, including LLM-as-a-judge evaluations using use-case-specific rubrics. This flexibility allows teams to define KPIs based on application requirements rather than being limited to predefined metrics.

watsonx.governance supports both offline and online evaluation. During development, offline evaluation enables successive model, prompt, retrieval, and architectural changes to be evaluated against a consistent benchmark. This creates a feedback loop where developers can make incremental changes, measure their impact on end-application quality, and select the checkpoint that delivers the best outcome.

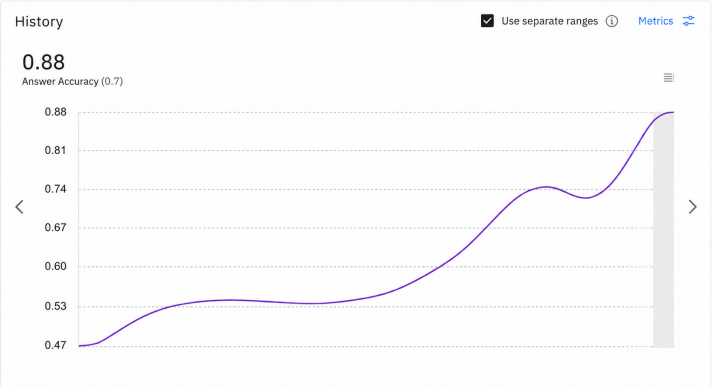


Figure 4 Solution quality improvement over development iterations driven by a structured and consistent evaluation framework implemented in IBM watsonx.governance (curve smoothed for easier trend viewing).

The diagram illustrates this workflow using Answer Accuracy as an end-application KPI. Across successive iterations, Answer Accuracy improves from approximately 0.47 to 0.88, with some intermediate fluctuations. The timeline makes these changes visible, helping developers identify the checkpoint with the strongest observed application performance. This demonstrates how governance-based evaluation enables evidence-based iteration and checkpoint selection rather than relying solely on subjective assessment.

For production deployments, watsonx.governance also enables online monitoring, with inference results evaluated and visualized through dashboards. Timeline-based KPI tracking helps teams identify trends, detect degradation, and use these observations to drive subsequent improvements.

This combination of standard and custom metrics, configurable LLM-based evaluation, offline benchmarking, checkpoint comparison, and production monitoring makes watsonx.governance a suitable governance layer for continuous quality improvement while maintaining developer control and supporting AI solution sovereignty.

6 SVAIAM implementation components

Sovereign AI deployment and scaling requires a full stack approach that empowers developers to rapidly develop high quality AI solutions, while meeting a number of requirements. Figure 5 annotates the different tools and providers contributing capabilities to the technology layers of SVAIAM in the reference implementation presented here. This combination of capabilities allows for the development and scaling of a full end-to-end voice-basic agentic AI solution powered by sovereign models on sovereign infrastructure and satisfying the characteristics and requirements of a Sovereign AI stack as per Sections 1 and 2.

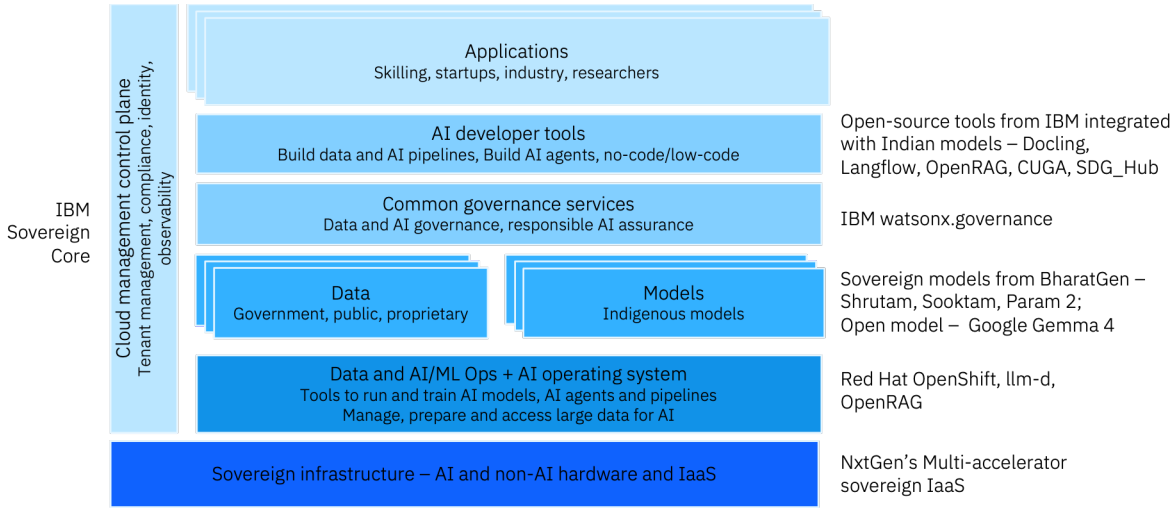


Figure 5 Tool and provider mapping to the technology layers of SVAIAM

However, much more can be done to address additional use cases requirements and continuously integrate and enhance this view with rapidly evolving technology. We look forward to further advancement and community engagement on this topic – there is certainly more to be done to further mature the architecture, tools and technologies that support Sovereign AI, and this is best done as a community effort.

7 Authors

BharatGen: Ganesh Ramakrishnan, Kundeshwar Pundalik, Piyush Sawarkar, Pranav Gaikwad, Bharat Patil, Kaustubh Deshpande, Amol Gite

IBM: Arkadeep Acharya, Tannavi Snehal, Rudra Murthy, Vrunda Gadesha, Ronak Jain, Jaydeep Sen, Sachindra Joshi, Amith Singhee

NxtGen: Abhisyant Anasapurapu, Jayanth Babu Reddy, Smrutikant Nayak, Abhishek Kumar Singh