# Agentic AI Technology Stack

## 1. Introduction

This document outlines the technical architecture and the technology choices made for the development of our AgenticAI. It provides detailed pros. for each technology selected, comparing them with available alternatives and explaining the technical advantages aligned with project requirements such as scalability, maintainability, real-time capabilities, and AI integration.

## 2. Frontend Stack

| Technology | Alternatives | Pros. | Comparison |
|---|---|---|---|
| **React 19** | Angular, Vue, Svelte, SolidJS | React 19 introduces significant performance improvements with React Compiler, enhanced Suspense for fine-grained async rendering, and modern features like Actions API. It supports future Server Components even though currently only client-side is used. Being the industry standard with a huge ecosystem, it ensures better onboarding, large community support, and compatibility with enterprise dev tools (e.g., Redux Toolkit, React Query, etc.). Also supports frameworks like Next.js if SSR is considered in future. | Compared to **Angular** and **Vue**, **React 19** offers **better performance optimizations** with **Enhanced Suspense** and the **React Compiler**, providing a more **scalable** solution for complex, large-scale applications. **Svelte** and **SolidJS** may be faster in specific scenarios, but they lack the mature ecosystem and industry support that **React 19** provides. **React** also has **better community support** and more **enterprise adoption**, making it the safest long-term choice for frontend development. |
| **TypeScript** | JavaScript | Provides type safety; enhances tooling and debugging; reduces production bugs; essential for multi-team, large-scale development. | TypeScript offers **early error detection**, **IDE support**, and **refactoring tools** that JavaScript lacks, making it the preferred choice for **maintainable enterprise applications**. |
| **Vite** | Webpack | Ultra-fast dev server with native ESM; faster HMR (Hot Module Replacement); minimal configuration; optimized production build process. | Compared to Webpack, Vite provides a **significantly faster dev experience**, **simpler configuration**, and better **tree-shaking**, especially beneficial for modern frontend projects. |
| http://Socket.io **Client** | WebSocket API, SSE | Simplifies real-time, event-driven communication; automatic fallbacks for browser support; seamless with FastAPI WebSockets backend. | http://Socket.io provides **reconnection logic**, **broadcast support**, and **fallbacks** out of the box, which are not available with the raw WebSocket API or SSE, making it more reliable for complex apps. |
| **SASS** | Less, Stylus, PostCSS | Extends CSS capabilities (variables, mixins, nesting); supports maintainable, scalable stylesheets; works well with Tailwind for customization. | SASS has **wider community adoption**, **better documentation**, and **stronger toolchain support** than Less or Stylus, and complements Tailwind for advanced theming and overrides. |

# 3. Backend Stack

| Technology | Alternatives | Pros. | Comparison |
| --- | --- | --- | --- |
| **FastAPI** | Flask, Django | FastAPI is an asynchronous, high-performance web framework that leverages Starlette and Pydantic for asyncio support, automatic OpenAPI documentation, and data validation. Its key benefits include:<br>• Asynchronous Support: Built for high concurrency and non-blocking I/O, ideal for real-time applications like WebSocket-based communication.<br>• High Performance: Optimized for speed, it handles requests faster than traditional frameworks like Flask or Django.<br>• Type Safety: Automatic validation with Pydantic ensures consistent and reliable data handling.<br>• OpenAPI & Documentation: Generates interactive API docs for quick testing and integration.<br>• Scalable & Modular: Ideal for microservices and supporting | Compared to **Flask** and **Django**, FastAPI is far superior in terms of **async/await support**, making it ideal for applications requiring **real-time** capabilities and **scalability**. While **Express.js** and **Sanic** are fast and lightweight, FastAPI offers superior **automatic documentation**, reducing the time spent on creating and maintaining API docs. Unlike Django, FastAPI is much **faster** due to its **async-first architecture**, making it ideal for applications with high throughput and low latency requirements. |

| | | | |
|---|---|---|---|
| | | features like OAuth and background tasks with minimal setup.<br><br>FastAPI is perfect for building high-performance, scalable enterprise applications, offering ease of use, speed, and developer-friendly tools. | |
| **SQLAlchemy** | cx_Oracle, Django ORM, Tortoise ORM | Mature Python ORM; clean separation of models and queries; native Oracle support; async capabilities improving rapidly. | Compared to Django ORM (tightly coupled with Django) and Tortoise (less mature), SQLAlchemy offers **flexibility**, **fine-grained control**, and **strong Oracle support**, making it suitable for complex enterprise use cases. |
| **Oracle 23AI Database** | PostgreSQL + pgvector, MySQL, MongoDB, Azure Cosmos DB | Combines traditional RDBMS with native vector storage; supports PL/SQL; ideal for secure, enterprise-grade AI-enhanced applications. | Oracle 23AI uniquely integrates **AI-native features (like vector indexing)** into a proven RDBMS, unlike PostgreSQL or Cosmos DB which require add-ons. It's **enterprise-hardened**, secure, and scalable. |
| **Redis** | Memcached, Kafka (for pub/sub), RabbitMQ | In-memory key-value store; ultra-fast caching; supports pub/sub and lightweight queuing; crucial for real-time updates and session storage. | Redis offers **multi-purpose support** (cache + pub/sub + streams), unlike Memcached (cache-only) or RabbitMQ/Kafka (heavier infra). It's lightweight yet powerful for **low-latency applications**. |
| **Python 3.10+** | Node.js, Go, Java, .NET | Modern syntax features (e.g., structural pattern matching); async/await support; rich AI/ML ecosystem; ideal for rapid prototyping and scalability. | Python balances **developer speed**, **rich AI libraries**, and **async capabilities**, unlike Java/.NET (more verbose) or Go (low-level), making it ideal for AI-driven backend services. |
| | pip + venv, | Modern dependency and package management; lockfiles | Poetry provides **dependency resolution + packaging** in |

| | | | |
|---|---|---|---|
| **Poetry** | Pipenv, Conda | ensure reproducible builds; better suited for enterprise and monorepo structures. | one tool, unlike pip + venv or Pipenv. It's more deterministic and CI/CD friendly than Conda for Python-only projects. |
| **RabbitMQ** | Kafka, Redis Streams, Amazon SQS, ActiveMQ | RabbitMQ is a reliable message broker designed for asynchronous communication and decoupling systems. Key benefits include:<br>• Reliability: Ensures message durability, preventing data loss even during failures.<br>• Flexible Routing: Supports advanced AMQP routing for various messaging patterns (e.g., publish/subscribe).<br>• Scalability: Scales horizontally to handle high-throughput, distributed architectures.<br>• Protocol Support: Offers compatibility with widely used AMQP for broad integration with other systems.<br>RabbitMQ is perfect for building scalable, decoupled systems that require efficient and reliable message processing. | Compared to Kafka, which excels in **high-throughput log streaming**, **RabbitMQ** is better suited for **task-based messaging** with complex routing needs. Unlike Redis Streams or SQS, RabbitMQ provides richer delivery guarantees (e.g., **acknowledgements, retries, dead-lettering**) and supports **pluggable exchanges** and **message filtering**. It also offers better **visibility into message queues** for debugging and monitoring, which is critical in microservices and enterprise applications. |

# 4. Generative AI Stack

| Technology | Alternatives | Pros. | Comparison |
|---|---|---|---|
| **LangGraph** | Haystack Agents, AutoGen, CrewAI | LangGraph enables construction of stateful, directed, multi-agent workflows using a DAG (Directed Acyclic Graph) architecture. Each node can represent an agent, tool, or decision step, and edges define transitions based on logic, memory, or external signals. It's particularly suitable for enterprise systems where complex decision flows, retries, and streaming outputs are needed. LangGraph's design promotes modular, testable, and observable pipelines, which is critical in production AI systems involving dynamic response routing and fallback strategies. | Compared to **Haystack Agents** and **AutoGen**, **LangGraph** provides better support for **stateful workflows** and **complex decision-making**, offering a clear advantage for enterprises dealing with complex interactions and the need for **reliable retry logic** and **streaming**. While **CrewAI** is also designed for multi-agent workflows, LangGraph's use of **DAG** and integration of modular and testable components gives it a more robust and flexible architecture for production-grade applications. |
| **Therix** | LangFuse, PromptLayer, WhyLabs, Arize AI | Therix specializes in observability and prompt management, providing robust tools for AI systems management.<br>• Observability: Offers real-time monitoring to track model performance, AI agent interactions, and system health, ensuring proactive issue resolution and continuous optimization.<br>• Prompt Management: Provides centralized tools to manage, test, and version AI prompts, making it easy to ensure consistency, scalability, and efficiency in prompt usage across multiple applications. | While tools like LangFuse and PromptLayer offer observability and prompt tracking, **Therix gives us full code-level control** since it is developed in-house. This allows deep customization tailored to our enterprise needs, seamless integration into our stack, and tighter data governance. Competing tools may offer similar features but are limited in extensibility, cost-effectiveness, or private hosting flexibility. Therix also aligns better with internal compliance and security requirements. |
| | | MCP (Model Context Protocol) is a specialized protocol designed to manage context during interactions with AI models. It enables persistent context storage, which allows the system to track user sessions, conversation history, and other dynamic context attributes that | Compared to **Custom API-based Context Management**, MCP offers more **standardized** and |

| | | | |
|---|---|---|---|
| **Model Context Protocol (MCP)** | Custom API-based Context Management, Memory Networks | influence AI outputs. MCP ensures that the model can refer back to previous interactions or system states, which is critical in multi-turn conversations or complex workflows. By using MCP, an AI model can be more context-aware, leading to more accurate and personalized responses. It also ensures scalability in multi-agent or multi-user environments by decoupling the AI model from direct session handling, allowing the system to scale across multiple parallel sessions while maintaining context integrity. MCP is vital in enterprise AI applications where state consistency and data privacy are paramount. It can be integrated with existing AI pipelines and can be used to update or retrieve context as needed, facilitating complex agent-based workflows and more robust decision-making in real-time. | **scalable** solutions by maintaining context in a structured and persistent manner. **Memory Networks** can also manage context, but they typically struggle with scalability and **real-time decision-making** in high-load environments. MCP is more efficient and resilient for **enterprise-scale AI systems**, supporting **complex workflows** with better **data privacy** and **session integrity**. |
| **AWS Bedrock** | Azure OpenAI, Google Vertex AI, Hugging Face Endpoints | AWS Bedrock provides access to multiple foundation models (Anthropic Claude, Mistral, Amazon Titan, Cohere, etc.) via a **unified API**, removing the need to manage infrastructure. Supports **RAG**, **Agents**, **Guardrails**, and **model chaining**. Integration with IAM, VPCs, Step Functions, and Lambda makes it ideal for building secure, scalable GenAI systems in regulated environments. | **AWS Bedrock** stands out due to its **wide range of model choices** and the ability to seamlessly integrate with AWS services like **IAM**, **VPCs**, and **Lambda**. Compared to **Google Vertex AI** and **Azure OpenAI**, AWS Bedrock simplifies model management with its **unified API** and robust security features, making it more suitable for organizations operating in regulated or enterprise environments. **Hugging Face Endpoints** also offers model hosting but lacks the same level of integrated security and service scalability offered by AWS Bedrock. |
| **Amazon SageMaker** | Vertex AI, Azure ML, | SageMaker enables end-to-end ML workflows including data preprocessing, model training, tuning, deployment, and monitoring. It supports secure custom LLM hosting with GPU acceleration, fine-tuning on domain-specific datasets, and scalable deployment via endpoints. Its | **SageMaker** provides a **comprehensive, fully-managed platform** for ML workflows, with the added advantage of deep integration with other AWS services like **S3**, **EventBridge**, and **Step Functions**. Compared to **Vertex AI** and **Azure ML**, SageMaker offers more advanced fine-tuning capabilities and a wider range |

| | | | |
|---|---|---|---|
| **SageMaker** | Databricks | datasets, and scalable deployment via endpoints. Its integration with S3, EventBridge, and Step Functions makes it well-suited for building robust enterprise-grade GenAI pipelines. | of deployment options, including custom LLM hosting with GPU acceleration. While **Databricks** excels in collaborative data science environments, SageMaker is a better fit for production-level GenAI systems that require a more integrated and scalable approach to deployment and monitoring. |

# 4. Generative AI Models

| Model Name | Alternatives | Pros. | Comparison |
|---|---|---|---|
| **Llama** | GPT-4, Claude 3 | Llama 3.3 70B is a multilingual, instruction-tuned, text-only large language model developed by Meta. It offers enhanced performance in reasoning, mathematics, general knowledge, instruction following, and tool use. Key features include:<br>• Multilingual Support: Natively supports English, German, French, Italian, Portuguese, Hindi, Spanish, and Thai.<br>• Extended Context Window: Capable of processing up to 128,000 tokens, facilitating long-form content generation and complex document analysis.<br>• Comparable Performance: Delivers performance on par with Llama 3.1 405B while requiring | Unlike GPT-4, Claude 3, or Gemini, **Llama 3.3 70B is open-access**, providing full control over deployment, fine-tuning, and on-prem hosting—crucial for compliance and cost control in enterprise environments. While closed models excel in raw |

| | | | |
|---|---|---|---|
| **3.3 70B** | Claude 3, Gemini 1.5, Falcon 180B | significantly fewer computational resources.<br>• Instruction Tuning: Fine-tuned using Supervised Fine-Tuning (SFT) and Reinforcement Learning with Human Feedback (RLHF) to align with human preferences for helpfulness and safety.<br>• Open Access: Available under the Llama 3.3 Community License Agreement, promoting transparency and accessibility for research and commercial use.<br>Use Cases: Ideal for applications requiring advanced reasoning, multilingual capabilities, and efficient processing, such as enterprise chatbots, content generation, and multilingual customer support systems. | performance, Llama 3.3 offers near-parity in benchmarks, with significantly reduced infrastructure requirements compared to Falcon 180B. Its native multilingual support and alignment training make it highly usable out-of-the-box for enterprise LLM applications. |