A modern platform for building GenAI applications at scale

The better the question. The better the answer.
The better the world works.

EY
Shape the future with confidence

Organizations can accelerate GenAI application development with a robust platform architecture built on scalability and collaboration.

Executive summary

01 Many organizations struggle with the complexities of developing GenAI applications at enterprise scale.

02 The platform architecture described here helps streamline the GenAI software development lifecycle by increasing standardization and improving efficiency.

03 This architecture establishes a future-ready foundation, with planned support for multimodal GenAI use cases and interoperability protocols.

Introduction

The advent of generative AI (GenAI) has ushered in a transformative shift across industries, pushing enterprises to rethink traditional application architectures, development methodologies and operational strategies. Promises for significant advancements in various fields, from enhancing productivity and creativity to driving innovation and shaping the future of work, are rooted in GenAI's ability to generate new content, discern underlying patterns and help automate tasks. As organizations explore the adoption of GenAI to drive competitive advantage, there is a pressing need for platforms that are not only scalable and resilient but also flexible enough to adapt to diverse use case requirements across sectors.

Despite the rapid evolution of GenAI capabilities, many enterprises struggle with the complexities of inconsistent prototyping methods, tight coupling of components, hyperscaler lock-in, and the absence of configuration-driven approaches while building applications across their use case portfolios. Furthermore, successful GenAI implementation demands integration with a wide variety of backing services, such as large language models (LLMs), vector data stores, machine learning (ML) inference endpoints and technical security controls. In today's fast-evolving technology landscape, integrating with these backing services and seamlessly adopting fit-for-purpose engineering techniques and implementation styles for specialized tasks are key architectural imperatives.

This article shares the framework for a cloud-native-yet-hyperscaler-agnostic enterprise GenAI platform that closes these gaps. The framework is designed with modularity and reusability in mind to accommodate the full spectrum of AI-native and enterprise-grade requirements using a layered architecture built on granular, service-oriented components. By decoupling the capabilities required for building GenAI applications, the framework enables seamless orchestration of AI-native tasks, while providing optionality for application developers to extend platform defaults. We will articulate the guiding principles, reference architecture and key components, design considerations, and common archetypes that make the platform a more robust foundation for building enterprise-scale GenAI applications compared to point solutions. We will also share how a well-architected GenAI platform can bridge the gap between experimentation and production, accelerate time-to-value, and support continuous innovation through secure, scalable and maintainable solutions.

Five guiding principles for a GenAI platform

1. Improved developer efficiency and faster time to production

The platform for implementing GenAI use cases should offer a comprehensive set of configuration-driven capabilities that enable users to define content ingestion and user request execution pipeline logic, allowing use case developers to focus solely on business logic and necessary customizations. This flexibility allows for rapid experimentation and iteration, significantly accelerating the overall development lifecycle of GenAI applications.

At the core of the platform should be a robust framework that provides ready-to-use components designed to run ingestion and execution pipelines, uniformly handle security and necessary compliance protocols, and integrate seamlessly with a variety of backing services, such as vector stores, language models and ML endpoints.

Use case-specific applications should be generated from a boilerplate codebase, which leverages reusable configuration profiles, pre-built scaffold code with clearly marked placeholders for custom business logic, deployment artifacts and infrastructure templates that have been optimized for scalability and efficiency. This promotes a smooth transition from prototyping to production-grade deployment.

These built-in capabilities not only streamline the development process but also reduce the effort and time required to build, test and deploy GenAI applications, allowing teams to focus on innovation and less on infrastructure and application coding overhead.

2. Consistency

A well-defined framework establishes a set of conventions and leading practices that enforce consistency across both the application codebase and the supporting infrastructure. This standardized approach minimizes the likelihood of developer-induced errors by reducing ambiguity and enforcing predictable patterns in how components are built, integrated and deployed.

By externalizing most of the application execution through configurations, the framework simplifies many of the underlying complexities. It handles tasks such as configuration parsing, management of input and output payload, intermediary data processing, and orchestration of task execution. Configuration also governs fine-tuning parameters for LLMs (e.g., temperature, top-k, top-p, max tokens, retry configurations), metadata mapping for vector data stores, labels and indexing strategies in graph data stores. This allows use case developers to focus their efforts on implementing custom business logic within clearly defined placeholders, significantly streamlining the development process.

The resulting consistency not only improves code maintainability but also makes the codebase easier to understand and leverage, particularly for new developers or those transitioning between projects. Clear structure and uniform patterns reduce the learning curve and make onboarding faster and more efficient.

Moreover, this approach fosters better collaboration among team members. With developers adhering to a common set of guidelines and architectural standards, teams can work more effectively together, reduce miscommunication and maintain alignment throughout the software development lifecycle (SDLC).

3. Scalability

A framework-based approach confirms that applications are built on solid architectural foundations, incorporating design patterns and open principles that inherently support scalability and maintainability. This approach provides a consistent structure and enforces leading practices, allowing applications to grow in complexity and scale without compromising stability or performance.

The framework introduces a modular architecture that enables developers to add or remove features independently, without impacting the integrity of the broader application. It abstracts complex operational flows and provides built-in capabilities for managing

the application lifecycle, executing context-aware multi-threading and multi-processing as necessary, and handling cross-cutting concerns such as uniform logging with traceability, caching, exception handling and automated retry mechanisms.

Boilerplate applications should be pre-packaged with scaffold code and deployment artifacts optimized for lean performance resulting in lightweight containers that can be deployed on thin virtual machines (VMs) behind secured load balancers. Deployments should also support horizontal autoscaling with a wide range of configurable scaling policies, enabling cost-effective and resilient infrastructure management.

To further enhance modularity and scalability, complex or resource-intensive tasks are decoupled from the core application and exposed as separate microservices. These services are deployed on dedicated infrastructure and accessed through RESTful endpoints, adhering to microservice architecture principles. This separation of concerns not only improves system resilience but also simplifies the development and maintenance of individual components.

4. Enhanced security and compliance

The platform should be capable of integrating with any backing service that conforms to established application programming interface (API) specifications, including those meant for specialized security and compliance enforcement. This standardized interface supports consistent integration and interoperability, while allowing the services to evolve independently.

While the platform may provide default implementations of security and logging services, developers should not be limited to using them. Use case developers should have the flexibility to substitute or extend these default implementations with their own solutions, leveraging approved libraries, tools and security providers. This extensibility empowers teams to meet domain-specific compliance and security requirements without being constrained by platform defaults.

To provide this flexibility, the platform should support creation and use of configurable profiles for defining security guardrails and compliance measures. These configuration blocks include input and output scanning rules, parameters for content obfuscation, and settings for generating logging. Security and responsible AI services typically use these profiles to perform critical validations, sanitize LLM prompts and outputs, and monitor adherence to ethical AI standards, all of which are accomplished in a centralized and automated manner.

5. Better multidisciplinary collaboration

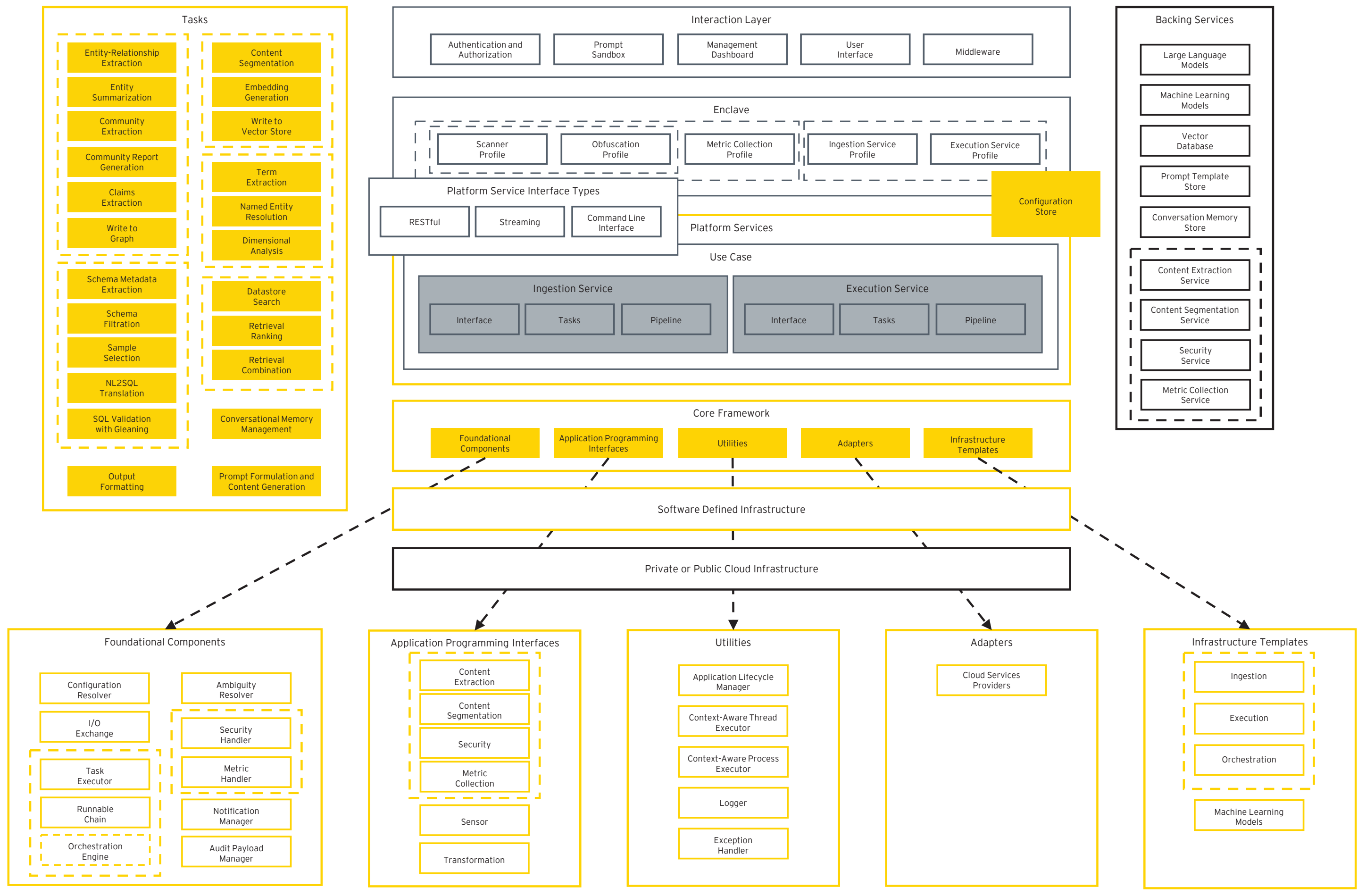
Delivery of production-grade GenAI applications typically requires a multidisciplinary team with a variety of specialized skill sets. The platform must provide isolated, role-based workspaces within its ecosystem, allowing team members to work independently while still facilitating collaboration when necessary. This separation of concerns enhances productivity by allowing contributors to focus on their respective domains without unnecessary implementation burdens.

For example, the platform should offer data scientists an isolated and purpose-built workspace for conducting training, experimentation, fine-tuning, and deployment of LLM and other ML models. Prompt engineers and business analysts should be able to create and pressure-test prompts separate from core application code to streamline iteration and reduce development overhead. Finally, application developers should be able to implement core business logic against the application's configuration without being burdened by managing prompt content.

Overall, this approach not only optimizes the technical processes involved in GenAI use case development but also fosters a collaborative and scalable culture. It enables clear separation of roles, better maintainability and faster iteration cycles, ultimately driving greater efficiency and innovation across the organization.

Reference architecture

For a clearer understanding of the platform's modular design and extensibility, the following diagram illustrates the various layers and the functional segregation of platform components. It also provides a visual representation of the core constructs and key conceptual elements that will be defined thereafter.



Tasks

Implementing GenAI use cases involves deploying tasks, which can be encapsulated as two distinct service classes: one for content ingestion from various sources and another for user request execution that results in AI-assisted response generation. Because the open-source community and commercial vendors offer many options to implement these tasks, task behavior should be tailored through platform-driven configurations, enabling dynamic orchestration and adaptability for complex activities. Further, some tasks may be optional for certain types of use cases, while others are necessary, and the strategies to implement these tasks are varied. For richer customization beyond default strategies, use case developers can deploy their own logic, either by deploying services that align with platform-governed API specifications or by extending the application using built-in code hooks. This flexibility makes each task reusable and adaptable to business needs.

Task orchestration in a content ingestion pipeline allows definition of data processing patterns (e.g., batch, micro-batch), content extraction protocols, transformation sequences including content segmentation, term extraction, named entity resolution, entity relationship extraction, vector embedding generation, and content persistence. For specialized use cases, tasks may also include analytical operations like community detection from extracted entities to be persisted in graph data stores.

Core framework

The platform's core framework provides a set of reusable code artifacts that facilitate task orchestration for executing both ingestion and execution pipelines. It is structured and organized into distinct segments, each addressing specific concerns to ensure separation of responsibilities:

- Foundational components form the core building blocks, driving a wide range of logic, including configuration parsing, standardized data exchange model definition, task pipeline orchestration and establishment of conversational hooks to resolve ambiguities in user interactions.
- The APIs provide a set of well-governed specifications for various tasks, allowing use case teams to build custom implementations. The platform offers a consistent invocation mechanism, streamlining integration with custom tasks while enforcing platform-governed specifications to ensure seamless interoperability and predictable behavior across services.
- The utils (short for utilities) module serves as the operational toolkit of the platform, offering plug-and-play support for core observability features like structured logging, distributed tracing and lifecycle hooks.
- The adapters layer enables hyperscaler-agnostic functionality by providing proxy implementations for integrating with various backing services.
- Infrastructure templates define reusable software-defined infrastructure modules for deploying consistent infrastructure components, including interfaces for ML model deployment.

Platform services

The platform services consist of a collection of one or more microservices designed to support specific business use cases. These services offer multiple integration points through RESTful or streaming endpoints, command-line interfaces and function-as-a-service models. Ingestion services define the pipelines for ingesting structured or unstructured data from various sources, applying a sequence of one or more transformations, and writing the processed data into one or more data stores. Execution services interpret user requests and manage the orchestration pipelines for data retrieval from various sources, refining the search results when necessary, enrichment using appropriate ML algorithms and LLMs, and producing a structured response for consumption by downstream systems. Both ingestion and execution services leverage platform configurations and the reusable framework components to orchestrate their respective pipelines effectively.

Interaction layer

Client-specific customizations can be implemented within the interaction layer, which includes a user interface and a middleware application acting as a proxy to the platform's core ingestion and execution services. This layer allows for the incorporation of use case-specific validation rules, authentication and authorization mechanisms, and management dashboards. Additionally, a dedicated workspace for prompt engineering enables business analysts, in collaboration with prompt engineers, to experiment with and evaluate various prompting techniques tailored to the client domain.

Backing services

The Twelve-Factor App methodology defines a backing service as any service an application consumes over the network as part of its normal operation. In the context of the platform, LLMs, ML inference endpoints, various data stores (e.g., vector, graph, relational, NoSQL), prompt template stores and conversational memory systems constitute the key backing services. The framework provides a configuration-driven integration interface and modular adapters for each service, offering plug-and-play flexibility. This architecture allows teams to easily integrate, replace or extend components to support a wide variety of use cases with minimal overhead.

Enclave and profiles

To meet both diverse and specific business requirements the platform allows the stakeholders to choose from a variety of options and implementation strategies to optimize outcomes via parametric levers. To organize and replicate the choices made at the appropriate level of isolation, such as across the enterprise, within a project or for a given use case archetype, the platform supports the concept of enclaves and creation of profiles within the enclaves. Profiles offer multiple configuration options and support user choices, and they may be defined for ingestion and execution services, security scanners, obfuscation routines and compliance metric collection. Profiles operate within isolated enclaves, so that constraints, preferences and policies are enforced and maintained at the right level. Parameters collected within the profiles are stored in the same store as all the other configurable parameters for the use case.

Examples of common task pipeline archetypes

The following examples illustrate typical task orchestration pipelines designed for content ingestion and execution services for common GenAI use case archetypes. These pipelines leverage foundational enterprise integration patterns, including routing, message translation, aggregation and wiretap, to orchestrate and manage GenAI workloads with precision. Seamless integration with a diverse set of backing services is achieved through a standardized invocation adapter, enabling consistent communication across systems. Built on the principles of hexagonal architecture, this integration platform offers a robust and scalable foundation that seamlessly unifies GenAI and non-GenAI workloads with agility and modularity.

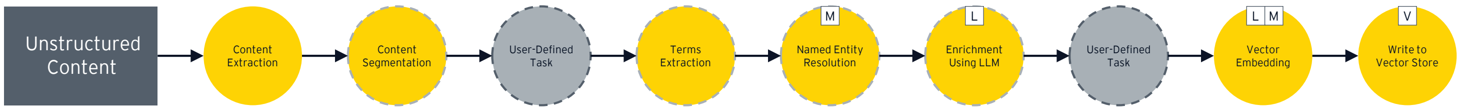
All figures below follow a standardized notation. Each task is represented as a circle, with background colors indicating responsibility for execution. Tasks in yellow are primarily governed by platform configuration and core framework components, except for content extraction and segmentation tasks specified within the ingestion pipeline, which can also be replaced by user-defined customized tasks.

Use case developers have limited control over these tasks, typically restricted to modifying their input and output payloads. In contrast, tasks in gray represent user-defined functions/tasks (UDFs), where developers have full control over the execution logic. Tasks outlined with a dotted line indicate optional execution paths. Letters in squares within a task represent integrations with backing services, with the following abbreviations:

- L = LLM
- M = ML model inference endpoint
- V = vector data store
- P = prompt template storage
- G = graph data store
- R = relational database
- N = NoSQL database
- DS = protocol for arbitrary data store (e.g., REST, SOAP, GraphQL)

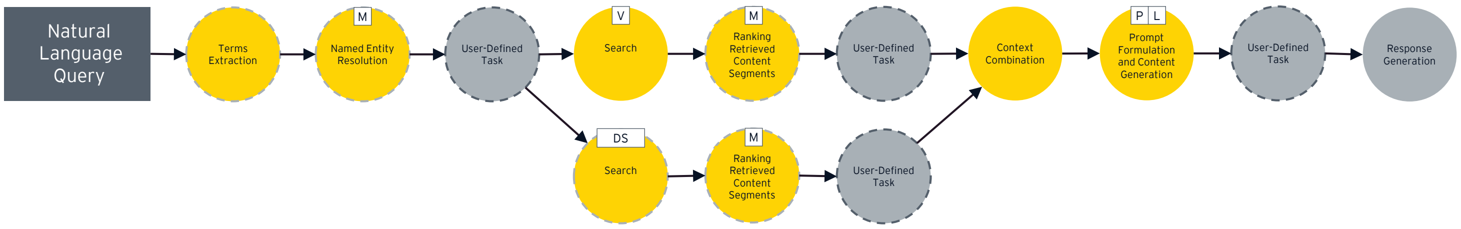
Retrieval-augmented generation (RAG)

The figure below illustrates a pipeline for ingesting unstructured content into a vector data store:



The pipeline begins with the extraction of unstructured content from source systems, followed by a series of transformation steps, including content segmentation, term extraction, named entity resolution and enrichment using an LLM. Use case developers can insert any number of user-defined tasks within this transformation sequence to tailor it to the specific needs of the use case. Vector embeddings are then generated through integration with either LLM or ML inference endpoints. Finally, the transformed content is written into a vector data store.

RAG is a mechanism often used to increase the acuity and domain specificity of responses in GenAI applications. The following diagram illustrates a typical RAG pipeline involving one or more data stores, including vector data stores:



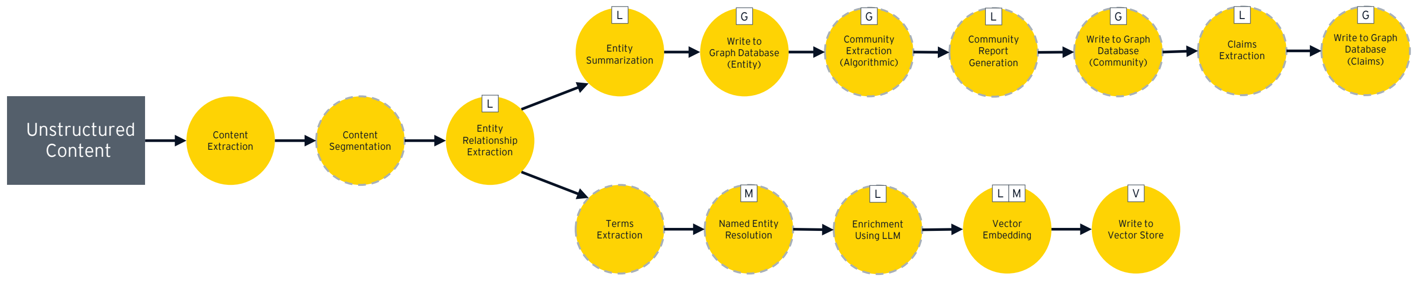
The pipeline begins with processing the user request through steps such as term extraction, named entity resolution and any number of user-defined tasks. This enriched information is then used to retrieve semantically similar unstructured content from vector data store and/or structured or unstructured content from other data stores, enhancing the search process according to the search scope defined in the configuration.

The retrieved content can be refined using ML-powered re-ranking algorithms, along with any number of additional user-defined tasks. The refined results are then combined using the configured strategy, embedded into a prompt template and submitted to an LLM to generate high-quality responses.

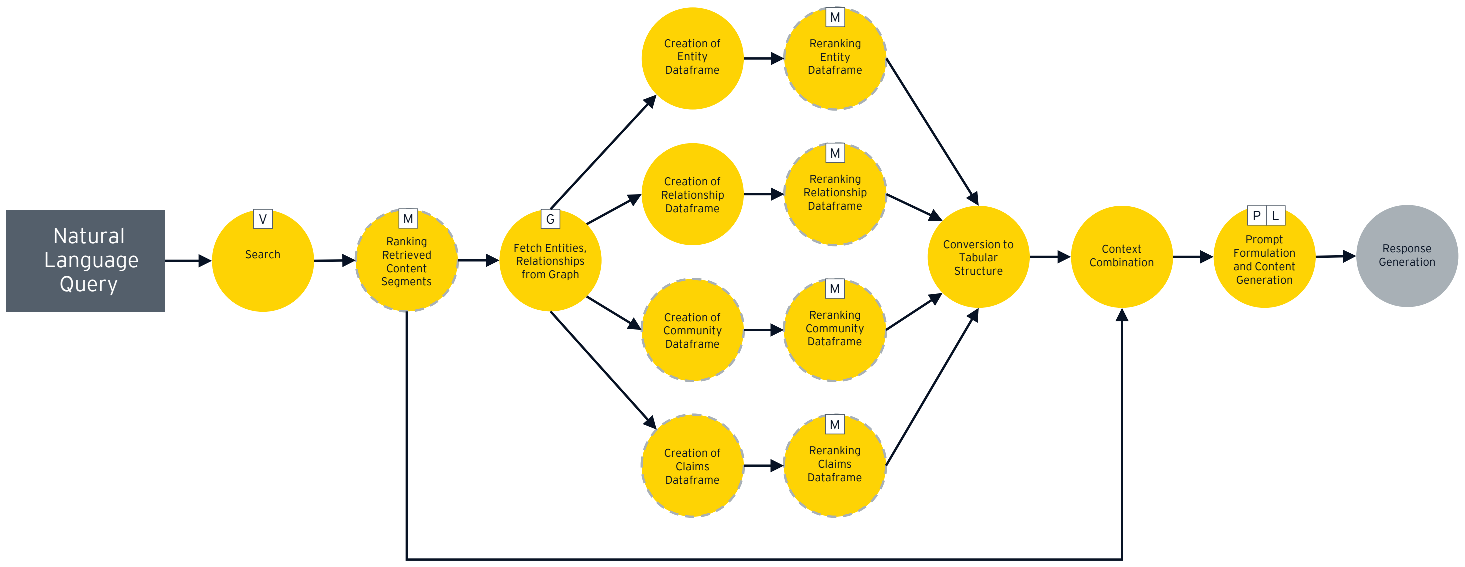
Following this, the LLM-generated output can be further processed through UDFs before returning to upstream systems. Throughout the pipeline, developers have the flexibility to define both fixed and conditional routing sequences, enabling dynamic and customizable flow control tailored to specific use cases.

RAG with vector and graph data stores

This pipeline extends the ingestion and execution flows described above by incorporating the power of graph data stores:



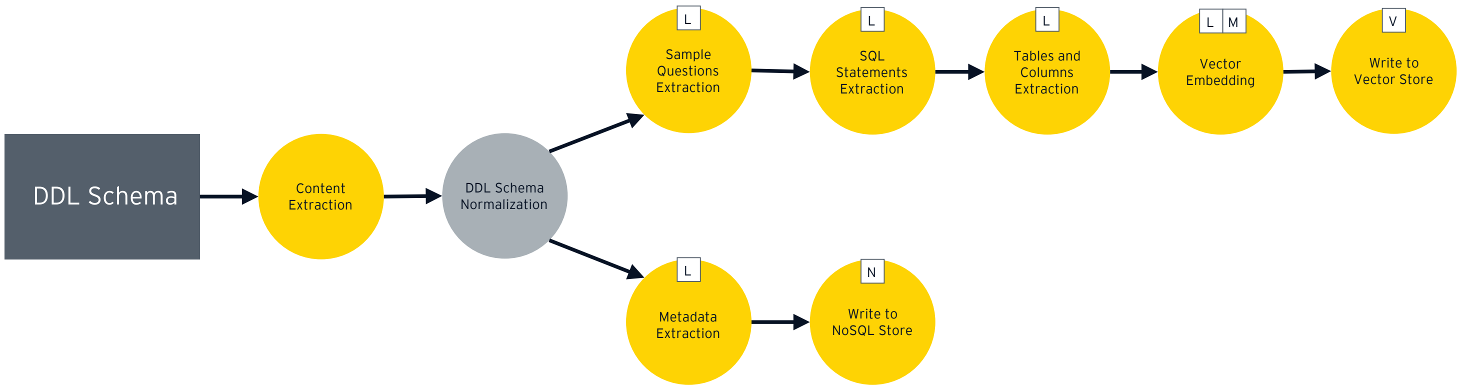
In addition to the usual RAG processing steps, the ingestion pipeline extracts entity-relationship data from unstructured content and writes it in a graph data store. These structured relationships can then be leveraged to uncover entity clusters or communities, generate community-level summaries, extract claims in relation to specific statements, and retain those claims for future use. This process effectively adds a structured layer of meaning to previously unstructured data, enhancing its analytical value.


Natural Language Query
V
Search
M
Ranking Retrieved Content Segments
G
Fetch Entities, Relationships from Graph
Creation of Entity Dataframe
M
Reranking Entity Dataframe
Creation of Relationship Dataframe
M
Reranking Relationship Dataframe
Creation of Community Dataframe
M
Reranking Community Dataframe
Creation of Claims Dataframe
M
Reranking Claims Dataframe
Conversion to Tabular Structure
Context Combination
P L
Prompt Formulation and Content Generation
Response Generation

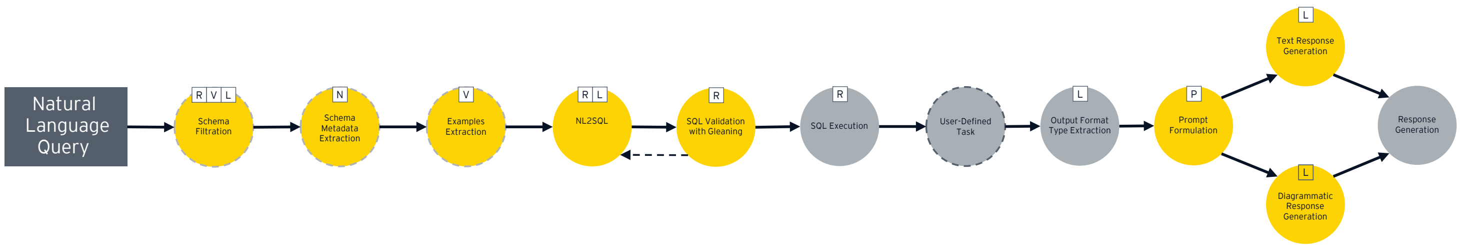
The execution pipeline above builds on the standard RAG flow by enriching the context for an LLM with entity relationships, community insights and extracted claims. This added context offers a more holistic and interconnected perspective, empowering the language model to generate deeper, more informed responses.

Natural language to structured query language (SQL)

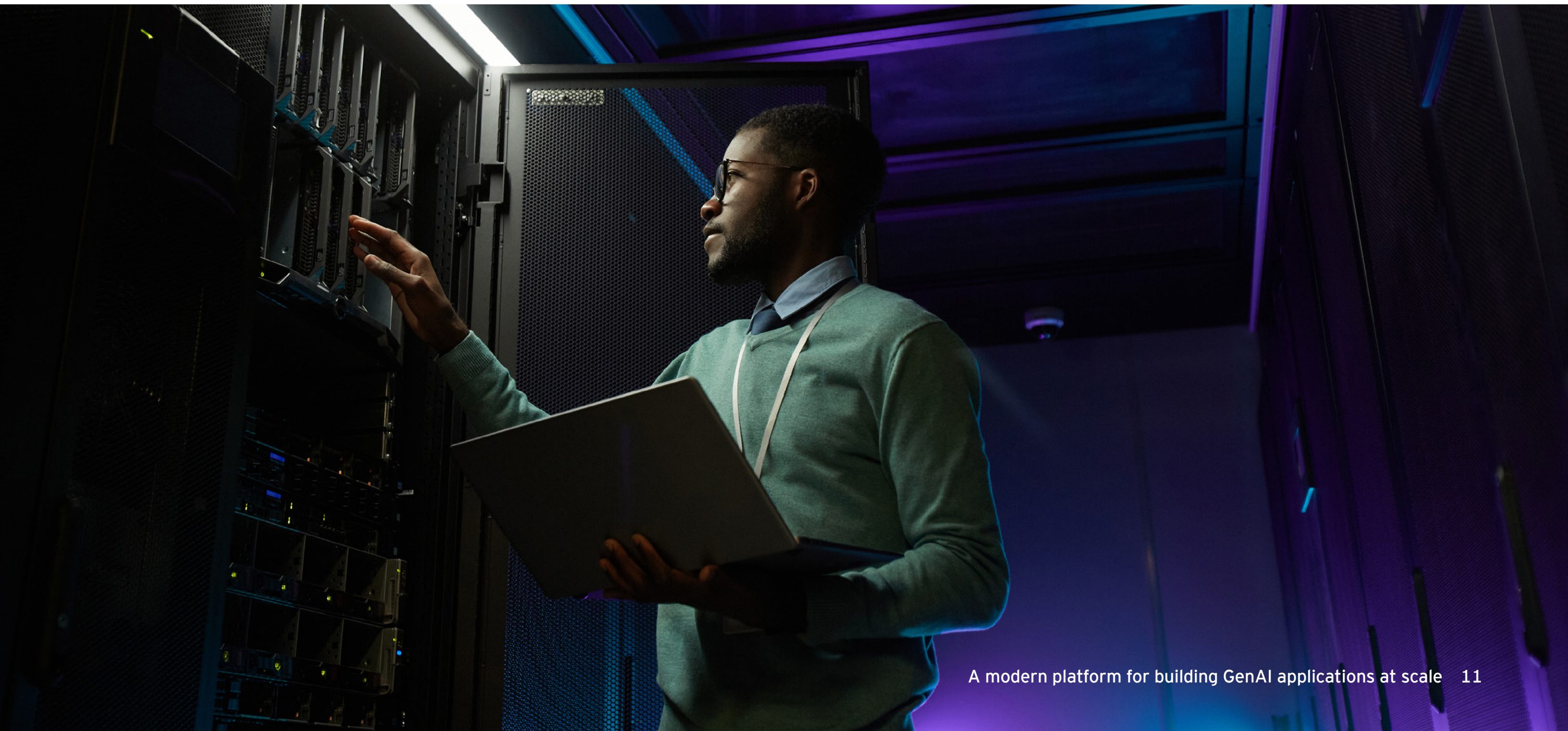
The following figures illustrate how ingestion and execution pipelines can be configured and used to generate structured queries from natural language inputs. While the example focuses on translating natural language into SQL statements, similar pipelines can be adapted to generate queries in other formats, such as Cypher and SPARQL.



The ingestion pipeline showcases how a standard data definition language (DDL) schema can be used to automatically generate sample questions, SQL statements, and metadata about tables and columns. This preprocessed data plays a crucial role in the execution pipeline, serving as the foundation to streamline and optimize the natural language to SQL transformation process.



The execution pipeline highlights the orchestration of tasks involved in converting natural language queries into valid SQL statements. It leverages the pre-ingested samples for schema filtering, metadata enrichment and example selection, all of which are critical steps when working with databases that contain large or complex schemas.



Benefits

GenAI applications are susceptible to producing inaccurate responses, exhibiting misplaced confidence, following flawed logic or delivering incomplete answers. Supporting multiple information retrieval techniques (e.g., kNN, TF-IDF, BM25) is possible within this configuration-driven framework, and the platform can effectively adapt to a wide variety of use cases and data structures. By retrieving examples that are closely aligned with the user's query, the system can provide more relevant and grounded responses, particularly in use cases requiring nuanced or domain-specific knowledge.

Generating structured output for external consumption is also a critical component of any GenAI application. However, even with a zero-temperature setting and well-crafted prompt instructions, the output can vary across iterations. This inconsistency can lead to a negative user experience, especially when results are not reproducible. To enforce reliable and structured output from GenAI applications, the platform facilitates the ability to declaratively define the expected output schema (e.g., JSON), which helps drive consistency, validate response structure and facilitate downstream processing. This approach allows developers to clearly specify the required fields, data types and nesting to align the application's expectations and the model's output.

Effectively managing conversational memory and minimizing hallucinations remains a significant challenge in the development of GenAI applications. Various memory persistence strategies, such as windowed buffering and summarization, can be employed in the platform to manage the evolving context within a conversation. To enhance the system's understanding of user intent, the platform also allows interceptors to be embedded within the conversation lifecycle. These interceptors monitor ongoing interactions to detect potential ambiguities between the user's current input and the existing conversation history. By identifying unclear or conflicting information early, the system can proactively address misunderstandings and maintain the quality of the conversation.

Finally, GenAI applications are expected to operate with high performance. However, these applications typically involve onerous dependencies and graphics processing unit (GPU)-accelerated workloads, which contribute to significant container sizes and prolonged initialization times. As a result, autoscaling introduces extended cold-start durations, adversely affecting user experience by increasing latency and limiting throughput. The platform leverages a microservices-based architecture that keeps containers lightweight and allows applications to be deployed either as serverless functions or as containers within VMs for optimal resource allocation. ML models can be deployed on GPU-enabled instances that are isolated within dedicated autoscaling groups, and these models can then be accessed via RESTful interfaces or inference endpoints. This allows the core application to remain decoupled from heavy computational workloads while still benefiting from GPU acceleration.

Opportunities to evolve

To promote seamless integration within diverse enterprise ecosystems, the platform roadmap includes planned support for following areas:

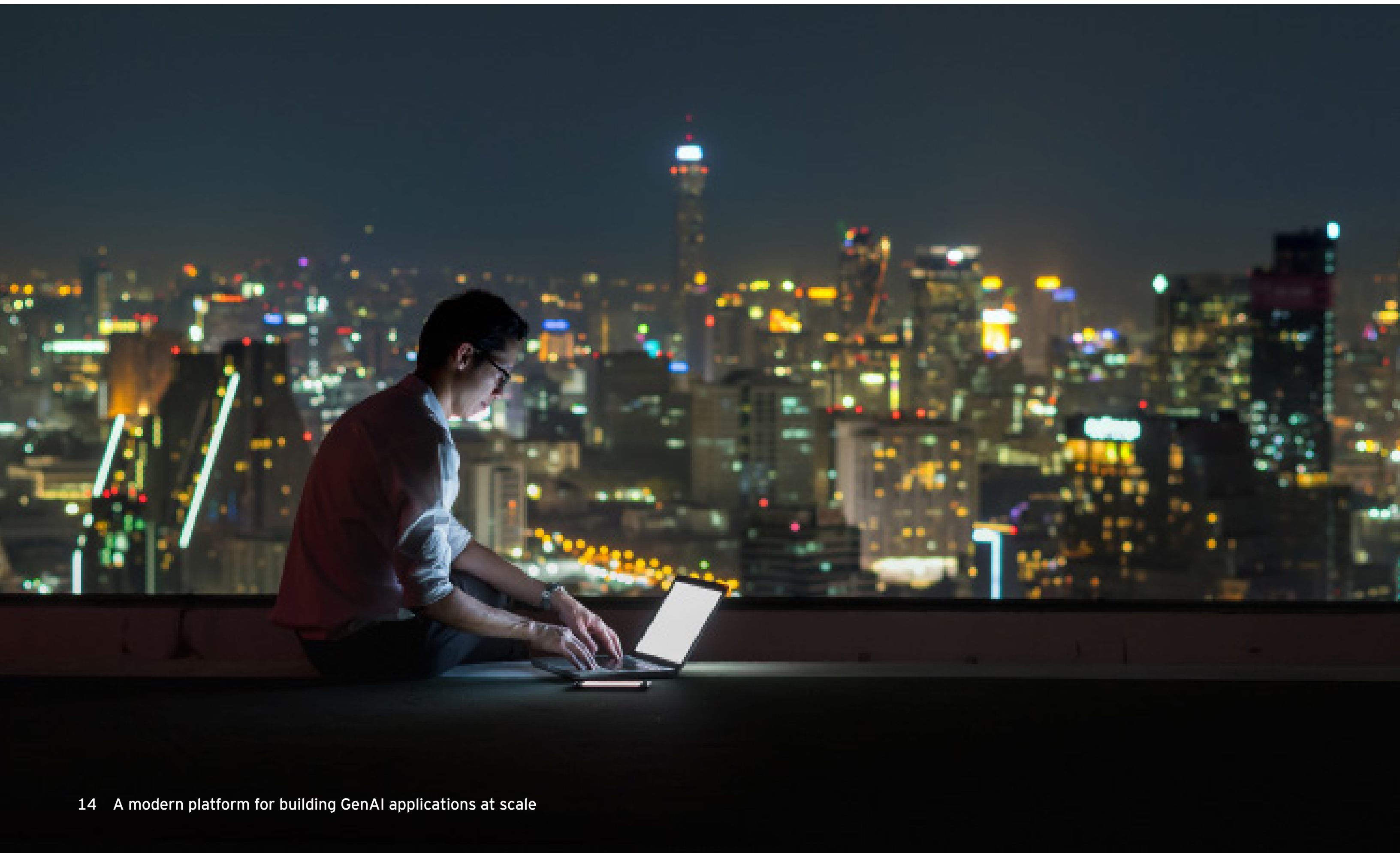
- The microservice-based ingestion and execution services form the backbone for enabling large-scale GenAI use cases through declarative service orchestration, LLM-driven multi-agent workflows and a hybrid (simultaneously agentic and declarative) standards-based approach.
- The pipeline domain-specific language can be extended to support multimodal GenAI use cases involving image, audio, video and 3D MIME types.
- Enhancing the agentic architecture with program-aided language prompting techniques enables automated code generation and execution.
- A visual design studio can empower nontechnical users to configure and generate ingestion, execution, and declarative or agentic task orchestration pipelines with minimal effort.
- Support for standardized interfaces, such as Model Context Protocol and Agent-to-Agent frameworks, offering structured and interoperable mechanisms for integrating with external systems and services.

Together, these innovation areas reinforce the GenAI platform's mission to remain a modular future-ready foundation for scalable domain-aligned AI adoption across the enterprise.

Conclusion and call to action

This article highlighted how a modern GenAI platform can accelerate GenAI application development efforts using established industry patterns, such as configuration-first design, reusable security and responsible AI profiles, enterprise-grade orchestration, contract-first interfaces, and scalable software-defined modular infrastructure. This platform is built on a modular hyperscaler-agnostic architecture that implements common GenAI patterns using fit-for-purpose reusable components. Further, its architecture allows use case teams to retain full control over the implementation of security validations and the collection of metrics, enabling compliance with organization-specific governance and regulatory requirements.

The proliferation of numerous point solutions to solve similar problems can lead to substantial technical debt from redundancy in the software development ecosystem and lack of development standards. More cybersecurity threats could surface as well. To scale the development of multiple GenAI applications that serve a breadth of business use cases, large organizations can adopt strategies such as the platform described here to protect the return on their GenAI investments and better position themselves to capitalize on technological advances in this domain.



Authors

Dipanjan Sengupta
Principal
EY GDS Consulting Distinguished Technologist and AI Engineering Leader
Ernst & Young LLP
dipanjan.sengupta@gds.ey.com

Kevin A. Tollison
Partner
Technology Consulting
Ernst & Young LLP
kevin.tollison@ey.com

William Carson
Senior Manager
Technology Consulting
Ernst & Young LLP
william.carson@ey.com