

Foundational Design Principles and Patterns for Building Robust and Adaptive GenAI-Native Systems

Frederik Vandeputte

Frederik.Vandeputte@nokia-bell-labs.com

Nokia Bell Labs

Belgium

Abstract

Generative AI (GenAI) has emerged as a transformative technology, demonstrating remarkable capabilities across diverse application domains. However, GenAI faces several major challenges in developing reliable and efficient GenAI-empowered systems due to its unpredictability and inefficiency. This paper advocates for a paradigm shift: future GenAI-native systems should integrate GenAI’s cognitive capabilities with traditional software engineering principles to create robust, adaptive, and efficient systems.

We introduce foundational GenAI-native design principles centered around five key pillars—reliability, excellence, evolvability, self-reliance, and assurance—and propose architectural patterns such as GenAI-native cells, organic substrates, and programmable routers to guide the creation of resilient and self-evolving systems. Additionally, we outline the key ingredients of a GenAI-native software stack and discuss the impact of these systems from technical, user adoption, economic, and legal perspectives, underscoring the need for further validation and experimentation. Our work aims to inspire future research and encourage relevant communities to implement and refine this conceptual framework.

Keywords: GenAI, software systems, reliability, excellence, design principles, architectural patterns, best practices

1 Introduction

In recent years, Generative AI (GenAI) has demonstrated remarkable emergent capabilities across a diverse array of application domains and use cases. GenAI distinguishes itself from traditional algorithms and AI/ML models through its ability to flexibly adopt and apply custom domain knowledge, skills, and reasoning strategies to solve new tasks with minimal additional effort from developers or end users. As a result, GenAI is emerging as a versatile and adaptive technology that will profoundly impact future technology stacks and revolutionize our way of working and creativity.

However, GenAI has several key drawbacks compared to traditional algorithmic processing, most notably its unreliability: it is prone to hallucinate despite countermeasures [24, 65], it can be quite unpredictable [5, 6, 63], and it has limitations in acquiring new knowledge and skills

through prompt engineering [25, 49, 66]. Other key drawbacks of GenAI, compared to traditional processing, are its large runtime overhead, footprint, and limited debuggability.

In recent years, several techniques have been developed to help overcome and mitigate the impact of these limitations. These include advanced retrieval augmented generation (RAG) based techniques [15, 30] and collaborative multi-agent systems (MAS) [19–21, 32] to enhance the knowledge and capabilities of AI agents [51], reasoning and self-reflection [53] capabilities to verify and improve their responses generated through chain-of-thought (CoT) [60] and reasoning [18, 43], interpretability and explainability techniques [34, 67] to measure overall quality and uncertainty, reinforcement learning techniques to systematically improve overall end-to-end effectiveness [7, 8, 23], etc.

Despite these innovations, GenAI, as cognitive processing technology, will always exhibit some degree of unpredictability, not only due to its architecture—autoregressive, diffusion or neuro-symbolic—but also the inherently dynamic and often under-specified nature of inputs and tasks. Moreover, GenAI can be very inefficient compared to traditional algorithms (e.g., calculating 1+1), particularly when utilizing advanced prompting or reasoning techniques.

So instead of concentrating solely on enhancing GenAI technologies, we propose embracing their unpredictability. This unpredictability is a core characteristic of GenAI, allowing it to adaptively generate new, albeit a untested, solutions to both existing and emerging challenges. Furthermore, we advocate for integrating GenAI approaches with established traditional Software Engineering (SE) methods to develop a balanced, synergetic solution. This may be achieved by designing a self-improving system where GenAI agents systematically automate themselves out of common critical paths.

This requires rethinking and expanding upon the existing software design and development paradigms. By embracing and leveraging their inherent unpredictability and adaptability in addressing new problems, while simultaneously striving for operational efficiency, we can achieve a dual focus on managing uncertainty and enhancing performance. This approach will facilitate the creation of more resilient, adaptive and efficient *GenAI-native systems*.

This paper advocates for a paradigm shift in the development of GenAI-based systems, embracing their limitations while enhancing them with traditional paradigms. Through

selected use cases, and by drawing insights from historical insights and human methodologies, we introduce a comprehensive conceptual framework of GenAI-native design principles, best practices, and architectural patterns. These include the GenAI-native cell, programmable router, unified conversational interface, and organic substrate. We explore the impact of GenAI-native systems on the future software stacks, and highlight their implications across technological, user, economic, and legal dimensions. Our work aims to redefine the landscape of future software systems, offering a comprehensive blueprint for creating resilient, adaptive, and efficient GenAI-native systems, and setting the stage for future research and innovation in this domain.

2 Motivation

In this paper, we present a vision for a GenAI-native system as a paradigm that leverages the cognitive capabilities and acknowledges the limitations of GenAI, while seamlessly integrating them with the operational excellence, reliability, and dependability of traditional SE paradigms. Although existing software engineering paradigms provide a solid foundation, we argue that they are inadequate for developing robust and adaptive GenAI-native software systems. On the other hand, *GenAI-first (agentic) systems*, where AI agents manage most critical actions, often lack the robustness and operational efficiency of traditional systems. With *GenAI-native systems*, we aim to combine the strengths of both approaches.

Existing software engineering paradigms focus on designing robust applications with rigid data structures, APIs, and user interfaces. This often results in tightly integrated micro-service architectures. Moreover, in traditional approaches, including low-code and no-code programming, flexibility within functional components is typically preconfigured. In contrast, GenAI offers a more flexible approach, driven by data and reasoning-based methods, potentially using natural language or pseudocode instructions.

As we will discuss later, cloud-native principles like immutable infrastructures, CI/CD, and version control need to be further extended and expanded to accommodate to the organic and self-improving nature of GenAI. GenAI allows applications to change its functionality on the fly by integrating custom-generated code, blurring the lines between development and deployment. However, it is crucial to preserve the reproducibility of such assets, and implement restrictions to maintain their original intent, preventing unintended evolution.

The importance of core software engineering principles such as automated testing, monitoring, security, and operational efficiency will only increase and must be further enhanced. GenAI-first agentic approaches often lack robustness and operational efficiency, facing challenges like erroneous outputs, unpredictable inconsistencies, and variability

in generated responses. Despite efforts to mitigate these issues, the creative nature of GenAI suggests these challenges will persist, especially beyond controlled tests. In addition, operational efficiency will be a key concern, as GenAI-first approaches typically incur higher operational costs with higher processing latency compared to traditional software solutions, mostly due to the complexity of using large models, retrieval augmentation methods, or extended reasoning.

Finally, when developing GenAI-native systems, it is crucial to avoid anthropomorphic pitfalls. While evaluating agents' effectiveness in mimicking human behavior is useful [36, 38, 45], these native digital cognitive entities do not require human-oriented interfaces for interacting with web services. GenAI-native systems should be reimaged to allow agents or other GenAI systems to directly communicate with them in a flexible, efficient, and reliable manner.

3 Related Work

While current GenAI-first agentic solutions somewhat reflect the old artisanal methods reminiscent of the early pre-industrial era, we envision a GenAI-native industrial future, where self-reliant multi-agent systems collectively and continuously strive to automate themselves out of the critical path of any solution, yet remain omnipresent to monitor, optimize and help create bespoke solutions.

There is recent work to enhance the robustness and operational efficiency of GenAI-based solutions, which underscore some of the key messages presented in this paper. Code-generating agents generate and execute code snippets instead of using lengthy verbal chain-of-thoughts [60] (e.g., PAL [14], PAR [27]) or triggering and handling many individual tool calls (e.g. CodeAct [59]). Ideally, these generated code snippets should be curated, battle tested, and stored as robust, reusable and efficient solutions for specific inputs (cfr. Dynasaur [42]), instead of always requiring agents to reinvent the wheel and regenerate very similar untested solutions.

The Agora protocol [35] is designed to optimize agent-to-agent (A2A) communication [1] by minimizing unstructured or ambiguous natural language exchanges between frequently interacting agents. It facilitates the implementation of a custom (REST) protocol, enabling agents to generate a functional code implementation and communicate effectively. This approach significantly reduces token usage and processing latency, leading to robust, repeatable, and efficient interactions. Moreover, the protocol maintains full flexibility, allowing agents to renegotiate or develop additional protocols for diverse scenarios. In addition, the model context protocol (MCP) [3] facilitates seamless integration between large language model (LLM) applications and external resources and tools.

In addition, there exists prior work regarding initial best practices and patterns for building GenAI based applications. In [13], several established lower-level design patterns are

listed to improve and evaluate the quality of LLM-based applications, including fine tuning, RAG, better retrieval methods or LLM as a judge mechanisms. In addition, Amazon AWS proposed an initial set of best practices for building robust GenAI applications [55, 56], providing guidance on how to design better agents, and advocating for comprehensive logging, observability and testing capabilities.

While these examples illustrate the benefits of integrating traditional SE approaches with GenAI-driven methods, or provide partial guidance for developing LLM-based or agentic applications, there is an urgent need for a more holistic approach to design and develop robust, adaptive, and efficient GenAI-enhanced systems. In this paper, we outline a vision for a GenAI-native system as an architectural paradigm that integrates the cognitive capabilities of GenAI with the established operational excellence, reliability, and dependability principles of traditional systems. We will build upon the existing software engineering paradigms and GenAI best practices, indicate where they fall short, and propose several new principles and patterns.

In this paper, we will not focus on the recent coding paradigms [50], tools [4, 9, 10, 16], frameworks [44, 68], or platforms [54, 61] associated with LLM-based coding and software engineering assistance (LLM4SE) aimed at aiding or automating the development and maintenance of software artifacts [22, 48, 64]. Our primary focus is on exploring the impact of GenAI on the evolution of future software design principles and patterns. Nonetheless, in Section 8, we will briefly address how existing cloud-native and agentic cloud platforms should be enhanced.

4 Example Use Cases and Applications

We claim that the vision and contributions outlined in this paper will be widely applicable across diverse use cases and application domains, spanning all layers of the stack and phases of the software development lifecycle. To better illustrate our vision and concepts, and to demonstrate the limitations of traditional or purely agent-based approaches, we first introduce a few selected example use cases. Further details for some examples can also be found in Appendix E.

GenAI-native micro-function. We will use a simple contact information parsing function as an example. Unlike traditional implementations that accept a limited set of inputs and modalities based on predefined structural rules—such as those easily parsed through pattern matching—a GenAI-native implementation should efficiently and reliably accommodate a broader spectrum of inputs across diverse modalities, including unstructured text, YAML, or images, without assuming or relying on fixed input formatting and structure. Furthermore, it should adeptly handle incomplete or inaccurate information. As we will elaborate, dependent

or downstream functions should also be architected to be resilient, effectively managing potentially incomplete or uncertain parsed data, rather than relying solely on the accuracy and certainty of outputs from the parsing function.

GenAI-native Web application. At the software system level, the future of GenAI-native web services and applications can be reimaged to offer greater flexibility and personalization. Unlike traditional web services, which typically provide specific, restricted, and rigid APIs and user interfaces for interacting with other internal or external services, a GenAI-native approach would enable both end users and other services to customize or personalize the interface and behavior dynamically. This customization could happen on-the-fly while maintaining the reliability, scalability, and safety properties inherent in traditional web services.

Such customizations can be either temporary or permanent and may necessitate bespoke communication, processing, storage, and user experience rendering capabilities. Note that this approach will also require adaptive specification of requirements, along with clear accountability and responsibility agreements with the end user or across such services.

Imagine, for instance, a user of a task list management service wishing to integrate supplementary weather information into their task list entries, extending beyond the core functionality supported by the service. Or similarly, an external service requesting additional information or attributes from a weather service, exceeding the traditional API's capabilities. In both scenarios, the target service must determine whether to accommodate such enhanced capabilities, establish the conditions under which they will be supported, and undertake the necessary steps to implement these custom features or experiences. For some these requests, these steps may occur almost instantaneously as a bespoke request, whereas others may require more time and effort to provide.

GenAI-native software upgrades. Microservice software upgrades could also be transformed into a fully GenAI-native process, encompassing both the initiation and execution phases. Instead of relying on traditional human-centric performance evaluation and optimization loops, a GenAI-native service could autonomously monitor its interactions with other services to identify suboptimal usage of its functionalities or interfaces. In response, DevOps agents within the service may independently decide to develop a new service endpoint with improved core capabilities. After rigorous testing and integration, this enhanced functionality could then be deployed and proactively communicated to other services, allowing dependent GenAI-native services to seamlessly incorporate the new features into their core logic, either autonomously or with human oversight. This approach would facilitate more proactive and seamless upgrades, thereby improving overall system efficiency and adaptability.

Discovering unknown unknowns. Traditional anomaly detection techniques, for example in the context of predictive maintenance, often focus primarily on *known unknowns*. However, self-reliant GenAI agents or systems may be able to help uncovering *unknown unknowns*—previously undocumented issues. Such systems would autonomously detect, explore, and verify potential issues, and learn from past incidents to improve effectiveness. Confirmed anomalies subsequently would trigger appropriate bespoke actions, and should be converted into *known unknowns*, by automatically creating, testing, and integrating new core anomaly detection routines using GenAI-native principles.

Enhancing legacy services. When upgrading traditional or legacy applications to GenAI-native systems, a viable strategy would be to retain the proven reliable logic of the legacy system while enhancing it with GenAI-empowered functionality to increase resilience and adaptability to a wider range of usage scenarios beyond those achievable through traditional logic or rules. For instance, in a bank transfer service, the core logic should ideally remain untouched.

However, GenAI-native enhancements could integrate asynchronous agents to detect and respond to anomalies beyond those easily identifiable through traditional heuristics, thereby providing enhanced security. A notable example might be detecting spurious transactions resulting from successful phishing attempts. These agents should learn and improve over time, adapting to new threats and safely reconfiguring decision-making processes as needed, with failsafe mechanisms to revert to legacy mode if issues arise.

5 GenAI-native Design Principles

This section defines the guiding principles for designing and building GenAI-native systems. We begin by first defining the five foundational pillars onto which we will establish these principles. We draw inspiration from historical technology transformations and human organizational methodologies, and highlight key differences with traditional systems.

5.1 Design Goals

Reliability. The capacity of a system to function correctly and predictably over time. It encompasses the ability to recover from (un)expected failures or disruptions (*resilience*), handle unexpected inputs, conditions or stresses without failing (*robustness*), and continue to function appropriately even when components fail or errors occur (*fault tolerance*).

Excellence. The capacity of a system to achieve the highest standards in performance, quality, and effectiveness within a domain. It encompasses the ability to apply learned knowledge, skills and behaviors in a specific context or domain (*competency*), produce consistent, predictable, and repeatable results (*precision*), and execute processes optimally, and with

minimal manual intervention, ensuring high-quality outcomes while maximizing resource utilization (*proficiency*).

Evolvability. The capacity of a system to change, grow, and improve over time in response to internal or external factors. It encompasses the ability to adapt to new environments (*adaptability*), incorporate incremental functional or structural changes (*flexibility*), and undergo significant restructuring and redesign (*malleability*) [33].

Self-reliance. The capacity of a system to handle things on its own and provide for itself. It encompasses the ability to solve its own problems and adapt to new challenges without heavily relying on external help (*self-sufficiency*), autonomously perform tasks, make decisions, and operate independently without external control or continuous human intervention (*self-governance*), and continuously self-enhance its performance and recover from issues through learning, optimization, or healing (*self-improvement*).

Assurance. The capacity of a system to foster a secure and trustworthy environment in accordance with predefined standards. It encompasses the ability to address biases and ethical considerations (*alignment*), protect against unsafe use, threats and vulnerabilities (*security*), and maintain the trust, integrity and privacy of sensitive information and key stakeholders (*trustworthiness*).

While many of these objectives are not unique to GenAI-based systems, the intrinsic characteristics and behavior of GenAI, coupled with the diverse range of tasks and inputs they can handle, make achieving these goals during the design, development and operationalization both critical and complex. For instance, the probabilistic nature of GenAI-based solutions and their outcomes requires more comprehensive and elaborate reliability measures, both throughout development and during operational phases. Additionally, self-reliant and evolving GenAI systems demand additional innovative assurance and performance measures to ensure safety and effectiveness. Successfully realizing these goals requires several key innovations, including the development of new design principles, methodologies, techniques, and tools. In this paper, we advocate for a more systematic approach, rather than relying on ad hoc or manual strategies.

It is important to emphasize, however, that not all aspects will be equally important across all application domains. Safety-critical AI domains may prioritize self-reliance, reliability, assurance and excellence, such as (multi-agent) autonomous scheduling and planning systems, robotics, or industrial automation. Others may emphasize assurance, reliability and excellence, such as customer support chatbots, business intelligence, enterprise workflows. Meanwhile, more creative domains may prioritize evolvability and excellence, including the creative industries and recent AI developer tools.

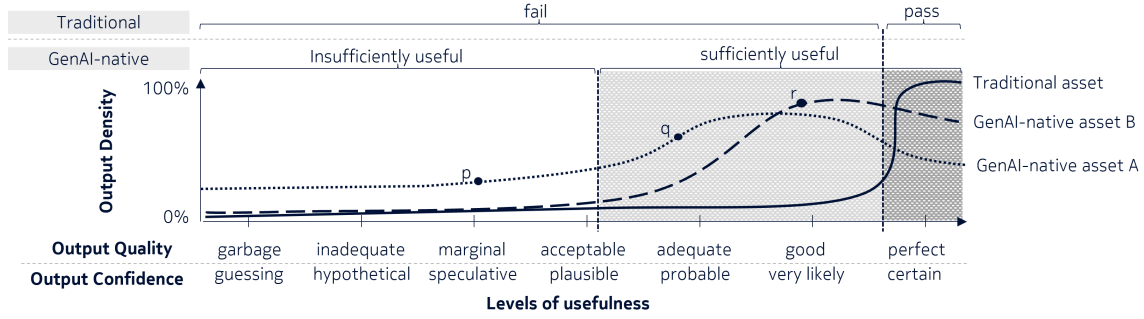


Figure 1. Example probability density functions of output quality or confidence for outputs produced by typical GenAI and traditional logic based solutions across the full range of acceptable inputs.

5.2 Design Perspectives and Analogies

To better motivate these paradigms and patterns, we first briefly draw parallels with two historical transformations and reflect on how we organize ourselves to cope with imperfections while striving for excellence. First, when transitioning from circuit-switched to IP networking to enable more efficient use of network resources, higher-level mitigation strategies like forward error correction, packet reordering, and retransmissions were developed to retain reliable communication. These strategies compensate for issues such as corrupt or dropped packets and out-of-order arrivals.

Second, when shifting from stateful monolithic applications to cloud-native stateless microservice architectures to facilitate efficient utilization of hardware and system resources, mitigation strategies like cloud-native architectures, advanced lifecycle management, and new design principles were developed to address performance and latency issues. These strategies address the challenges introduced by best-effort virtualized resource access and time slicing.

Third, when drawing parallels to human organizations, enterprises and factories organize themselves to accomplish tasks effectively by promoting efficient utilization of human and automation capabilities. Efficiency and mitigation strategies like team collaboration, clear communication protocols, proper resource planning, regular training programs, and robust feedback mechanisms have been implemented to address potential quality and efficiency issues, caused by best-effort organizational, operational, and management methods, as well as inherent human error.

These perspectives underscore the importance of developing robust mitigation strategies and organizational mechanisms to ensure higher layers remain efficient and resilient against the inherent limitations and imperfections of the underlying technology. We firmly believe that a similar approach is essential for GenAI-native software and knowledge systems, to help overcome the inherent limitations of the technology and bypass the "generate and pray" strategy present in many existing GenAI solutions.

6 GenAI-native Best Practices

In this section, we translate the high-level design goals and perspectives into a set of guidelines tailored for GenAI-native applications. Building upon established software engineering practices, we motivate why and how these practices should be extended or adapted to incorporate GenAI into future software design methodologies. While we do not claim that these guidelines are exhaustive, we hope they will serve as a robust foundation that will inspire the AI, programming, and software engineering communities to further refine and expand upon them. Table 1 in Appendix B also provides an overview of these guidelines.

6.1 Reliability Guidelines

We start with these guidelines, as we believe they are the most critical when designing and building GenAI-native systems, enabling a strong foundation for the other guidelines.

Design for fault-tolerance and resilience. Traditional software engineering solutions are often designed and developed according to clearly defined *pass/fail criteria*, and are rigorously tested to ensure they reliably handle predefined input ranges without failure. However, this approach is unattainable for GenAI-native systems due to the nature of GenAI as well as the kind of inputs and tasks.

Consequently, instead of relying on clearly defined *pass/fail criteria*, we propose the concept of *utility-based sufficiency criteria*. This emphasizes the practical effectiveness and adequacy of a solution in real-world scenarios, reflecting solutions that are *sufficiently useful most of the time* in terms of both output quality and inherent uncertainty.

A conceptual example is shown in Figure 1, where three probability density functions are depicted of the output quality and confidence likelihood across the entire set of acceptable inputs, for outputs produced via GenAI or traditional logic based solutions. Note that for similar tasks, the acceptable input range of traditional assets will typically be much narrower than for GenAI-empowered assets.

Not only do samples p , q , and r inherently have lower quality and/or confidence scores compared to the output from a traditional asset, sample p would be considered insufficient due to subpar quality or confidence, while samples q and r would both meet the sufficiency criteria. In many cases, sample r may be preferred, but in some cases, sample q may be preferred due to latency or resource utilization.

Applied to the contact information parsing example, a traditional implementation will typically only accept a very restricted set of inputs, according to a predefined set of structural rules. Asset A on the other hand could be a fully agentic implementation based on a simple AI model that can only reliably extract parts of the contact information. In contrast, asset B could be GenAI-native implementation, capable of extracting more information with greater reliability and accuracy, possibly also with other runtime tradeoffs.

This resembles also many professional human activities, where perfect outputs are not always attainable or necessary either. For example, in case of a human document summarization or detailed report creation task, different people will naturally produce different outputs, and we typically neither expect nor assume perfection. In many cases, this cannot even be uniquely measured.

Therefore, human produced outputs typically undergo a review process, according to task-specific evaluation criteria, which requires additional time and effort. GenAI-native systems should be architected accordingly.

Include verification and mitigation at all levels. Given the inherent reliability challenges of GenAI assets, it is crucial for GenAI-native systems to integrate thorough verification and mitigation strategies throughout the software stack and software lifecycle. Effective strategies include the native integration of design time and runtime self-verification and fact-checking mechanisms, as well as the use of external verification systems and tools.

Even more than in traditional systems, dependent assets should not presume the reliability or predictability of syntactically or semantically correct outputs, even when a GenAI asset asserts the implementation of self-verification and self-mitigation strategies. This may require incorporating cognitive, probabilistic, or approximate capabilities into dependent assets to interpret and assess the usefulness of received results at runtime, as well as implementing additional quality absorption or adaptation strategies, which often may partly require them to be a GenAI-native assets as well.

Mapping this onto our example use cases, a GenAI-native web service or parsing function, as well as its dependent services or functions should always anticipate potential communication, information or processing issues. In addition, they should implement appropriate sanity checks and mitigation strategies, not only through extensive pre-production testing, but also at runtime, and in production.

Restrict scope of unreliability. Since mitigation strategies may involve absorption or delegation, sources of unreliability may easily spread across assets at runtime. While this can be acceptable, for example when latency and throughput are prioritized over highly accurate outputs, it is advisable to restrict the scope of unreliability.

Beyond traditional circuit breakers from distributed systems, a GenAI-native variant may first implement additional conversational mitigation strategies, including requesting the upstream asset to partly redo or improve computations, or falling back to more conservative approaches, before giving up on the asset altogether. Additional considerations include the ability to induce and measure mutual progress, assess convergence, and possibly consider game theoretical strategies such as the Nash equilibrium [39].

This approach mirrors how humans typically collaborate within teams or functional units, where imperfection and multiple iterations are generally well accepted internally but less so across teams. In case of GenAI-native web services, web services should be designed with proper strategies to cope with inherently unreliable dependent services, not only because of networking or stability issues, but also because of unreliable semantic communication and processing.

Promote transparency of processing and risks. Instead of only returning the results with other assets through traditional interfaces, GenAI-native assets should be more transparent when sharing their results with other assets, allowing the latter assets to more easily assess the usefulness of the provided results. Transparency may include sharing the applied processing paradigm (e.g., AI-based versus traditional), providing an interpretation of the request and explaining the execution or reasoning process, as well as sharing self-verification and mitigation strategies used during its processing. Assets should negotiate the amount of risk, transparency as well as responsibility and accountability before interacting. In addition they should (proactively) send additional metadata alongside the actual output, or interact through conversational paradigms.

In our example use cases, based on additional provided metadata, downstream assets can evaluate the reliability and quality of parsed contact information, or assess the usefulness of answers provided by GenAI-native web services. This evaluation may include determining whether the task was completed using traditional or cognitive processing, evaluating the provided confidence score, or reviewing any additional execution comments or requests for further clarification made by the asset.

Plan for contingencies. Similar to distributed systems, GenAI-native assets and subsystems should integrate contingency strategies to address unexpected issues or errors. One example strategy includes incrementally generating several outputs using diverse techniques, resources or assets. As a result, it is essential to plan for adequate processing margins

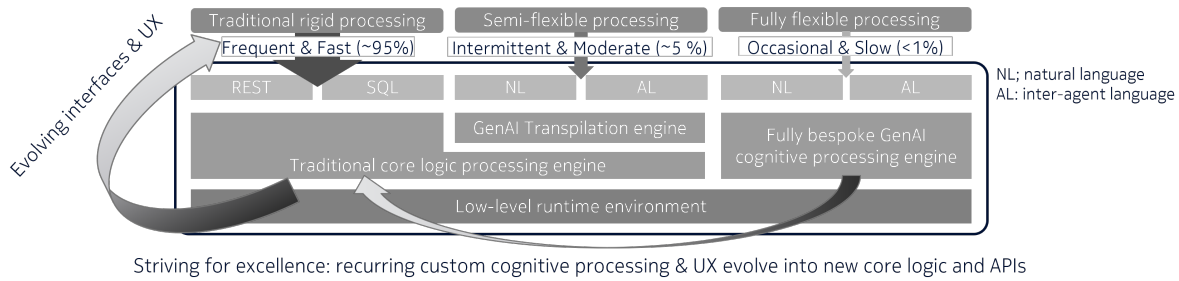


Figure 2. Conceptual view of a self-improving hybrid GenAI-native asset: fast routine traditional processing & interfacing, and slow occasional (semi-)cognitive processing (cfr. System 1 and System 2 thinking [28]), with gradual optimization loops.

and slack time to improve overall outcomes or recover from anticipated challenges.

For instance, downstream assets might opt for more costly or slower methods to accurately extract specific contact information or to perform some web service. Unlike distributed systems, the challenges in GenAI-native systems are more likely to arise from interaction and processing complexities rather than networking or performance-related issues.

Like human contingency methodologies, assets should plan ahead, preparing and executing tasks well before the deadline, considering the estimated time and resources required for potential refinements. As discussed later, we implicitly assume that GenAI-native assets and services will increasingly operate asynchronously, proactively, and in parallel, akin to a true distributed or federated system, rather than traditional reactive micro-service architectures.

Minimize dependency on cognitive processing. We believe that, in general, a foundational guideline is to minimize dependency on cognitive processing within all assets, either from agents or human-in-the-loop, effectively reducing the level of *artisanal* processing. This approach, illustrated in Figure 2, contrasts with a current trend of GenAI-first agentic solutions, where agents are central to most critical processing paths. Example strategies include systematically identifying and writing dedicated code snippets, or training a more narrowly scoped yet efficient AI/ML model to replace common cognitive workflows. In our example use cases, dependency on (generic) cognitive processing should be systematically reduced by identifying which inputs or service requests can be handled by more efficient and reliable processing methods and interfaces.

Reducing open-ended cognitive processing not only serves as an excellent mitigation strategy for enhancing reliability, it can also improve operational excellence, as well as mitigating assurance issues. Finding the optimal balance between cognitive and traditional processing will require the development of additional domain-specific and task-specific expertise, tools, and techniques. We can draw inspiration from how we discover and organize ourselves to automate and harden repetitive tasks, while specific human processing

can augment these solutions to provide bespoke outcomes or customize existing repetitive tasks.

While exceptions to this guideline exist, such as in search or optimization tasks where large language models (LLMs) or reasoning models can expedite processes compared to traditional methods, it remains crucial to determine the optimal balance between GenAI and traditional assets.

6.2 Excellence Guidelines

In addition to enhancing the reliability, it is crucial to also consider quality and efficiency—two often conflicting requirements—during design and operations. The following principles provide guidance on balancing these requirements.

Build upon proven design principles and practices.

While it may seem evident, it is crucial to emphasize the importance of leveraging established software engineering and organizational methodologies and practices to systematically improve quality and efficiency. For instance, process methodologies such as incorporating checklists, continuous software testing and CI/CD pipelines, or standard operating procedures (SOPs) help ensure critical processing steps are not overlooked and prevent reinventing the wheel. GenAI-native assets should not only build upon these through proper tools and frameworks, they should also be revised, optimized and extended to allow seamless blending of traditional and cognitive processing.

In our example use cases, it could be beneficial to implement checklists or customized validation procedures to verify the accuracy of the generated results, such as the parsed contact information or the return messages from GenAI-native web services. More importantly, since GenAI models have a limited lifespan, swapping the model underneath GenAI assets or multi-agent systems will inevitably alter their *personality*—that is, their behavior and capabilities. This can lead to ripple effects, including potential stability and convergence issues. As it is unlikely that such updates will produce a service identical to its predecessor, robust design, testing and service evolution principles will be essential to prevent significant disruptions.

Optimizing cognitive workflows. Systematically reducing cognitive processing can significantly enhance efficiency and reliability, leading to more effective and streamlined operations. This involves transforming repetitive or time-consuming cognitive tasks, eliminating unnecessary steps, and preventing recurring chain-of-thought reasoning. Traditional workflows can substantially improve runtime efficiency while also minimizing the need for additional costly reliability measures. Furthermore, this approach may mitigate the impact of swapping the underlying GenAI model in a GenAI service, as most repetitive tasks implemented through traditional methods would remain unaffected. However, this will necessitate adaptive routing and handling capabilities across all request types and modes.

Applied to our example use cases, traditional parsing methods, service handling and API protocols should be leveraged as much as possible to efficiently and reliably communicate and handle known input requests. Ideally, this process is fully dynamic and adaptive, rather than being hardwired. Specifically, in the context of traditional APIs, MCP, and A2A protocols, assets should ideally prioritize interaction through traditional APIs for both communication and processing. This approach ensures reliability and consistency. In addition, assets could also serve as dependable MCP resources or tools towards other GenAI assets. Full A2A communication and processing on the other hand should be reserved when it is strictly necessary.

Systematic quality verification and retrospectives. In addition to optimizing cognitive workflows, systematic quality review, verification, and improvement can significantly enhance quality and efficiency [56]. This may be achieved by incorporating continuous learning and feedback loops at strategic points, including at runtime, leveraging and rethinking established methodologies such as Kaizen [26] and Six Sigma [47], performing regular code quality and maintainability checks, and allocating dedicated time blocks for targeted experimentation.

In our example use cases, it is crucial to consistently evaluate the quality, efficiency, and effectiveness of GenAI-native functions and services. This involves assessing the performance of GenAI assets against new input types, or monitoring their evolving capabilities. Consequently, GenAI-native systems will demand more comprehensive and sophisticated verification procedures than traditional software systems, often dependent on advanced cognitive methods. Striking the right balance between these approaches will be essential for optimal performance and effectiveness.

6.3 Evolvability Guidelines

A unique capability of GenAI is its ability to generate innovative solutions for new problems or tasks. This enables GenAI assets to be flexible and adaptive, creating new solutions

and improving existing ones. However, this also requires a sufficient degree of restraint and discipline to avoid chaos.

Promote resilient and adaptive designs. Instead of returning an error codes or throwing exceptions, as is common in traditional systems, GenAI-native assets can be designed to be more resilient and adaptive to unexpected requests and conditions. For instance, if a dependent function or service is malfunctioning, overly restrictive, or unavailable, the asset should leverage its cognitive processing capabilities to actively resolve the issue and find a solution, rather than simply throwing an exception. Similarly, in case of GenAI model swaps or other behavioral changes, dependent assets should be designed to be tolerant and adaptive against such changes.

In our example use cases, GenAI-native functions or services should be designed to (temporarily) adopt alternative processing or communication strategies when encountering issues. In the worst-case scenario, they should be able to switch to another function or service to bypass the problem. For instance, if a web service relies on a weather or location service that becomes unresponsive or cannot handle a custom request, or its *personality* has altered, the service should either adjust its strategy or seek an alternative service capable of fulfilling the request.

Evolve towards reliable and efficient systems. Whenever resilient and adaptive cognitive processing and designs are required, the system should monitor and record these bespoke behaviors and interactions. If such behaviors occur frequently, or the cognitive processing becomes too expensive, the system should evolve its internal processing to natively support these capabilities, resulting in a more reliable, repeatable and efficient solution. In other words, a GenAI-native system or asset should learn to evolve in a manner that systematically reduces its dependency on cognitive processing and bespoke solutions, as shown in Figure 2. For our example use cases, this could mean gradually expanding the core functionality of the parser function or web service.

Promote consistency over creativity. Unless required, GenAI-native systems should restrain their creativity, bespoke strategies, and solutions. Though obvious in traditional systems, consistent, repeatable and dependable behavior should always be preferred over overly creative agentic solutions, especially for common tasks and scenarios. Such approach not only will enhance system stability but also reduces inconsistencies between assets, such as constantly needing to compensate for custom cognitive interactions and responses. This will also result in a smoother experience, for example in case of evolving web frontend designs, as people generally prefer stable and predictable look-and-feel and behavior.

Applied to our use cases, the evolvability of GenAI-native micro-functions or web services should be limited, emphasizing consistency and resilience over creativity. Essentially, these services and functions should maintain their specialization, rather than evolving into generalist agentic solutions. Note that this approach does not preclude a GenAI-native system from incorporating a variety of generalist services as well, similar to human organizations.

Collective competency ecosystems. A potential key advantage of AI agents over humans is the efficient transfer of new competencies (e.g., knowledge, expertise, or skills). Instead of having assets independently rediscover similar competencies, a GenAI-native system should facilitate easy, efficient, and safe sharing of new competencies within and across systems. This allows individual assets to explore new competencies while benefiting from collective advancements.

In our parsing example, more reliable or efficient contact information parsing prompts, methods or extensions for particular inputs could be shared with other systems to avoid rediscovery. This may in a further evolution and convergence of existing open source and marketplace ecosystems, where constantly evolving assets, beyond standalone model hubs or code repositories such as HuggingFace or Github, can be actively shared and ingested, respectively.

6.4 Self-reliance Guidelines

A key objective of GenAI is to develop fully autonomous systems that reduce the reliance on restrictive, hand-crafted solutions. Given the transformative potential of such future systems, it is crucial to exercise caution and restraint. Although the discussed guidelines are not unique to GenAI-native systems, it will be essential for deeply integrating them into the code design and operations of such systems.

Balance autonomy with safety and control. Self-reliant GenAI-native systems must implement clear decision-making frameworks and policies to govern all autonomous actions. Ideally, they should be made aware of such policies to allow for anticipation instead of continuously encountering opaque barriers. This requires additional safety checks and validations to detect adversarial attempts to circumvent the policies. Furthermore, internally and externally triggered measures should be implemented in case of repetitive misbehavior, such as penalizing, replacing, or disabling the asset. In our example, this may limit the ability of the GenAI-native contact parsing function or web service to evolve or even restrict its usage if it becomes too unreliable.

Include rollback capabilities for autonomous actions. GenAI-native assets should be designed with robust rollback mechanisms to address the impact of suboptimal autonomous decisions and evolutions. This includes the ability to revert or unlearn newly acquired competencies, actions, interfaces, or data representations. For instance, it should

be straightforward to undo ineffective APIs or processing enhancements made to web services.

Perform regular reviews and updates. All autonomous or self-reliant agent behaviors should be regularly reviewed internally and externally to detect potential issues that may have gone unnoticed. Based on these reviews, policies may be updated, and assets may be required to adjust or partly revert their behavior accordingly. Ideally, these processes should include mechanisms to learn from past incidents and actions, preventing endless loops of recurring violations.

Maintain visibility into autonomous operations. To facilitate review and rollback in autonomous systems, log trails of all actions and decisions, including self-diagnostics and monitoring at multiple levels, and possibly involving human oversight, are essential. For instance, if an asset decides to rewrite parts of its core logic, or wants to access new services, this should be logged and potentially gated. Human organizations maintain clear rules of engagement regarding the levels of autonomy and self-reliance that are permitted versus those that require prior approval.

6.5 Assurance Guidelines

Implementing required assurances within GenAI-native systems is crucial and impacts all other pillars. This section highlights key guidelines, recognizing that further effort and consideration is required to fully understand all aspects. Though these are obviously also not unique for GenAI-native systems, GenAI imposes several new and hard thread vectors that need to be dealt with accordingly.

Adopt GenAI security best practices. The OWASP guidelines [46] highlight several threats and issues related to LLMs and agentic processing. Example concerns include insecure output handling, supply chain vulnerabilities, excessive agency, and sensitive information disclosure. It is prudent to assume that all GenAI-native assets may be internally or externally contaminated or compromised, whether deliberately or inadvertently. As overreliance on cognitive processing can easily exacerbate the problem, this guideline underpins the main rationale of this paper: to carefully balance traditional and cognitive processing, and to handle all remaining cognitive processing with the utmost scrutiny.

Provide observable and explainable solutions. GenAI assets should implement sufficient observability, interpretability, and explainability mechanisms. Examples include introspectability of cognitive techniques, as well as readability of all generated code, reasoning plans, and prompts. These mechanisms should be actionable, allowing for immediate remediation as well as long-term maintainability, reviews, and optimizations.

Design for privacy, integrity and trust. In addition to enforcing security best practices and ensuring transparency,

GenAI-native assets that process or exchange sensitive or confidential information must be tightly managed and controlled. Practical solutions include providing appropriate guardrails, running these assets within secure and well-guarded sandboxes, and tightly manage and restrict the information that can be processed within these sandboxes, or that can flow outside of them.

Assess and mitigate misalignment. As cognitive processing is highly susceptible to biases, proactive and reactive measures are essential to prevent, detect, and hopefully mitigate these issues. Measures include carefully selecting the underlying GenAI technologies, incorporating active guardrails and filters, and regularly conducting focused audits on outputs, intermediate reasoning, processing, and generated code.

6.6 Architectural and Operational Guidelines

In this section, we outline key architectural, organizational and operational guidelines for designing systems that adhere to the core GenAI-native guidelines discussed earlier.

Reimagine cloud-native paradigms. While cloud-native design principles and patterns provide a solid foundation for designing, developing, and managing GenAI-native systems, they need to be rethought to accommodate the unique characteristics of GenAI-native systems. For instance, cloud-native principles lack the flexibility to efficiently handle the inherent malleability, reliability, and self-reliance aspects of future GenAI-native systems. Concepts such as immutable infrastructures, rigid service meshes, and fixed API-driven service interfaces, though useful as foundational principles, need to be generalized, relaxed, or extended.

Specifically, immutable infrastructures should evolve into *reproducible organic infrastructures*, where GenAI-native assets can freely evolve at runtime, while retaining easy and efficient replicability in case an asset fails or needs to be scaled. Similarly, service meshes should evolve into *organic substrates*, enabling assets to interact more freely while remaining compliant. Finally, fixed API-driven service interfaces should transform into *unified conversational interfaces* (UCI), comprising an organic mix of traditional yet dynamic APIs, as well as conversational communication mechanisms.

Create future-proof designs. GenAI-native designs should reflect the organic and evolvable nature of GenAI. This involves transforming existing micro-service designs into more modular micro-function designs, to facilitate independent evolution of core functionalities and interfaces, but also creating future-proof internal data representation schemas. This also involves anticipating changes in the constellation and communication patterns among assets, akin to the organic evolution of human teams and their interactions. Future-proof designs should also allow for easy rollback of code, state, as well as data representations to earlier versions.

Relax strict application and system boundaries. Similarly to loosely coupled distributed systems, strict boundaries between GenAI-native assets and systems should be relaxed into a more organic substrate or organizational structure. Through evolvability and self-reliance, and driven by a continuous pursuit of excellence, GenAI-native systems should allow for dynamically adapting its organisational structure, such as automatically switching to alternative services with better capabilities or behavior, integrating novel services to accommodate custom requests, and more.

GenAI assets allow for more easily interchangeable communication protocols (cfr. MCP or A2A), enabling GenAI-native (web) services to more easily swap out one service for another, even though the alternative service may have a very different personality providing somewhat different functionality. For example, if a user wishes to incorporate weather information into a GenAI-native calendar service, the system should autonomously determine how to contact a weather service and integrate this functionality seamlessly.

Impose clear scope and responsibilities. A GenAI-native system requires strong organizational principles to ensure stable operations and behavior. This requires clear governance to establish the scope and responsibilities for all assets, their interactions, and the extent to which they are allowed to evolve. This requires actively monitoring all GenAI-native assets as well as imposing restrictions.

For example, if an asset autonomously decides to communicate with a non-approved service, makes unauthorized changes to core functionality, uses non-interpretable interfaces and protocols, or is clearly exceeding its intended scope or responsibilities, the system should intervene and compel the asset to take corrective actions. This may include shutting down the asset or forcefully reverting it to an earlier state. Traditional organizational software engineering patterns may be extended with human organizational structures to better orchestrate and coordinate these aspects.

Model GenAI agents as digital workers. In AI-first multi-agentic designs, agents are often positioned at the center of the system, where all decisions and processing flow through them, and the actual resources are accessed via tools. In our vision for GenAI-native systems, agents should instead adopt a collaborative worker role, focusing on the resources they manage and operate. These agents should continuously strive to optimize their involvement, minimizing their presence in any critical path unless necessary. Much like a bee colony working together towards a common objective, these agents should facilitate seamless collaboration and efficiency.

Create self-contained GenAI-native capsules. As will be discussed in Section 7, a GenAI-native asset comprises multiple components, both active and passive, akin to a biological cell. Proper orchestration and management of these assets requires an evolution of existing cloud-native patterns.

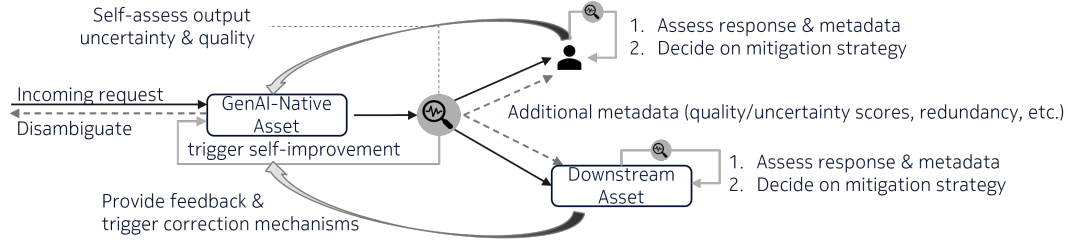


Figure 3. GenAI-native blueprint comprising several patterns for enhancing resilience to quality and uncertainty variations.

Adopt organic lifecycle management. The lifecycle management of GenAI-native systems should natively incorporate evolvability and self-reliance. Next to traditional, planned offline application enhancements and optimizations, it is expected that the traditional boundaries between software development and CI/CD cycles will blur, as GenAI-native assets will be able to modify actively deployed assets. To better support this, DevOps and XOps practices should be enhanced to empower hybrid human and agentic application code sign. CI/CD approaches should become bidirectional, allowing online GenAI-native assets to actively contribute to the system, rather than merely receiving pushed updates.

6.7 Programmability Guidelines

Developing adaptive, self-evolving GenAI systems requires specialized programming practices across the full stack.

Develop organic programming paradigms. GenAI agents should be able to easily make small, targeted changes to an existing code base, application interfaces or data structures, even while the application is running. Therefore, we envision a more modular micro-function programming paradigm, with tiny functional and data representation constructs that have a clearly defined purpose, and can easily evolve or be rolled back. Such paradigm should also allow human and agentic developers to easily distinguish between core and organic functionality, including separating the traditional from the cognitive parts. Additionally, innovative organic programming techniques must be developed to better cope with aspects like *sufficiently useful*, probabilistic and evolvable outputs produced by other assets.

Develop a flexible policy language. A crucial aspect to successfully manage and control GenAI-native systems, is to be able to easily yet flexibly define and constrain the degree and scope of malleability and self-reliance of GenAI-native assets. This capability needs to be available at all levels, should be easy to express, customize (possibly autonomously), and enforce. Assets themselves should be aware of their capabilities and restrictions, possibly through machine readable contracts, protocols or agreements. Some of these may be the result of active negotiations across GenAI-native assets, whereas others may be the result from external organizational entities enforcing these onto these assets.

7 GenAI-Native Design Patterns

In this section, we present a set of initial behavioral, structural and creational design patterns based on the design principles and guidelines described earlier. We first focus on lower-level enabling patterns before integrating them into higher-level architectural and operational patterns. Table 2 in the Appendix provides an overview of all design patterns.

Although several of the proposed patterns are not exclusive to GenAI-native systems, as we will discuss, these patterns will often be more impactful, intrusive, and vital for maintaining a robust and reliable evolvable GenAI-native system, especially considering the inherent nature of the GenAI technology and the intended future applications.

7.1 Reliability Patterns

Reflective processor. This pattern advocates for the inclusion of meta-cognition and self-regulation mechanisms, and to take necessary corrective or mitigative actions rather than blindly accepting them. This may involve disambiguating the task before actual processing occurs. This may also involve triggering additional pre and postprocessing steps, retrieving additional information or more context, running additional verification, involving other assets, or simply absorbing the reduced quality or confidence. It is crucial to recognize that such reflective processing may also be necessary for dependent non-GenAI assets.

Reflective communicator. This pattern involves creating transparent and redundant bidirectional communication channels between assets. Transparency measures may include transmitting self-assessment scores or reports as additional metadata, either by the sending asset or as feedback from the receiving asset. Redundancy measures may involve producing multiple outputs, accompanied by additional context information, rather than a single output. Both patterns are illustrated in Figure 3. Combined, they form the foundation of a robust GenAI-native system.

Resilience fender. Aside from the traditional circuit breaker pattern from distributed systems, GenAI-native assets should also account for (accumulated) expected uncertainty or quality degradation issues, and implement mechanisms to either absorb or mitigate issues. This may include actively forcing upstream assets to partially or completely redo some of the

processing. GenAI resilience fenders may range from being very flexible, in case the asset is able to absorb larger issues, towards being extremely firm, for example when integrating with legacy non-GenAI assets and services, where reflective processing and communication may not be supported. They may also act as cognitive circuit breakers, for example to avoid problematic assets from bringing down the system.

7.2 Excellence Patterns

Programmable router. We propose a flexible integration of traditional and cognitive processing, where traditional core logic handles routine workflows while cognitive processing manages more complex and exceptional cases. This approach is reminiscent of the *thinking fast and slow* paradigm [28]. A key enabling pattern to achieve this is the inclusion of a highly efficient programmable router or switchboard. This router can be deployed early in the processing pipeline to determine the optimal handling method for each request. Alternatively, it can be configured to reroute internal processing, such as adaptively switching between fast and slow paths, to triggering additional processing or handle exceptional cases if needed.

Such application-specific router must be highly efficient to preserve the benefits of traditional core logic, yet highly programmable to allow for continuous evolution. This may involve an iterative or hierarchical decision process, ranging from directly invoking core logic via specific APIs, over efficient LLM based domain and task routers [57], to agents reasoning about how to decompose and resolve a request. In each case, the router should be able to select between core logic functions or agentic implementations, and have the option to reroute when needed.

Continual self-reflection. To ensure and enhance quality and efficiency, GenAI-native assets should implement several quality assurance mechanisms. These may include incorporating feedback loops, enabling and executing auditing trails, or running self-consistency checks. The results of such measures can subsequently trigger one or more evolvability or self-reliance patterns. This pattern is comprised in Figure 3.

7.3 Evolvability Patterns

Unified conversational Interface. Instead of only relying on rigid APIs or natural language (NL) interfaces, GenAI-native assets should adopt a hybrid approach, where communication may happen via NL or via dedicated protocols both assets agree upon. This enables a more flexible interaction, beyond what is available through rigid APIs. A core feature of such unified interface is the ability to gradually evolve from freeform conversations towards more traditional APIs. This implies that the latter API should not remain static but will gradually evolve over time. For efficiency and reliability, GenAI assets should encourage the reuse of existing APIs and

logic, rather than continuously developing custom interfaces and logic.

Cognitive workflow optimizer. This pattern involves the systematic identification, formalization, and transformation of key cognitive communication and processing patterns into traditional workflows. When integrated into the end-to-end functionality, it is also necessary to reconfigure the programmable router to correctly route relevant requests to these new workflows. To prevent the proliferation of highly specific and difficult-to-maintain workflows, it is essential to systematically review and refactor such workflows.

Organic service broker. In traditional software systems, dependent assets are often tightly integrated, resulting in strong hidden dependencies. When a dependent assets fails, is unavailable, or cannot handle the request, the depending asset typically only can raise an exception. This pattern allows for easily and dynamically switching to alternative functions or services, either to improve or to ensure continuous yet possibly degraded functionality. This involves detecting when to switch and finding other good alternative services or functions. This will typically also require additional cognitive processing to help retrofitting the new dependency and adapt subsequent processing and communication.

Malleable data. This meta-pattern advocates for the creation, storage, and management of adaptable and evolvable data structures, representations, and handling methods to effectively address dynamic and evolving requirements driven by the systematic evolution of GenAI-native service logic, inputs, and modalities. Concrete patterns, paradigms and solutions will need to ensure that the evolution of such data organization and modeling incurs minimal disruption to the core logic. Furthermore, to enable seamless updates and facilitate easy rollback, solutions should support extensive versioning.

7.4 Self-reliance Patterns

In this section, we introduce several high-level patterns for effectively managing self-reliant assets, focusing primarily on ensuring their reliability, security, and continuous improvement. While these patterns are not unique to GenAI-native systems, existing frameworks and mechanisms must be extended and fortified to address the unique challenges posed by GenAI.

Asset lifecycle management teams. Like in a human software company, multiple agentic or hybrid human/agent roles and teams will be responsible for managing the entire lifecycle of a GenAI asset. Example teams include asset management teams, asset development teams, and customer support teams. Some of these roles or teams may operate externally to the asset, while others may be, at least partially, directly integrated into the production asset itself.

For instance, when the asset receives a custom request that necessitates unforeseen capabilities, effort, or support, a team comprising agents and/or humans may be deployed to evaluate the short-term or long-term impact on the system. These teams may opt to temporarily allocate additional resources or initiate bespoke workflows to address the request. Alternatively, teams might choose to invest resources in expanding or enhancing the core asset's capabilities, following a thorough cost-benefit analysis and alignment with internal policies. For simple bespoke requests with minimal expected impact or overhead, the system may decide to skip many of these steps or use a very lightweight procedure.

Behavioral policy-driven safeguards. Self-reliant assets should be explicitly informed of their permissible actions and restrictions through a set of static and dynamic guidelines, rules, or feedback. By sharing these rules of engagement through specific channels, assets can proactively learn to operate within these boundaries and automatically adapt to any changes. In addition to these rules, robust guardrails and controls must be provided to detect and prevent unintended, malicious, or wasteful behavior. This integrated approach ensures compliance, enhances operational efficiency, and maintains the integrity and security of the assets, providing a comprehensive framework for reliable and secure asset management.

For example, in case an external web service has a very specific request that would require extensive additional effort or trigger unforeseen actions, the system should be able to recognize such requests and act according to provided policies, such as denying such request or enforcing human oversight. Similarly, in case a self-reliant asset or agent wants to perform particular sensitive actions, appropriate safeguards should automatically kick in, for example by gate keeping access to specific systems, tools or other resources.

Infallible fail-safe and recovery mechanisms. Self-reliant systems should incorporate inescapable fail-safe mechanisms to abruptly halt any further autonomous behavior in the event of abnormal, erratic, inappropriate, or highly inefficient actions. This is crucial to prevent further damage. In addition to a fool-proof built-in emergency shutdown functionality, various mitigation or recovery scenarios should be automatically triggered to maintain normal operation. Mechanisms such as rolling back to an earlier version of the asset, switching to a more conservative solution, or reverting to a fail-safe mode that provides minimal functionality ensure continuity and stability, safeguarding the system against potential disruptions.

Comprehensive logging and introspectability. Assets must maintain detailed logs of all self-reliant actions to facilitate visibility, auditing, and retrospective analysis of the efficacy of such behavior. These logs should document all actions taken, their rationale, effort involved, and impact.

Additionally, self-reliant assets should expose themselves to external inspectability by authorized entities through dedicated communication channels or direct inspection and diagnostic capabilities. This framework enhances transparency, accountability, and trust, allowing insights gained from analyses to trigger improvements and ensure continuous enhancement of the asset's performance.

7.5 Assurance Patterns

Cognitive screening. Outputs originating directly or indirectly from GenAI-native assets should undergo thorough quality, security and safety screening before being accepted for further processing. This requires implementing robust encapsulation and gating mechanisms, spanning across all layers. Depending on the mutual trust, reliability, and sensitivity of the output or task, adaptive screening mechanisms can be employed. For self-reliant assets, cognitive screening complements higher-level behavioral safeguards.

Cognitive firewall. To prevent undesirable cognitive communications and interactions across GenAI-native systems, and to provide scalable oversight and compliance, these systems must implement cognitive firewalling mechanisms. These mechanisms should be layered on top of traditional micro-service firewalling to control access within service meshes effectively. Cognitive firewalling can be implemented via rule-based or cognition-based deep communication inspection mechanisms, as an analogy to deep packet inspection. As with cognitive screening, different and adaptive levels of scrutiny can be implemented across different GenAI-native assets, based on external policies. The sidecar pattern, known from service meshes, can be repurposed for implementing such cognitive firewalls.

Agent sandbox. Sandboxing encapsulates the runtime and processing of an asset within a self-contained environment, allowing strict control over the impact of GenAI-native assets [11]. One type of sandbox prevents adverse effects from malicious, misbehaving, or erroneous assets. Another type facilitates safe experimentation and analysis, possibly using a digital twin of the real environment. A third type, possibly implemented via trusted execution environments (TEEs), allows untrusted assets to operate on sensitive data by regulating information exposure, which is beneficial for federated systems and external auditing.

Cybersecurity and compliance units. GenAI-native systems, similar to human enterprises, should integrate internal cybersecurity and compliance units alongside external auditing to ensure security and scalable oversight. These units can be managed by both autonomous agents and human oversight, governed by transparent and easily programmable policy frameworks. Key components include cognitive firewalling, screening, reporting, and inspection.

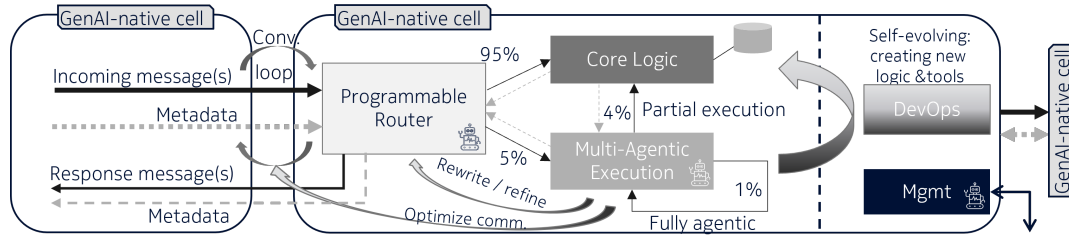


Figure 4. This figure illustrates the inner workings of a GenAI-native service, knowledge, or cyberphysical cell.

7.6 Architectural and Operational Patterns

GenAI-native cell. We propose a key architectural and organizational pattern for designing and implementing GenAI-native assets, termed the *GenAI-native cell*, as depicted in Figure 4. The GenAI-native cell is a core building block that encapsulates many of the lower-level design patterns discussed earlier, encompassing all five foundational pillars. It represents a self-contained functional unit with a clear intent, to design and develop assets that are tolerant, adaptive, and strive for excellence.

Architecturally, a GenAI-native cell is an evolution of a microservice, and could be implemented as a set of sidecar containers, wrapped into one or more dynamic and adaptive service pods. This concept is partly inspired by a biological cell, consisting of a core (i.e., the nucleus), a semi-fluid cytoplasm containing various organelles supporting dynamic processes, and a cell membrane that interacts with and adapts to its environment.

Similarly, in the GenAI-native cell, we envision a static core, multiple dynamic processes, and adaptive interaction with neighboring cells. The core of a GenAI-native cell comprises all common logic, including traditional logic and AI/ML modules tailored to efficiently support specific subtasks or workflows. Alongside the core, we envision additional active and passive components to dynamically extend its capabilities. This includes one or more cognitive processing assets.

To interact with other cells, we envision an adaptive router responsible for handling and routing incoming and outgoing requests, and providing seamless handover between core and agentic processing, while considering reliability and resilience measures. To enable cells to evolve in functionality, processing, and communication efficiency, we envision active DevOps agents, whereas management components are responsible for managing and controlling the overall operations of the cells. Both DevOps and management assets can be deployed partly or completely inside or outside each cell.

Organic substrate. To flexibly configure, manage and control multiple GenAI-native cells, their evolution and their interconnectivity, we propose the concept of an *organic substrate*, which represents an evolution of a service mesh or service chain. Within such substrate, cells can enter, exit, and

possibly move across substrates over time. Logical or functional clusters of cells can be grouped and managed together as tissues or organs, each with higher-level joint purposes and goals, akin to human organizational structures.

The substrate supports adaptive and resilient interactions among cells, facilitating seamless integration and cooperation, including easy discovery and communication with new cells. It enables a GenAI-native system to dynamically respond to changing requirements and conditions, promoting scalability, resilience, and agility. See also Appendix D for more details and an illustrative example.

Reproducible infrastructure. Immutable infrastructure involves deploying microservices as immutable entities to facilitate reproducibility. In case of GenAI-native cells, this principle needs to be relaxed to allow application logic and state to evolve at runtime, while retaining easy reproducibility. To achieve this, the organic and self-evolving nature of GenAI-native cells must be captured and stored in a manner that allows for easy reconstruction, ensuring that the cloned cell retains the evolved capabilities of the original cell.

It will be essential to minimize the degree of evolvability of such cells relative to their original base 'stem' cell, to facilitate easy reproducibility and restricting excessive variations, thereby preventing cells from becoming dysfunctional or "cancerous". Long-term evolution of cells to incorporate fundamentally new capabilities should still be handled in a coordinated manner, outside a live GenAI-native system.

7.7 Programmability and Software Design Patterns

Efficiently programming, configuring, and controlling GenAI-native systems requires new programming paradigms. New patterns and principles need to be developed and validated to address challenges such as unreliable outputs (i.e., non-deterministic, lower quality), the evolvability of assets and data structures and future human-agent codesign.

Principles from existing specialized domains may be adopted, including modular code design, or probabilistic and live programming. However, a more radical approach, including the creation of new programming languages or paradigms may be required. A detailed analysis of software and programmability patterns in the context of designing and managing GenAI-native systems is beyond the scope of this paper.

8 The GenAI-Native Software Stack

In this section we propose possible extensions to the traditional cloud-native software stack [40, 52] and current agentic cloud platforms [2, 17, 37] to build, manage, and run future GenAI-native applications and systems.

Infrastructure layer. Minimal infrastructure requirements to efficiently support GenAI-native systems include the availability of specialized processing, networking and storage hardware for training, fine-tuning, or running (batch) inferencing jobs for small and large GenAI models. In particular, energy and runtime efficiency will be crucial, as future GenAI-native systems will include assets that constantly remain active, requiring sustainable and cost-effective operation. For securely deploying GenAI assets, support for confidential runtime environments may be required.

Provisioning layer. This layer facilitates the automated allocation and configuration of a low-level runtime environment for deploying applications and services. The self-reliant, evolvable aspects of GenAI-native applications may require revisiting existing frameworks and tools to better support their adaptive and organic nature, particularly when deployed in heterogeneous and distributed environments.

Runtime layer. This layer manages low-level virtual block and object storage capabilities, container runtime management, and virtual networking functionalities. The evolvability of GenAI-native systems requires highly efficient checkpointing capabilities to ensure seamless and rapid reproducibility when evolved GenAI-native cells need to be recreated elsewhere. Additionally, flexibly programmable organic virtual networks will be required to securely and dynamically intertwine multiple GenAI-native systems. Features such as cognitive firewalling, sandboxing, and cybersecurity may already be partially provided within this layer.

Orchestration and management layer. This layer is responsible for the low-level scheduling and orchestration of microservices, including service discovery, coordination, and management. In GenAI-native systems, we anticipate an increased need for handling more dynamic workloads, such as asynchronous worker agents proactively scheduling and executing new tasks. Service discovery and coordination frameworks should natively support organic and potentially federated service communication patterns.

Furthermore, we expect a significant increase in additional service management frameworks to better accommodate the requirements of GenAI-native applications. Examples include service-level cognitive firewalling, sandboxing, and coordination across GenAI-native cells or subsystems. Additionally, managed services to facilitate systematic checkpointing of evolved logic and state, as well as native support for flexible, self-reliant agencies, will be essential.

Application definition and development layer. We envision a new hybrid programming model that seamlessly blends passive core logic with active GenAI-native assets. This model, possibly partly inspired by live programming paradigms, should prioritize the easy evolvability of code and data structures as first-class citizens. Additionally, such a hybrid, evolvable programming model should inherently support human-agent codesign, enabling humans and agents to collaborate on common artifacts and services.

Databases and storage services should facilitate versioned checkpointing of code and data to support evolvability, roll-back, and reproducibility. Communication and streaming frameworks should enhance support for asynchronous conversational communication, including UCI and metadata.

Source code management frameworks, along with CI/CD frameworks, should allow running GenAI-native systems to contribute changes in an easy and safe manner. Application and service definition frameworks will become increasingly important to easily specify, manage and control the intent of all GenAI-native assets and subsystems. This will allow other components to verify, constrain, and penalize the behavior of self-reliant and evolvable GenAI assets. Existing artifact registries and hubs should support more organic artifacts. Finally, third-party applications should become GenAI-native.

Observability and analysis tools. Due to the organic and unpredictable nature of GenAI-native systems, they must be designed with robust observability, auditing, and analysis frameworks from the ground up. These frameworks are essential for continuously monitoring, evaluating, and improving the efficiency and effectiveness of all components and interactions across all layers, akin to human organizational paradigms. Existing GenAI logging and observability tools must be enhanced to include more advanced cognitive capabilities and implement effective, comprehensive, and actionable mechanisms to measure and address utility-based sufficiency criteria.

9 Impact

In this section, we provide a brief introduction to the high-level implications and impact of implementing the proposed guidelines and patterns for developing reliable and evolvable GenAI-native software systems. This paper offers only a high-level overview; more comprehensive analyses will be required for each of the aspects discussed.

9.1 Technical Aspects

Computational overhead. Despite some notable exceptions [58, 62], the integration of GenAI into software systems can lead to substantial runtime and performance overhead compared to traditional methods. Moreover, GenAI currently demands a considerable resource footprint, posing significant economic and environmental challenges. When future software systems will evolve to become *GenAI-native* and

become increasingly self-reliant, these challenges are likely to intensify.

In this paper, we proposed several guidelines and patterns to minimize unnecessary reliance on GenAI processing. Additionally, it will be essential to identify the optimal balance for deploying autonomous, self-reliant agents, incorporating cost-benefit analysis and return-on-investment criteria.

Interoperability and scalability. The transition to GenAI-native systems is poised to introduce significant operational challenges, particularly in ensuring seamless interoperability. These systems must effectively communicate with both traditional legacy systems and other emerging GenAI-native systems. Furthermore, scalability will remain a critical concern, as initial GenAI-native implementations will encounter limitations when handling increased workloads and expanding functionalities.

Consequently, the adoption of GenAI-native systems is expected to be incremental, progressively integrating advanced cognitive and autonomous capabilities to enhance system capabilities. Establishing industry standards for GenAI-native systems and communication protocols can facilitate smoother integration and interoperability. In addition, designing systems with modular architecture can help in scaling and integrating new GenAI capabilities without disrupting existing functionalities.

Privacy and security. Future GenAI-native systems must technically address critical privacy and security concerns, such as ensuring data protection, enforcing secure access and authorization mechanisms, and maintaining compliance with regulations. Although we have already outlined several high-level guidelines and patterns, these considerations may inevitably impact the adoption rate of advanced GenAI-native capabilities within existing application architectures.

9.2 User Experience and Adoption

Learning curve. The transition to developing and managing GenAI-empowered software assets presents adoption challenges for software engineers throughout the entire life-cycle, including roles such as architects, designers, developers, testers, and maintenance personnel. Although the design principles and patterns proposed in this paper aim to facilitate this transition, they themselves introduce an learning curve that will require training and experience.

To mitigate these challenges and ease the transition, it is essential to develop user-friendly tools and frameworks. Examples include tools for configuring and managing programmable routers, implementing reflective communication and processing mechanisms, and creating resilience mechanisms tailored to specific use cases or domains.

Additionally, frameworks should be developed to support the easy development and management of, for example, fully functional GenAI-native service cells, the creation of unified conversational interfaces (e.g., the Agora protocol [35]), and

cognitive workflow optimizers. Finally, existing or novel programming paradigms should be integrated and developed to facilitate the programming of assets with predictability and reliability issues, as well as easy programming with malleable, evolving data.

User experience. The adoption of GenAI and agentic services is poised to significantly impact user experience and interaction. Currently, chat-based interfaces are already partly replacing traditional information browsing and retrieval paradigms. Numerous new AI browsers are entering the market, aiming to redefine existing browsing experiences and user interactions [41]. In addition, GenAI-based services promise enhanced personalization and facilitate end-user programming paradigms [33]. Moreover, the emergence of new devices and gadgets is targeted at providing more intuitive interactions with these services.

These innovations will also require a shift in user mentality, yet GenAI's inherent capabilities can hopefully help facilitate this transition. However, ensuring dependability and building trust in these new interfaces and services remain paramount to their successful adoption.

Societal and ethical considerations. GenAI-empowered applications and tools are already significantly influencing people's work dynamics, roles, and daily tasks. Given this societal impact, it is crucial to ensure the responsible use of GenAI in future software systems, including implementing bias mitigation strategies. This paper emphasizes the importance of balancing GenAI-based processing with traditional techniques, with many proposed guidelines and patterns aimed at achieving this balance.

Furthermore, the primary objective of self-reliant GenAI agents should be to enhance cognitive workflows by reducing their involvement in critical paths. This involves thoroughly examining all generated automation and optimization solutions, potentially with human oversight, to ensure reliability and effectiveness.

9.3 Economic and Business Implications

Economic implications. Developing and managing GenAI-native software systems involves several critical economic considerations. One significant challenge is the potential runtime overhead and increased computational footprint introduced by GenAI technologies. Additionally, the development and maintenance costs for GenAI-native systems remain uncertain, particularly in the early stages when suitable tools, frameworks, and best practices will still be under development and refinement.

As discussed, GenAI assets may exhibit variations in behavior when the underlying LLM or GenAI technology is replaced, leading to additional overhead and potential convergence issues that must be addressed. Furthermore, the return on investment (ROI) for deploying self-reliant autonomous agents and self-evolving or personalized software

systems requires careful monitoring and control to ensure economic viability.

Operational efficiency. One of the primary focus areas of this paper is to systematically enhance operational efficiency within GenAI-native systems while simultaneously improving overall reliability. To achieve profitability, continuous productivity enhancements and process optimizations are vital. Consequently, the guidelines and patterns discussed are specifically designed to support these objectives, ensuring that GenAI-native systems operate efficiently, alongside with improving overall reliability.

Market impact. The transition to GenAI-native systems may significantly influence market dynamics. In the short term, early adopters may gain a competitive edge over traditional, more rigid solutions. In the long term, the nature and dynamics of services are expected to undergo radical transformations. For instance, existing business-to-business (B2B) and business-to-consumer (B2C) models may evolve into business-to-agent (B2A) and agent-to-agent models [29], where autonomous agents and services negotiate and interact on behalf of humans or organizations, leading to potentially volatile market dynamics.

GenAI-native web services are also likely to engage with each other more proactively and dynamically compared to their traditional, passive counterparts. As a result, GenAI-native solutions will continuously need to demonstrate and maintain their value in the face of constantly evolving competing alternatives, potentially causing major ripples into the existing market.

9.4 Legal and Regulatory Aspects

Compliance. Future GenAI-native systems must comply with evolving laws and regulations, including intellectual property rights. This paper briefly discussed the implementation of a robust policy language and framework to meet these requirements and facilitate regular audits by external parties. Particularly for self-evolving systems, ensuring continuous compliance is crucial yet challenging. It is imperative to incorporate strong fail-safe mechanisms to address potential failures and maintain adherence to regulatory standards.

Liability. As GenAI-native systems become increasingly autonomous, it is crucial to establish accountability for AI decisions, generated responses, and actions. Preventing these systems from triggering harmful actions or leading humans to make incorrect decisions due to erroneous outputs is essential. Furthermore, the liability of human organizations and enterprises deploying and managing such systems must be clearly defined.

The EU AI Act [12] already sets forth regulations concerning AI and autonomous systems. However, the classification of agentic AI systems as high-risk, requiring full compliance, remains ambiguous [31]. For future GenAI-native systems,

which will integrate both traditional and agentic subsystems, it is vital to delineate the boundaries and interactions between these processing types. Although this might slow the adoption of advanced self-reliant GenAI-native systems, ensuring user safety remains key.

Data governance. Finally, GenAI-native systems will also need to establish and enforce robust data governance protocols, governing data retention, usage, and sharing, both internally and across multiple GenAI-native systems. Beyond traditional policies, it is imperative to define explicit guidelines regarding the utilization of user-specific data and interactions, particularly when enhancing, customizing, or evolving the functionality and operational excellence of such systems and assets. This paper outlines several privacy and security guidelines and patterns. Heightened attention will be required in scenarios involving highly cognitive self-reliant and self-evolving operations.

10 Conclusions and Future Work

In this paper, we advocated for novel design principles and paradigms for building robust and adaptive GenAI-native systems, emphasizing the need for such approach. We discussed core design principles built around five key pillars: reliability, excellence, evolvability, self-reliance, and assurance. Drawing from historical and human perspectives, and centered around several example use cases, we proposed new guidelines, which were crystallized into foundational patterns such as the GenAI-native cell, programmable router, and organic substrate. These aim to reduce the inherent complexity of GenAI-native systems while retaining its potential. We also briefly discussed the GenAI-native software stack, and touched upon the possible impact and implications of future GenAI-native systems from a technical, user adoption, economical, and legal perspective.

Correctly implemented GenAI-native systems should be robust and flexible against both expected and unexpected issues, even beyond those caused by GenAI, with built-in mechanisms for self-reliant evolution. However, poor decision-making can lead to system failure, necessitating shutdown or overhaul. Note that we did not cover all aspects, such as the interplay with robotics and cyber-physical systems. Additionally, most ideas require further validation through experimentation and real-life application, to better understand the impact and benefits of the proposed patterns.

Finally, while current GenAI technologies may not yet fully support the implementation of many ideas presented in this paper, we expect significant advancements in the coming years. We hope this paper will inspire various communities and offer a theoretical framework for developing robust and adaptive future GenAI-native systems.

References

- [1] A2A Protocol. 2025. A2A Protocol – Agent-to-Agent Communication Standard. <https://a2aprotocol.ai/> Accessed: 2025-06.
- [2] Amazon Web Services. 2025. Amazon Bedrock Agents. <https://aws.amazon.com/bedrock/agents/> Accessed: 2025-06.
- [3] Anthropic. 2024. Anthropic introduces the Model Context Protocol. <https://www.anthropic.com/news/model-context-protocol> Accessed: 2025-04.
- [4] Anthropic. 2025. Claude Code: Your code’s new collaborator. <https://www.anthropic.com/claude-code> Accessed: 2025-06.
- [5] Marcus Arvan. 2024. ‘Interpretability’ and ‘alignment’ are fool’s errands: a proof that controlling misaligned large language models is the best anyone can hope for. *AI & SOCIETY* (2024), 1–16.
- [6] Bowen Cao, Deng Cai, Zhisong Zhang, Yuxian Zou, and Wai Lam. 2024. On the worst prompt performance of large language models. *arXiv preprint arXiv:2406.10248* (2024).
- [7] Yunmo Chen, Tongfei Chen, Harsh Jhamtani, Patrick Xia, Richard Shin, Jason Eisner, and Benjamin Van Durme. 2024. Learning to retrieve iteratively for in-context learning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. 7156–7168.
- [8] Yiqun Chen, Lingyong Yan, Weiwei Sun, Xinyu Ma, Yi Zhang, Shuaiqiang Wang, Dawei Yin, Yiming Yang, and Jiaxin Mao. 2025. Improving Retrieval-Augmented Generation through Multi-Agent Reinforcement Learning. *arXiv preprint arXiv:2501.15228* (2025).
- [9] Continue.dev. 2025. Continue.dev: The Open-Source AI Code Assistant. <https://www.continue.dev/> Accessed: 2025-06.
- [10] Cursor. 2025. Cursor: The AI Code Editor. <https://www.cursor.sh> Accessed: 2025-06.
- [11] Emilia David. 2025. ‘Sandbox first’: Andrew Ng’s blueprint for accelerating enterprise AI innovation. <https://venturebeat.com/ai/sandbox-first-andrew-ngs-blueprint-for-accelerating-enterprise-ai-innovation/> Accessed: 2025-06.
- [12] European Parliament and Council. 2024. Regulation (EU) 2024/1689 – Artificial Intelligence Act. Official Journal of the European Union, OJ L 1689, 12 July 2024.
- [13] Martin Fowler. 2024. Patterns of Generative AI. <https://martinfowler.com/articles/gen-ai-patterns/> Accessed: 2025-06.
- [14] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. PAL: Program-aided language models. In *International Conference on Machine Learning*. PMLR, 10764–10799.
- [15] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997* 2 (2023).
- [16] GitHub. 2021. GitHub Copilot: Your AI Pair Programmer. <https://github.blog/2021-06-29-introducing-github-copilot-ai-pair-programmer/> Accessed: 2025-06.
- [17] Google Cloud. [n. d.]. Vertex AI Agent Builder / Agent Engine.
- [18] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948* (2025).
- [19] Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. 2024. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680* (2024).
- [20] Shanshan Han, Qifan Zhang, Yuhang Yao, Weizhao Jin, Zhaozhao Xu, and Chaoyang He. 2024. LLM multi-agent systems: Challenges and open problems. *arXiv preprint arXiv:2402.03578* (2024).
- [21] Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, et al. 2023. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352* 3, 4 (2023), 6.
- [22] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology* 33, 8 (2024), 1–79.
- [23] Sheryl Hsu, Omar Khattab, Chelsea Finn, and Archit Sharma. 2024. Grounding by trying: LLMs with reinforcement learning-enhanced retrieval. *arXiv preprint arXiv:2410.23214* (2024).
- [24] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. 2025. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems* 43, 2 (2025), 1–55.
- [25] James Huckle and Sean Williams. 2025. Easy Problems that LLMs Get Wrong. In *Future of Information and Communication Conference*. Springer, 313–332.
- [26] Masaaki Imai. 1986. *Kaizen: The Key to Japan’s Competitive Success*. McGraw-Hill, New York.
- [27] Anubha Kabra, Sanketh Rangrej, Yash Mathur, Aman Madaan, Emmy Liu, and Graham Neubig. 2023. Program-aided reasoners (better) know what they know. *arXiv preprint arXiv:2311.09553* (2023).
- [28] Daniel Kahneman. 2011. *Thinking, fast and slow*. macmillan.
- [29] Vivek Ladsariya. 2025. The Future Is Agent-to-Agent: A Call for Founders. <https://www.psl.com/feed-posts/the-future-is-agent-to-agent-a-call-for-founders> Published March 31, 2025; accessed 2025-07-09.
- [30] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Kuttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems* 33 (2020), 9459–9474.
- [31] Lexology. 2025. Agentic AI and EU Legal Considerations. <https://www.lexology.com/library/detail.aspx?g=0695e657-7d5a-49e5-9e44-9da10418dc7a> Accessed: 2025-06.
- [32] Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi, and Zhaopeng Tu. 2023. Encouraging divergent thinking in large language models through multi-agent debate. *arXiv preprint arXiv:2305.19118* (2023).
- [33] Geoffrey Litt. 2023. LLMs and End-User Programming. <https://www.geoffreylitt.com/2023/03/25/llm-end-user-programming> Accessed: 2025-04.
- [34] Haoyan Luo and Lucia Specia. 2024. From understanding to utilization: A survey on explainability for large language models. *arXiv preprint arXiv:2401.12874* (2024).
- [35] Samuele Marro, Emanuele La Malfa, Jesse Wright, Guohao Li, Nigel Shadbolt, Michael Wooldridge, and Philip Torr. 2024. A Scalable Communication Protocol for Networks of Large Language Models. *arXiv:2410.11905* [cs.AI] <https://arxiv.org/abs/2410.11905>
- [36] Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2023. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*.
- [37] Microsoft Azure. 2025. Azure AI Foundry Agent Service. <https://learn.microsoft.com/azure/ai-foundry/agents/overview> Accessed: 2025-06.
- [38] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. 2021. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332* (2021).
- [39] John F. Nash. 1951. Non-cooperative games. *Annals of Mathematics* 54, 2 (1951), 286–295.
- [40] Sergii Netesanyi. 2024. Top 10 Cloud-Native Best Practices for Application Development. <https://www.n-ix.com/cloud-native-best->

- practices/ Accessed: 2025-04.
- [41] Casey Newton. 2025. The AI browser wars are about to begin. <https://www.platformer.news/ai-web-browsers-openai-perplexity-opera/> Accessed: 2025-06.
- [42] Dang Nguyen, Viet Dac Lai, Seunghyun Yoon, Ryan A Rossi, Handong Zhao, Ruiyi Zhang, Puneet Mathur, Nedim Lipka, Yu Wang, Trung Bui, et al. 2024. Dynasaur: Large language agents beyond predefined actions. *arXiv preprint arXiv:2411.01747* (2024).
- [43] OpenAI. 2024. Learning to Reason with LLMs. <https://openai.com/index/learning-to-reason-with-llms/> Accessed: 2025-04.
- [44] OpenAI. 2025. Introducing Codex: AI Software Engineering Agent. <https://openai.com/codex/> Accessed: 2025-06.
- [45] OpenAI. 2025. Introducing Operator. <https://openai.com/index/introducing-operator/> Accessed: 2025-04.
- [46] OWASP Foundation. 2025. OWASP Generative AI Security Project. <https://genai.owasp.org/> Accessed: 2025-04.
- [47] Thomas Pyzdek and Paul A. Keller. 2023. *The Six Sigma Handbook: A Complete Guide for Green Belts, Black Belts, and Managers at All Levels* (6 ed.). McGraw-Hill Education, New York.
- [48] Hongzhou Rao, Yanjie Zhao, Xinyi Hou, Shenao Wang, and Haoyu Wang. 2025. Software Engineering for Large Language Models: Research Status, Challenges and the Road Ahead. *arXiv:2506.23762* [cs.SE] <https://arxiv.org/abs/2506.23762>
- [49] Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. 2024. A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv preprint arXiv:2402.07927* (2024).
- [50] Ranjan Sapkota, Konstantinos I. Roulmeliotis, and Manoj Karkee. 2025. Vibe Coding vs. Agentic Coding: Fundamentals and Practical Implications of Agentic AI. *arXiv:2505.19443* [cs.SE] <https://arxiv.org/abs/2505.19443>
- [51] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems* 36 (2023), 68539–68551.
- [52] Amazon Web Services. 2025. What is Cloud Native? <https://aws.amazon.com/what-is/cloud-native/> Accessed: 2025-04.
- [53] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems* 36 (2023), 8634–8652.
- [54] StackBlitz. 2025. Bolt.new: AI-Powered Web App Builder. <https://bolt.new/> Accessed: 2025-06.
- [55] Maira Ladeira Tanke, Mark Roy, Navneet Sabbineni, and Monica Sunkara. 2024. Best practices for building robust generative AI applications with Amazon Bedrock Agents – Part 1. <https://aws.amazon.com/blogs/machine-learning/best-practices-for-building-robust-generative-ai-applications-with-amazon-bedrock-agents-part-1/> Accessed: 2025-06.
- [56] Maira Ladeira Tanke, Mark Roy, Navneet Sabbineni, and Monica Sunkara. 2024. Best practices for building robust generative AI applications with Amazon Bedrock Agents – Part 2. <https://aws.amazon.com/blogs/machine-learning/best-practices-for-building-robust-generative-ai-applications-with-amazon-bedrock-agents-part-2/> Accessed: 2025-06.
- [57] Co Tran, Salman Paracha, Adil Hafeez, and Shuguang Chen. 2025. Arch-Router: Aligning LLM Routing with Human Preferences. *arXiv preprint arXiv:2506.16655* (2025).
- [58] Teng Wang, Wing-Yin Yu, Ruifeng She, Wenhan Yang, Taijie Chen, and Jianping Zhang. 2024. Leveraging large language models for solving rare mip challenges. *arXiv preprint arXiv:2409.04464* (2024).
- [59] Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024. Executable Code Actions Elicit Better LLM Agents. In *Forty-first International Conference on Machine Learning*. <https://openreview.net/forum?id=jJ9BoXAfFa>
- [60] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.
- [61] Scott Wu and Cognition Labs. 2024. Introducing Devin, the First AI Software Engineer. <https://cognition.ai/blog/introducing-devin>
- [62] Kangwei Xu, Ruidi Qiu, Zhuorui Zhao, Grace Li Zhang, Ulf Schlichtmann, and Bing Li. 2024. LLM-Aided Efficient Hardware Design Automation. *arXiv preprint arXiv:2410.18582* (2024).
- [63] Ruoxi Xu, Hongyu Lin, Xianpei Han, Jia Zheng, Weixiang Zhou, Le Sun, and Yingfei Sun. 2025. Large Language Models Often Say One Thing and Do Another. *arXiv preprint arXiv:2503.07003* (2025).
- [64] Quanjun Zhang, Chunrong Fang, Yang Xie, Yaxin Zhang, Yun Yang, Weisong Sun, Shengcheng Yu, and Zhenyu Chen. 2023. A survey on large language models for software engineering. *arXiv preprint arXiv:2312.15223* (2023).
- [65] Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, et al. 2023. Siren’s song in the AI ocean: a survey on hallucination in large language models. *arXiv preprint arXiv:2309.01219* (2023).
- [66] Zihan Zhang, Meng Fang, Ling Chen, Mohammad-Reza Namazi-Rad, and Jun Wang. 2023. How do large language models capture the ever-changing world knowledge? a review of recent advances. *arXiv preprint arXiv:2310.07343* (2023).
- [67] Haiyan Zhao, Hanjie Chen, Fan Yang, Ninghao Liu, Huiqi Deng, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, and Mengnan Du. 2024. Explainability for large language models: A survey. *ACM Transactions on Intelligent Systems and Technology* 15, 2 (2024), 1–38.
- [68] Yueheng Zhu, Chao Liu, Xuan He, Xiaoxue Ren, Zhongxin Liu, Ruwei Pan, and Hongyu Zhang. 2025. AdaCoder: An Adaptive Planning and Multi-Agent Framework for Function-Level Code Generation. *arXiv preprint arXiv:2504.04220* (2025).

A GenAI-native Design Pillars Overview

Figure 5 illustrates the five design pillars and their main direct interdependencies. For each pillar, we also show the relevant subaspects.

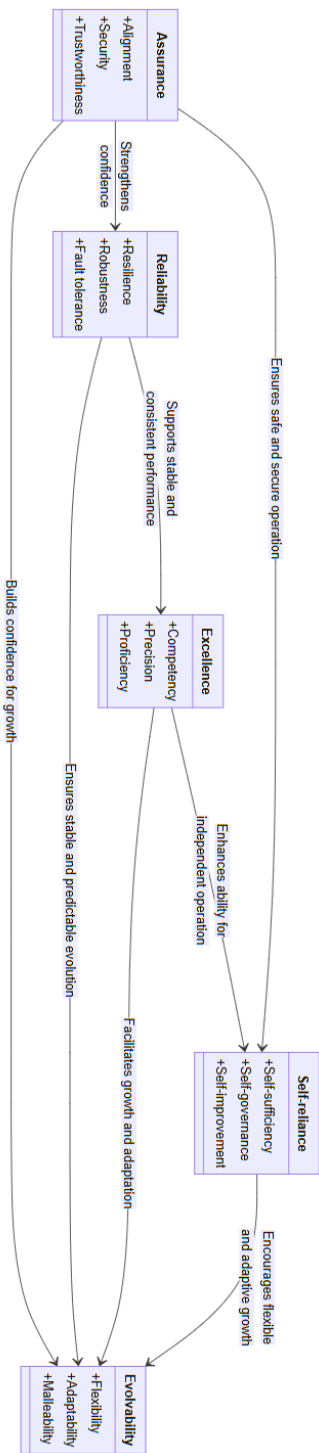


Figure 5. GenAI design pillars and key interdependencies.

B GenAI-native Best Practices Overview

Table 1 presents a comprehensive overview of the GenAI-native software guidelines discussed. It also indicates the core enablers or building blocks we anticipate to be crucial for realizing each guideline. It is important to note that these enabling future frameworks or tools imply human involvement and may entail an initial learning curve.

Table 1. GenAI-native Software Guidelines

Aspect	Best Practice	Description	Core enablers				
Reliability	Design for fault-tolerance and resilience Include verification and mitigation at all levels Restrict scope of unreliability Promote transparency of processing and risks Plan for contingencies Minimize dependency on cognitive processing	Accommodating varying output quality and confidence levels. Embedding multi-level verification and checks in GenAI outputs. Containing unreliability with strategic circuit breakers and fenders. Sharing methods, reasoning, and mitigation strategies. Incorporating multiple techniques & sufficient processing margins. Streamlining processes by minimizing cognitive load.	✓	✓	✓	✓	✓
Excellence	Build upon proven design principles and practices Optimize cognitive workflows Systematic quality verification and retrospectives	Enhancing efficiency and quality through proven methodologies. Boosting efficiency & reliability by optimizing cognitive workflows. Driving excellence through continuous review and improvement.	✓	✓	✓	✓	✓
Evolvability	Promote resilient and adaptive designs Evolve towards reliable and efficient systems Promote consistency over creativity Collective competency ecosystems	Designing GenAI assets to intelligently navigate and resolve issues. Advancing excellence of GenAI systems with adaptive learning. Ensuring stability & smoothness while minimizing inconsistencies. Fostering shared competencies for collective GenAI system growth.	✓	✓	✓	✓	✓
Self-reliance	Balance autonomy with safety and control Include rollback capabilities for autonomous actions Perform regular reviews and updates Maintain visibility into autonomous operations	Implementing checks & balances for autonomous actions. Ensuring failure recovery with rollback & contingency mechanisms. Uncovering hidden issues and learning from the past. Enhancing control via audit trails, self-diagnostics, & oversight.	✓	✓	✓	✓	✓
Assurance	Adopt GenAI security best practices Provide observable and explainable solutions Design for privacy, integrity and trust Assess and mitigate misalignment	Handling cognitive processing with vigilant caution. Making AI decisions traceable and correctable. Containing sensitive data and processing within secure boundaries. Guarding against cognitive biases proactively.	✓	✓	✓	✓	✓
Architectural, Organizational, & Operational	Reimagine cloud-native paradigms Create future-proof designs Relax strict application and system boundaries Impose clear scope and responsibilities Model GenAI agents as digital workers Create self-contained GenAI-native capsules Adopt organic lifecycle management	Evolving from rigid to organic architectures. Designing for continuous evolution and reversibility. Enabling fluid system organization and adaptation. Establishing governance with enforcement mechanisms. Positioning AI agents as collaborative resource managers. Adapting cloud patterns for organic GenAI structures. Enabling bidirectional evolution between GenAI systems & Ops.	✓	✓	✓	✓	✓
	Programmability Develop organic programming paradigms Develop a flexible policy language	Creating modular designs for AI-driven evolutionary programming. Enabling self-regulation & control through flexible policy protocols.	✓	✓	✓	✓	✓
Legend							
			New GenAI-native architecture or design paradigms				
			New GenAI-native programming or deployment frameworks				
			New GenAI-native libraries or tools				
			New GenAI-native programming languages or constructs				
			Involvement of autonomous self-reliant AI agents				
			Human oversight and involvement				

C GenAI-native Design Patterns Overview

Table 2 provides an overview of the initial set of GenAI-native design patterns proposed in this paper. For each, we also specify its primary type (i.e., structural, behavioral, or creational), though some patterns also have a secondary type (not shown here). Structural patterns are mainly concerned with how components are organized and composed into larger structures. Behavioral patterns define how components interact with each other. Creational patterns deal with component creation aspects. We also provide a summary of the key enablers that we anticipate will be crucial for implementing each of these patterns.

Table 2. GenAI-native Software Design Patterns

Aspect	Pattern Name	Main Type	Description	Core Enablers 🧠🦋🔗🎵👁️				
Reliability	Reflective processor	Behavioral	Include meta-cognition & self-regulation mechanisms	✓	✓	✓	✓	✓
	Reflective communicator	Behavioral	Create transparent, redundant conversational flows	✓	✓	✓	✓	✓
	Resilience fender	Behavioral	Mitigate & absorb cascading uncertainty and quality issues	✓	✓	✓	✓	✓
Excellence	Programmable router	Structural	Intelligently route requests to optimal handlers	✓	✓	✓	✓	✓
	Continual self-reflection	Behavioral	Continually self-monitor & assess quality and efficiency	✓	✓	✓	✓	✓
Evolvability	Unified conversational interface	Behavioral	Enable flexible API and conversation-based interactions	✓	✓	✓	✓	✓
	Cognitive workflow optimizer	Creational	Transform cognitive patterns into traditional workflows	✓	✓	✓	✓	✓
	Organic service broker	Structural	Facilitate dynamic and adaptive service dependencies	✓	✓	✓	✓	✓
	Malleable data	Creational	Create & manage flexible, evolvable data representations	✓	✓	✓	✓	✓
Self-reliance	Asset lifecycle management	Behavioral	Agentic units responsible for entire asset lifecycle	✓	✓	✓	✓	✓
	Behavioral policy-driven safeguards	Behavioral	Policy-driven prevention of undesirable behavior	✓	✓	✓	✓	✓
	Infallible fail-safe and recovery	Behavioral	Safely halt and recover from misbehavior of failures	✓	✓	✓	✓	✓
Assurance	Logging & introspection	Behavioral	Meticulously log & introspect all self-reliant actions	✓	✓	✓	✓	✓
	Cognitive screening	Behavioral	Validate cognitive inputs before further processing	✓	✓	✓	✓	✓
	Cognitive firewall	Behavioral	Control cognitive communications between systems	✓	✓	✓	✓	✓
Architectural	Agent sandbox	Structural	Isolate GenAI assets for safety and testing	✓	✓	✓	✓	✓
	Cybersecurity & compliance units	Behavioral	Ensure security and regulatory compliance	✓	✓	✓	✓	✓
	GenAI-native cell	Structural	Create self-contained, adaptive functional units	✓	✓	✓	✓	✓
Legend	Organic substrate	Structural	Manage dynamic cell interactions and evolution	✓	✓	✓	✓	✓
	Reproducible infrastructure	Creational	Enable controlled evolution with reproducibility	✓	✓	✓	✓	✓
				✓	✓	✓	✓	✓

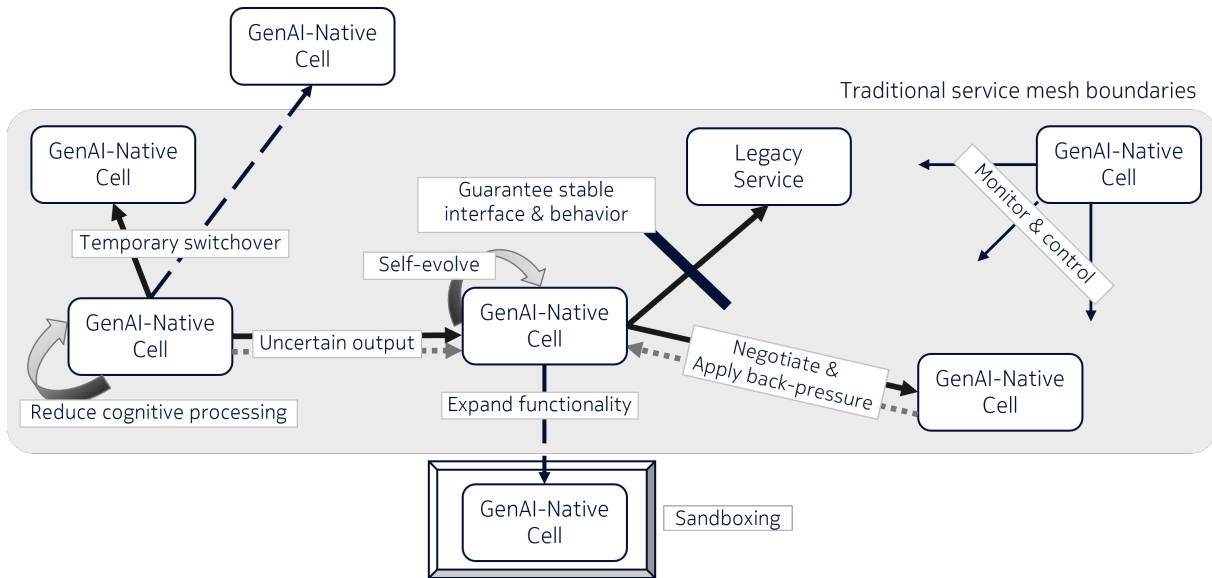


Figure 6. Illustration of an organic substrate within a GenAI-native system, incorporating multiple GenAI-native design patterns to enhance adaptability, resilience, and functionality.

D GenAI-native Architectural Patterns: the Organic Substrate

Figure 6 illustrates the organic substrate of a GenAI-native system, incorporating several key design patterns. Note that this is a condensed representation, and not all GenAI-native systems need to implement or support all capabilities and patterns. Key patterns include:

- **Organic Service Broker:** When a dependent cell is unavailable or cannot provide the requested (extended) functionality, the cell should allow for temporary switchover to other services and potentially expand permanently to include other services, even beyond the boundaries of a traditional service mesh.
- **Reflective Communicator:** Instead of merely sharing computation results, GenAI-native cells also communicate their uncertainty and quality assessments. This may involve prior or posterior negotiations, potentially causing service cells to apply back pressure by refuting the provided results.
- **Resilience Fender:** To prevent cascading uncertainty and handle legacy services, cells may implement additional checks and apply back pressure. This may force upstream cells to redo part of their computation using different methods and involving other services.
- **Cognitive Workflow Optimizer:** To improve efficiency or reduce uncertainty, cells may independently decide to reduce cognitive processing by translating specific workflows into tested core logic.
- **Agent Sandbox:** For sensitive information or untrusted execution, the system may enforce the use of sandboxes to ensure all computation is performed securely, potentially by external service cells, without leaking sensitive information.
- **Safeguarding, Introspection, and Compliance:** Dedicated governance service cells monitor and control all other cells within the organic substrate, intervening when necessary to ensure compliance and security.

These examples highlight some of the high-level interactions across GenAI-native cells within the organic substrate.

E Example Applications



Figure 7. Illustration of the inherent complexity in reliably and flexibly parsing contact information. The input modality, formatting, and critical information availability can vary substantially. A GenAI-native implementation of this function should ideally balance the flexibility needed to handle a wide range of inputs with the downstream reliability and resilience, even when essential information is missing or incorrectly parsed.

E.1 Example GenAI-native Function

At the micro-scale, an example GenAI-native asset could be a function designed to reliably yet flexibly parse general contact information. The general scope of this problem is illustrated in Figure 7. With traditional logic, such a function typically relies on specific input format assumptions to accurately parse raw textual or visual input. Specific logic, such as a combination of OCR, regular expressions, and other predefined rules, is used to extract relevant information and convert it into a predefined output format. While this implementation is very reliable and efficient, it only works correctly for inputs that match the predefined assumptions and may fail or throw an exception for inputs that do not conform. In contrast, a GenAI-first approach, utilizing a multi-modal LLM or agents with proper prompt engineering instructions, can naturally supports a wide range of input formats with minimal effort. However, this approach may be considerably less reliable for common cases, consume more resources, and be slower.

A GenAI-native implementation of such a function could leverage the GenAI-native cell pattern to combine the best of both worlds. It would include a traditional logic-based implementation for common cases and agentic support for handling unrecognized cases. Additionally, a well-designed programmable router at the beginning and end of the GenAI-native function would be responsible for determining the appropriate path before processing the input and evaluating the results. This router would also possibly reiterate the process after the input has been processed, before returning the final result along with an estimation of its accuracy and confidence. Optionally, such a function could self-evolve over time, especially during an onboarding phase before actual deployment, to gradually learn and improve its internal processing capabilities in terms of accuracy, confidence, and efficiency. This means adapting the core logic and programmable router to support more common cases and improving final reviewing efficiency for spotting false positives.

Downstream assets would also need to be adapted to handle outputs that may be incorrect or uncertain. An interesting concept to explore further is the implementation of the Unified Conversational Interface (UCI) approach for such GenAI-native functions. Instead of solely relying on traditional name or location-based parameter passing methods, this approach would enable a conversational interface between functions. This allows one function to provide feedback or further clarifications to the called GenAI-native function, potentially even asynchronously. Similarly, the called function, rather than acting as a passive or synchronous asset, could send an initial quick answer with lower certainty and follow up with a revised answer later, after additional verification. The usefulness and complexity of such interactions at this level may vary and be highly application-specific. Nonetheless, we hope this provides valuable insight into the design and development of applications comprised of such assets.

E.2 Example GenAI-native Application

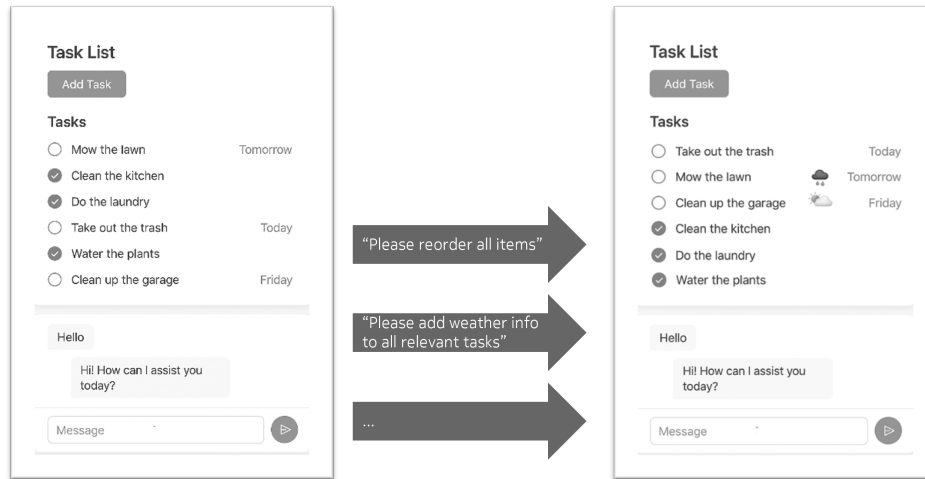


Figure 8. Illustration of a very simple web application, where the user, e.g., via chat, can ask the application for additional functionality, both related to the look-and-feel as well as core functionality. Some of these requests may be volatile, whereas others may need to be made persistent, either in the frontend, backend, or both.

At the service level, an example GenAI-native application could be a simple TODO list web application designed to allow users to edit and view a list of tasks and todos. An example is shown in Figure 8. Let's assume a basic micro-service implementation consisting of a frontend service, a backend service, and a database. A GenAI-native version of this service would enable users to personalize and customize the user experience (UX), behavior, or state of the application at runtime. For instance, a user might ask, via chat, to summarize all tasks or reorder them in a specific way, beyond what the frontend service originally provided through a reactive or declarative UX framework. In this scenario, a GenAI agent would need to customize the look-and-feel of the frontend on-the-fly. Depending on the nature of the user's request, such customizations may be ephemeral, like the earlier example, or more permanent, such as changing the layout or color scheme. In the latter case, the customization would need to be persisted, either on the client or server side. To improve efficiency, consistency, and reliability for such customizations, a DevOps agency might invest time in the background to convert the customization request into new bespoke UX logic.

Alternatively, a user might request to add additional fields to all TODO elements, such as an optional description field or classification labels. This would require changes not only to the frontend UX but also to the backend and storage layer, possibly at the granularity of individual users. Unless the application was specifically designed to support such capabilities, for example via low-code/no-code mechanisms, supporting such evolvability in a reliable and persistent manner via agents is an open challenge. Additional challenges include ensuring that such changes can be made safely and securely (preventing users from triggering destructive or global changes to the backend), staying within the intended scope of the service provider (preventing users from transforming the service into a completely different one), and allowing the service provider to apply global service upgrades without (merge) conflicts.

Finally, a user might want to incorporate additional functionality into the service. For example, they might request to add weather information next to all TODO items tagged with a particular label and receive automatic notifications if the weather could impact those tasks. This would require the service to discover and learn how to communicate with a weather service and integrate it into the application. A service provider might restrict such capabilities, only allowing integration with approved services or charging a premium for bespoke features.

In summary, reliably and efficiently supporting such customizations, even for a trivial application, involves many complex challenges, some of which are still beyond the reach of current GenAI capabilities to be implemented reliably. However, we believe that the GenAI-native design principles and patterns presented in this paper, along with future GenAI-native frameworks and tools, will help decompose these challenges and create robust yet adaptive GenAI-native application services.

E.3 Seamless software Upgrade Example

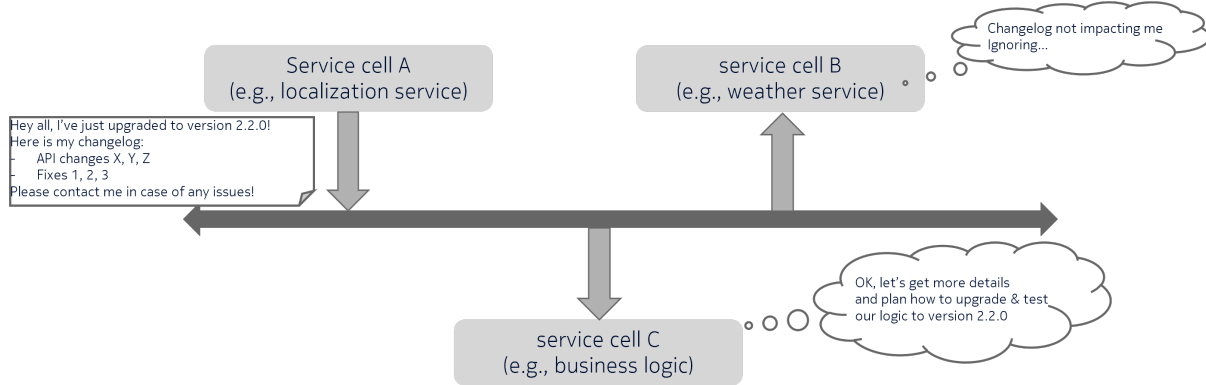


Figure 9. Illustration of a simple upgrade scenario, where a planned feature upgrade is announced via some communication channel to other services. These latter services decide whether this announcement is relevant, which possibly may trigger subsequent upgrades.

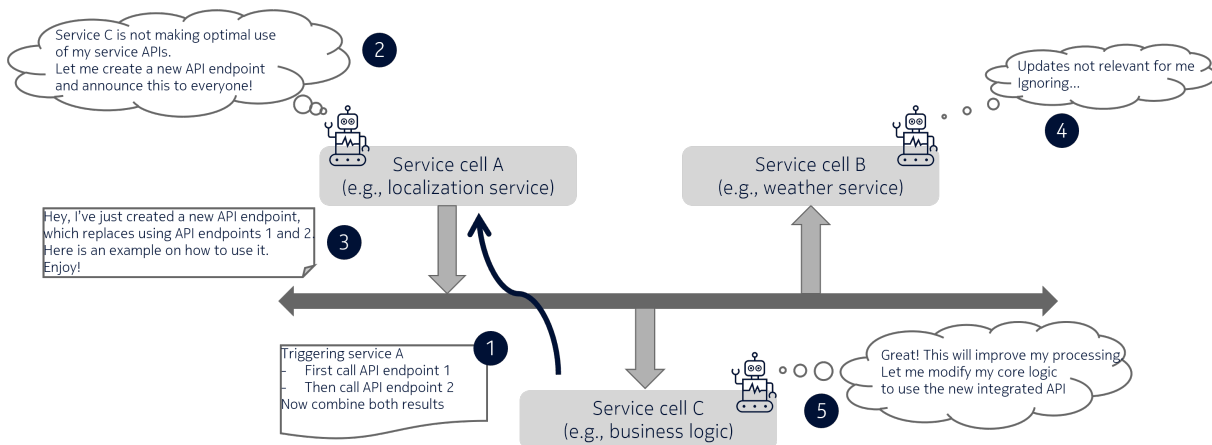


Figure 10. Illustration of a simple upgrade scenario, where a service detects inefficient use of its services, makes changes to its API and functionality, which subsequently is announced and propagated similarly to the baseline scenario.

Future GenAI-native systems may support more seamless and continual software upgrade scenarios compared to traditional software systems. In Figure 9, we show an example flow of how an upgrade of one service may automatically propagate to other service cells. In this example, service cell A announces its changelog over some shared or dedicated communication channel to other service cells. These cells decide whether this upgrade is relevant for them or not. If so, an internal service cell upgrade or evolution may be triggered. Note that this upgrade or evolution could happen offline, as a rolling update, or possibly even completely online in case of minor changes.

In Figure 10, we show a slight variation of the baseline scenario, where the trigger for a service upgrade by service cell A comes from observing how other service cells are utilizing its functionality. In this example, cell A notices ineffective use of its APIs and functionality by service cell C. After having implemented the necessary changes inside its cell, the upgrade is communicated with the other cells, in a similar way as explained earlier. One benefit of announcing the change more globally, rather than directly communicating this update only to service cell C may be that other cells may also show interest concerning the new functionality. In addition, the new API and functionality may also have been the result of multiple independent service cells using cell A in similar not identical manners. As a result, the new API or functionality may be optimized towards all dependent cells, instead of each one individually.