Scientific Journal of the Ternopil National Technical University
2026, vol. 122, no. 2, pp. 70–83,
ISSN: 2522-4433 Print, ISSN: 2522-4441 Online
https://doi.org/10.35784/ visnyk_tntu2026.02.070

Submitted: 2026-03-01 | Revised: 2026-05-12 | Accepted: 2026-05-26

Strategies for LLM-Driven User Interface Generation in Smart Systems: A Comparative Analysis with Decision Guidance

Taras Lypak[1], Halyna Lypak[2], Taras Kramar[3], Oleksij Duda[4]

[1]Ternopil Ivan Puluj National Technical University, Ukraine, taraslypak.work@gmail.com

[2]Ternopil Ivan Puluj National Technical University, Ukraine, halyna.lypak@gmail.com

[3]Ternopil Ivan Puluj National Technical University, Ukraine, kramartar18@gmail.com

[4]Ternopil Ivan Puluj National Technical University, Ukraine, oleksij.duda@gmail.com

Abstract: *Large language models (LLM) provide opportunities to generate user interfaces directly from natural-language intent, but the emerging generative-UI literature remains focused primarily on web and chat contexts rather than smart systems. Smart-system interfaces are characterized by stricter demands, necessitating grounding in concrete device registries, validation against real-world action consequences, and explainability to non-developer users. This paper identifies and compares six distinct strategies currently used for LLM-driven UI generation (direct structured-output prompting, template-augmented generation, retrieval-augmented generation, model-driven hybrid pipelines, constrained generation, and multi-agent decomposition) using criteria tailored to smart-system contexts: output consistency, safety and accuracy validation, explainability, latency, implementation complexity, generalisability, and failure handling. The comparison is then adapted to three representative domains: smart-home consumer IoT, smart industrial supervisory control, and smart cultural heritage (virtual and AR museums, AI-assisted reconstruction interfaces, and smart exhibit displays). For each domain, a primary strategy is recommended, modifications required to meet specific domain constraints are demonstrated, and an adapted reference architecture is presented. A decision matrix and a selection flowchart are developed to guide the selection of an LLM-based UI generation strategy for specific smart-system deployments. The paper closes by outlining several open challenges, including the verification of accuracy-critical generated interfaces and the integration of static generation with runtime adaptation.*

Keywords: *generative user interfaces; artificial intelligence; large language models; machine learning; smart systems; internet of things; cultural heritage;*

1. INTRODUCTION

The rapid development of information technologies provides opportunities for large language models to produce user interface artefacts directly from natural-language intent. This shift is visible across both research and production. In particular, the Jelly system is used to generate malleable UIs from a task-driven data model, which is constructed and updated by the model itself [1]. GenerativeGUI framework serves as a means of reducing conversational cognitive load by emitting HTML widgets inline within a chat dialogue [2]. Similar advancements are evident in industry practice, where tools such as v0.dev, Vercel's AI SDK,

Thesys, and CopilotKit provide deployable generative-UI stacks. This pattern is termed «Generative UI» in the literature, wherein an LLM produces an interface from natural-language intent by combining an intermediate representation with a renderable artifact.

Almost all of these works target one of two deployment contexts: a web page or a chat conversation. In both cases, outputs are primarily informational, the cost of an imperfect interface is low, and the surrounding system is fixed. Recent web-page generation benchmarks are used to score produced pages against expert-built equivalents and to measure artefact quality in isolation. The UI is treated as a finished artefact which worth is judged at delivery, without asking whether the artefact can safely act on a system. These approaches cannot be transferred cleanly to systems where interfaces drive devices or surface scholarly claims.

Smart systems break three fundamental assumptions upon which web and chat generation implicitly rely. First, every UI element must be bound to a concrete entity in a device or asset registry, a synthesized dashboard listing devices that are not actually exposed by the system is considered unusable [3]. Secondly, the generated controls possess action consequences of a physical, financial, or process-level nature, such as opening a valve, calling an actuator, or triggering an automation [4]. Thirdly, the end user is rarely a developer. In particular, homes are configured by residents, plants are operated by specialists, and museums are explored by visitors. These audiences are unlikely to reason about the probabilistic outputs of an LLM without additional assistance. Explainability is therefore treated as a primary design requirement rather than an evaluation afterthought [5].

The inherent limitations of the smart-system UI engineering tradition have long been recognized in the literature. Specifically, a mapping of 212 intelligent-UI studies demonstrated that only ten of them proposed frameworks, and only four proposed methodologies [6]. AdaptUI and related works re-confirm this methodological deficit [3]. An asymmetric situation is observed, where an active generative-UI literature operates with an incorrect scope, whereas the mature smart-system UI tradition offers the appropriate scope but lacks sufficient comparative tools. This aligns with broader methodological concerns regarding the capabilities and constraints of generative AI models operating as data analysts [7]. As a result, practitioners involved in deploying LLM-driven UIs into smart systems receive limited guidance regarding which strategy is most suitable for a specific deployment context.

This paper directly addresses this existing gap. Six strategies for LLM-driven UI generation recur across the recent literature: direct structured-output prompting, template-augmented generation, retrieval-augmented generation, model-driven hybrid pipelines, constrained generation, and multi-agent decomposition. They are compared along seven criteria tuned to smart-system constraints, providing opportunities for producing a comprehensive comparison matrix. The comparison is adapted to three representative domains: smart-home consumer IoT, smart-industrial supervisory dashboards, and smart cultural heritage interfaces. Decision matrix and a selection flowchart are developed to serve as a means of methodological support for practitioners.

The present analysis is conducted as a narrative synthesis rather than a systematic review or an empirical study. In particular, the six proposed strategies represent recurring architectural patterns that have been identified across recent peer-reviewed works concerning LLM-driven UI generation. Developed taxonomy is not considered exhaustive, rather, it captures patterns visible in current practice instead of all theoretically possible construction routes. Three adjacent categories are deliberately treated as cross-cutting concepts rather than separate strategies. Tool-augmented generation is defined as a capability layer that several of the six aforementioned strategies already incorporate (with Retrieval-augmented generation (RAG) being one form, and multi-agent decomposition typically depending on tool access). The human-in-the-loop approach serves as an interaction mode that is orthogonal to the generation

strategy. At the same time, agentic UI generation, in its current usage, is largely collapsed into multi-agent decomposition.

2. BACKGROUND

Smart-system user interface engineering has been organized around a limited number of model-driven reference frameworks since early 2000s. The most influential among them is the CAMELEON framework [8], which is used to decompose UI construction into four fundamental levels: Tasks & Concepts, Abstract UI (independent of platform or modality), Concrete UI (tied to a specific class of interaction surfaces), and Final UI (representing the deployed artefact). Each level is connected to the subsequent one by means of a transformation, and each transformation is, in principle, validatable. This strict separation between abstract and concrete representations is exactly what provides opportunities for a single task model to drive multiple final UIs across various devices, modalities, and contexts. CAMELEON has been adopted, extended, and refined by multiple research groups over the intervening two decades. In parallel, adaptive UIs moved beyond rule-based context handling into reinforcement-learning-based runtime adaptation [9].

Empirical footprint of this tradition remains thin despite the theoretical depth. In particular, a mapping of 212 intelligent-UI studies identified only ten frameworks and four methodologies [6], while the AdaptUI and related works re-confirm this methodological deficit [3]. The literature has the appropriate scope but lacks sufficient comparative tools. LLM-driven generation serves as both an opportunity for creating new construction routes and a methodological hazard due to the absence of established ways to compare them.

Today the rapid development of large language models provides opportunities for opening a distinct construction route. Instead of transforming a pre-designed UI model at runtime, the interface artefact is produced directly from natural-language intent by the LLM itself. The clearest academic instances of this approach are represented by the Jelly system [1], which is used to construct and evolve a task-driven data model from user prompts in order to render the UI from it, and the GenerativeGUI framework [2], which emits HTML widgets inline within a chat dialogue. Therefore, a working understanding is starting to crystallize: a combination of an interface specification, an intermediate representation, and a renderable artefact is produced by an LLM in response to user intent and runtime context.

Two recurring design choices serve as distinguishing features across the recent cohort. The first is the use of structured intermediate representations between the prompt and the artefact is observed, which is exemplified by Jelly's task-driven data model [1]. The second is constraint-based generation. Rather than allowing the LLM to produce free-form output, the generation process is gated by a formal grammar or JSON schema at decoding time [10]. Engines such as XGrammar are currently standardized in serving stacks like vLLM and SGLang [11]. These two threads can be combined to apply constraint-based generation to both the final artifact and the intermediate representation itself.

The terms «generative UI» and «generative interfaces» are used interchangeably in the literature to describe the broader paradigm. The specific construction technique examined in this paper is defined as LLM-driven UI generation, where an LLM, potentially combined with retrieval, schemas, or model-driven transformations, produces UI artefacts from natural-language intent. Intelligent UI adaptation, referring to the older CAMELEON-tradition runtime work, is distinguished from generative UI – adaptation transforms a pre-designed model in response to context, whereas generation produces the initial model. The scope of the current research is defined as UI generation for smart systems, focusing on LLM-driven UI generation in deployments where interfaces drive devices or present scholarly claims.

3. LLM-DRIVEN UI GENERATION STRATEGIES: ANALYSIS AND GUIDANCE

Six strategies for LLM-driven UI generation are most frequently identified across the recent literature. These strategies are differentiated along two primary axes: output grounding (which runs from free-form generation with no external anchor to generation heavily constrained by a registry, template library, ontology, or grammar) and structural enforcement (which runs from post-hoc validation after generation to enforcement applied at decode time). Process complexity is encoded in node size where a larger node signals more pipeline stages or more specialised agents. The strategies are not mutually exclusive, since modern production systems frequently incorporate a combination of two or more approaches.

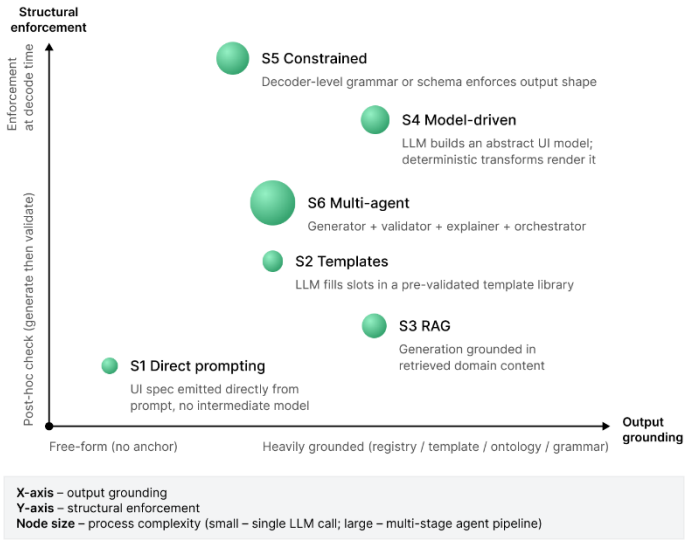

Figure 1. Taxonomy of the six LLM-driven UI generation strategies on a two-axis design space.

Direct structured-output prompting. The simplest strategy relies on the approach where a structured UI specification (typically JSON, YAML, or HTML) is emitted by the LLM directly from a natural-language prompt, without the involvement of any intermediate model or grounding pass. In this case, a prompt and a schema description are accepted by the system as input, while a UI specification is produced as output, which is subsequently converted into a visible artefact by a renderer. GenerativeGUI framework applies this approach within a chat context, emitting HTML widgets inline during a conversation [2]. Industry stacks such as v0.dev and Bolt.new utilize this method as their default mode of operation.

Direct prompting provides the lowest implementation cost and the highest flexibility among the six considered strategies in smart-system context. In particular, this approach serves as a useful means for the rapid prototyping and ad-hoc visualization of unstructured data.

However, specific limitations arise regarding smart-system constraints. In the absence of a grounding step, device names, attribute identifiers, or component types that do not exist in the target system can be invented by the LLM. For instance, a dashboard might reference a specific entity that the registry does not expose, or a control widget may be bound to an unsupported attribute. Additionally, the output structure tends to vary between calls for identical prompts, which inevitably breaks downstream renderers. Direct prompting is considered a starting point rather than a deployment-ready strategy for environments where action consequences are critical.

Template-augmented generation. In this strategy UI components are not generated by the LLM from scratch. Selections are made from a pre-validated template library instead, and specific slots within the chosen template are filled. Since the template library is fixed at the deployment stage, the freedom of the LLM is restricted strictly to slot fillings and the selection of appropriate templates for specific intents.

A practical instance of this approach is observed in the generation of smart-home automation configurations from natural language, where slots in a platform-defined schema are filled by the LLM rather than components being generated freely [15]. A conversational agent developed for smart-home automation operates on the same principle, relying on the LLM to fill slots within a curated vocabulary of platform-supported actions [16]. Similarly, industry stacks such as CopilotKit and Thesys provide curated component libraries that are subsequently composed by the LLM.

For smart systems, predictability at a moderate implementation cost is provided by template-augmented generation. Hallucinated component types become structurally impossible: if a specific component is absent from the library, it cannot be utilized by the LLM. Accessibility, branding, and interaction-pattern compliance can be guaranteed at the template level once and inherited subsequently. The associated cost is rigidity: requirements outside the template library necessitate expanding the library itself rather than re-prompting. Thus, the strategy suits deployments with a bounded component vocabulary better than open-ended creative ones.

Retrieval-augmented generation. RAG grounds to domain content before generation. A retriever, typically backed by a vector database, fetches entity descriptors, schemas, or ontology fragments relevant to the user's intent. The UI specification is produced by the LLM with that retrieved context integrated into its prompt. This grounding is data-side rather than structure-side – the LLM remains free to compose any specification, but references to domain entities are conditioned exclusively on the data returned by the retriever.

For cultural-heritage interfaces, the LLM is grounded in CIDOC-CRM heritage metadata via generated SPARQL queries using a two-stage ontology-path-patterns method [17]. A subsequent comparative study analyzes this ontology-based strategy against LLM-extended and LLM-pure approaches for virtual-museum semantic annotation, identifying specific trade-offs for each [12].

In smart systems if the retriever indexes the device or asset registry, UIs referencing only entities the system actually exposes. The associated trade-offs are non-trivial: the pipeline is bottlenecked by retrieval quality, latency rises alongside retrieval depth, and a maintained knowledge base becomes part of the deployment. Furthermore, RAG handles what the UI references, but not what it does. For example, knowing that «lock.front_door» exists does not equate to knowing whether the device accepts an unlock command from the current user at the given time. Since RAG addresses entity grounding rather than action validity, pairing it with constrained or model-driven strategies is used to mitigate this gap.

Model-driven hybrid pipelines. Instead of producing a final artefact directly, the LLM operates on an abstract UI model. Subsequent to the generation or modification of the abstract model by the LLM, deterministic transformations are utilized to convert it through Concrete and Final UI levels (as established in the CAMELEON tradition) into a renderable output.

The most explicit recent instance is represented by the Jelly system: a task-driven data model is constructed from user prompts by the LLM, which drives UI synthesis through predefined patterns, while users can inspect and adjust both elements [1]. Consequently, the CAMELEON-tradition separation between abstract and concrete UI is inherited by this approach in a contemporary form [8].

For smart systems, the model-driven hybrid provides advantages that direct prompting cannot: validation of the abstract model prior to rendering, and an audit trail from intent to artefact via deterministic transformations. This approach suits regulated or safety-relevant deployments, including cases where explanation necessitates demonstrating why a particular UI was produced. However, the associated cost is the highest implementation complexity among the six strategies. The abstract model must be designed, transformations implemented, and the LLM steered toward valid abstract-model outputs, which itself frequently requires constrained or template-augmented prompting at the model-construction stage.

Constrained generation. At decoding time, the output of the LLM is gated by a formal grammar or JSON schema. Tokens that would violate the constraint are masked out prior to sampling, the produced output is structurally valid against the grammar by construction. The constraint is typically represented by a JSON schema, a context-free grammar describing UI layout primitives, or a domain-specific specification language.

A motivating study based on a survey of 51 industry professionals documents developer demand for structured-output guarantees across compliance reporting, UI generation, and other contexts [10]. The underlying technical substrate has matured rapidly: XGrammar improves substantially on earlier grammar-constrained methods, being currently utilized as the default structured-generation backend in vLLM, SGLang, and TensorRT-LLM [11]. Similarly, this principle is applied to IEC 61131-3 PLC code generation within industrial contexts, resulting in the production of verifiable control programs [13].

Constrained generation for smart systems offers the strongest structural guarantees among the six strategies. However, structural validity is not equated with semantic correctness: a schema-conformant output can still bind the incorrect actuator, reference the wrong device, or specify an alarm threshold inconsistent with deployment invariants. Schema compliance is considered critical for actuator-binding configurations, protocol traffic, and regulatory-audit trails, where malformed output represents a deployment-blocking issue rather than a mere quality defect. The associated limitations are substantial: reasoning quality can be degraded by hard constraints, particularly on schemas with recursive or deeply nested structures. Furthermore, recursive schemas require CFG-based engines rather than FSM-based ones, and the grammar itself serves as a design artefact demanding continuous maintenance. Constrained generation is rarely deployed alone, being naturally paired with template-augmented or model-driven approaches.

Multi-agent decomposition. A single LLM call is replaced by a team of specialized agents: a UI specification is produced by a generator, domain invariants are verified by a validator, a user-facing rationale is created by an explainer, and handoffs are managed by an orchestrator. Other specific roles (such as refiner, critic, or planner) can be additionally incorporated. It should be noted that this decomposition remains functional rather than architectural: an underlying model may be shared among agents, with prompts and tool access distinguishing their specific roles.

In the context of smart systems, the most direct instance is represented by recent industrial-LLM-agent research, where generator-plus-validator decomposition is used to produce controllable automation interfaces, supported by experimental evidence across multiple production-system case studies [14].

For smart systems, this strategy transforms validation and explanation into first-class concerns rather than additional afterthoughts. Generated outputs are passed through validators before reaching the user, while explanations are provided by a dedicated agent instead of being extracted from the prompt that produced the action. The associated costs involve coordination overhead, latency stacking (since each agent necessitates an LLM call), and a non-trivial orchestrator. Multi-agent decomposition suits deployments where validation and explanation are non-negotiable: regulated industrial supervision, scholarly cultural-heritage interfaces, and healthcare.

Strategy comparison. Generic UI creation evaluation frameworks cannot be transferred cleanly to smart-system contexts. Web-page generation benchmarks measure artefact quality purely in isolation. The UI is treated as a finished artefact whose worth is judged strictly at delivery. Such frameworks do not evaluate whether the artefact can act safely on a system, whether its references resolve to real entities, or whether a non-developer end user is capable of understanding the executed actions. The criteria adopted in this study translate the four smart-system constraints (registry grounding, action-consequence validity, explainability, as well as factual or normative correctness) into evaluable dimensions.

For the purpose of this analysis, seven criteria are used. Output consistency measures whether identical inputs produce structurally equivalent UIs across invocations. Safety and

accuracy validation evaluates the degree to which a specific strategy can be paired with explicit checks of generated content against domain invariants prior to deployment. Explainability assesses the strategy's capacity to surface the rationale behind why a particular UI was produced, presented in a form manageable for a non-developer end user. Latency measures the end-to-end response time from intent to renderable artefact, incorporating any intermediate retrieval, validation, or multi-agent coordination. Implementation complexity determines the engineering effort required to deploy and maintain the strategy at production quality, including auxiliary infrastructure (such as template libraries, retrieval indexes, abstract-model definitions, schemas, and orchestrators). Generalisability indicates how effectively the strategy is transferred across smart-system domains without necessitating per-domain reimplementation. Finally, failure handling measures the strategy's behavior when the generation process fails, distinguishing between graceful degradation, blocking failures, or silent corruption.

The developed comparison matrix in Figure 2 displays strategies as rows and criteria as columns, with cell entries providing qualitative ratings. A «Strong» rating indicates that the property is reliably achieved by the strategy under typical conditions utilizing standard engineering effort. A «Moderate» rating implies that the property is achievable but necessitates additional infrastructure, careful design, or pairing with another strategy. A «Weak» rating signifies that the property is achievable exclusively through substantial supplementary measures, or that the strategy is structurally ill-suited. Regarding implementation complexity, the rating scale is inverted: «Strong» indicates a lower level of complexity. It is important to emphasize that each cell reflects what can be achieved by a maximally well-engineered instantiation rather than presenting a fixed benchmark score. Thus, the proposed matrix serves as a comparison of design-space points rather than a conventional leaderboard.

	Consistency	Safety	Explainability	Latency	Complexity (low=good)	Generalisability	Failure handling
S1 Direct prompting	Varies between calls	No validation pass	No rationale produced	Single LLM call	No auxiliary infra	Domain-independent	Silent corruption
S2 Templates	Template-bounded	Structural only	Template intent traceable	Single call + selection	Library to maintain	LIbrary is per domain	Fallback to known templates
S3 RAG	Retrieval variance	Entity grounding only	Provenance if surfaced	Retrieval round-trip	Vector store + index	Transfers if data structured	Graceless on miss
S4 Model-driven	Deterministic transforms	Validated abstract model	Transformation chain auditable	Multi-stage pipeline	Model + transforms to design	Abstract model is per domain	Fails at transformation step
S5 Constrained	Grammar-bounded	Structural guarantee at decode	Grammar trace opaque	Constraint masking overhead	Grammar to design	Grammar is per-domain	Rejects on violation
S6 Multi-agent	Validator enforces baseline	Dedicated validator agent	Dedicated explainer agent	LLM calls stack	Orchestrator non-trivial	Agent roles reusable	Structured failure reports

Strong Moderate Weak Limited

Figure 2. Comparison matrix: strategies and seven criteria.

Three patterns are observed. Constrained generation, model-driven hybrids, and multi-agent decomposition share the top of the safety and accuracy column. Each approach validates the produced artefact (or its abstract model) before rendering through different mechanisms: decoder-level grammar constraints, deterministic model transformations, and dedicated validator agents, respectively. Model-driven and multi-agent methods also share the top of the explainability column, making explanation a first-class pipeline stage. RAG's claim to explainability is architecturally the most direct (since every entity reference traces back to a retrieved source), but it scores one tier below because the delivered explainability depends on whether the implementation surfaces those provenance links rather than merely consuming them within the prompt. Direct structured-output prompting dominates the implementation complexity column by requiring almost no auxiliary infrastructure, paying for this simplicity in consistency, validation, and failure handling. Multi-agent decomposition is unique in trading latency for built-

in validation and explanation. Thus, no strategy dominates across all seven criteria, which explains why the three domain adaptations (smart-home consumer IoT, smart-industrial supervisory dashboards, and smart cultural heritage) each select a different primary strategy.

One important observation concerns how the criteria are weighted in the current literature. Failure handling is considered the criterion most likely to be under-weighted in strategy selection: the majority of published evaluations treat the produced UI as a finished artefact rather than analyzing how the system behaves upon generation failure. Consequently, the three domain adaptations that follow treat it as a primary concern.

Strategy adaptation to smart home consumer IoT. Smart-home interfaces are characterized by a distinctive combination of constraints. The device vocabulary remains diverse but bounded (lights, thermostats, locks, sensors, speakers, appliances). End users are typically non-developers, representing family members with mixed technical literacy. Action consequences are mainly low to moderate, although privacy concerns and action irreversibility elevate risks in specific cases. Furthermore, modalities span voice, text, app dashboards, and increasingly multimodal voice-plus-display agents [4].

The primary strategy proposed here is template-augmented generation, utilizing RAG for personalization against the device registry. The bounded device vocabulary is naturally mapped to a curated template library, while accessibility and brand consistency are more effectively guaranteed at the template level than extracted from free-form LLM outputs. Hallucinated component types are considered particularly disruptive in voice-driven flows, where users cannot easily verify the executed system actions.

This adaptation is structured around three primary components. First, the template library is tied to a device-registry abstraction similar to Home Assistant's exposed-entities model, ensuring components reference entity types and states actually provided by the registry [15]. Second, a goal-oriented reasoning layer is positioned above the template selector, translating user utterances (such as «make it cosier») into concrete template instantiations against available entities. This specific pattern is demonstrated by the Sasha framework [4]. Finally, RAG applied against historical interaction logs and household preferences optionally personalizes template selections without granting the LLM free generative scope.

Figure 3a shows the architecture as layered: user intent is interpreted by a goal-oriented LLM agent; interpreted goals are mapped to candidate templates by a selector; slot values are filled from actually-exposed devices through entity-registry grounding; and outputs are produced by the renderer depending on the modality. Failed grounding triggers a graceful fallback designed to explain why a requested action could not be completed.

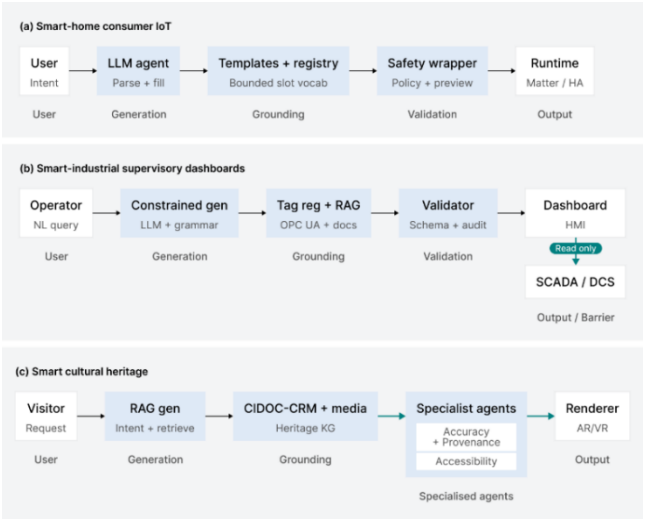

Figure 3. Three domain-adapted reference architectures.

It should be noted that two issues remain unresolved. While privacy-conscious deployments increasingly favor on-device LLM inference, template-selection accuracy and goal-interpretation depth are significantly constrained by small models [15]. Additionally, multi-user households (characterized by different preferences, authority levels, and a shared UI surface) have barely been addressed in the current LLM-driven UI literature.

Strategy adaptation to smart-industrial supervisory dashboards. These specific interfaces cover alarm management, process visualization, natural-language queries against SCADA historians, and operator-instruction surfaces. The underlying control loop, autonomous control agents, and PLC code generation constitute the substrate upon which this case is built. The focus is directed toward the operator's interface with that substrate, ensuring the case remains directly comparable with the smart-home and cultural-heritage interfaces within the other two adaptations.

The associated constraints are considered highly stringent. Operator dashboards are regulated through ISA-101 (HMI design) and ISA-18.2 (alarm management), while surrounding industrial-cybersecurity standards (IEC 62443) impose audit-trail requirements on every operator-facing change. Furthermore, real-time response is strictly non-negotiable. Hallucinated alarm conditions (a fabricated process tag, or an alarm threshold inconsistent with ISA-18.2 priority categories) would be both operationally dangerous and an auditable regulatory event. Since operators are skilled but rarely computer-science-trained, the time pressure during incidents renders UI consistency across shifts strictly critical. The primary strategy involves constrained generation paired with multi-agent decomposition. Constrained generation guarantees that dashboard configurations, alarm bindings, and query results conform strictly to deployment schemas: recent research demonstrates feasibility for IEC 61131-3 in adjacent control-code territory [13]. Multi-agent decomposition introduces the explicit validation and explanation layers required by the regulatory environment, utilizing a generator-plus-validator pattern already established within this domain [14].

This adaptation encompasses three specific engineering requirements: the generator operates under a schema derived from the deployment's HMI configuration model; generated artefacts are verified against ISA-101 invariants (information density, color-coding consistency, navigation hierarchy) by a validator prior to reaching the operator; and a per-change rationale for the audit log is produced by an explanation agent, successfully satisfying both regulatory expectations and operator situational-awareness needs.

The architecture (Figure 3b) positions the operator at the top, integrating generator, validator, and explanation agents within a sequential pipeline above a fixed substrate (control system, SCADA historian, asset-administration shell). This specific substrate is exclusively read by the agents without any write access. Furthermore, failed validation results in the production of a structured failure report rather than a degraded artefact.

Two open challenges remain unresolved. First, integrating LLM-generated dashboard changes with legacy SCADA platforms, whose configuration languages predate JSON-schema-aware tooling, requires bridging layers that remain largely undocumented in public literature. Second, an evaluation methodology for LLM-generated operator HMIs is essentially absent. Established methods (such as eye-tracking, SAGAT-style situation-awareness probes, and secondary-task cognitive-load measures) were originally constructed for static HMIs, and their adaptation to dynamically generated interfaces represents a distinct field of scientific inquiry.

Strategy adaptation to smart cultural heritage. Cultural heritage contexts may initially appear disparate from smart-home and industrial domains. Nevertheless, modern heritage interfaces have become architecturally similar to the other two: virtual tours, AR-augmented exhibits, and adaptive visitor displays operate over structured content acting as a retrieval substrate. Furthermore, multimodal output, context-aware rendering, and asset registries (including 3D-scanned artefacts and ontology-encoded provenance) fulfill the identical architectural roles observed in smart homes and supervisory dashboards. Thus, the primary domain-specific constraint is defined by scholarly accuracy rather than the underlying system architecture itself.

The domain combines three specific constraints that distinguish it from the other two cases. Scholarly accuracy is considered binding. For example, a virtual-museum interface misattributing a Roman fresco to a Greek workshop results in a regulatory and reputational failure, not just a mere annoyance. Multimodal and immersive rendering constitute baseline expectations, with VR, AR, and 3D-scanned assets performing first-class roles, particularly within the context of small local museums where mobile AR solutions enhance visitor interaction [18]. The audience is highly heterogeneous in a way the other cases are not: one interface may serve curious tourists, school groups, and visiting scholars simultaneously, with each group necessitating distinct cognitive levels and specific accessibility accommodations.

The applied stakes can be clearly observed within conflict and disaster contexts. Since 2022, more than 500 Ukrainian heritage sites have been damaged or destroyed, and thousands of high-resolution 3D scans of at-risk monuments and artefacts have been produced by Ukrainian institutions. These scans are accompanied by metadata and provenance records detailing their capture process. Such artefacts map directly onto the proposed architecture: the asset registry is populated by 3D scans. Furthermore, metadata and provenance records form the retrieval substrate from which the generator draws and against which the validator checks. Pre-conflict scholarly descriptions provide the ground truth used by the validator to flag fabrications. Consequently, this specific pattern can be generalized to other at-risk-heritage contexts where digitization necessarily precedes physical loss, supported by machine learning frameworks designed for the preservation and classification of documentary heritage artifacts.

The primary strategy involves the combination of RAG with multi-agent decomposition. This specific case is supported by a recent JOCCH study, wherein knowledge-graph, LLM-extended, and LLM-pure strategies for virtual museums are compared [12]. It should be noted that heritage knowledge is well-structured within standardized ontologies, most prominently CIDOC-CRM, with formal SPARQL infrastructure already deployed in production at major institutions [17]. By applying RAG against these ontologies, verifiable provenance is provided to the LLM, whereas multi-agent decomposition introduces scholarly validation and accessibility adaptation as explicit pipeline stages.

This adaptation comprises four specific components. Initially, the institution's CIDOC-CRM-encoded knowledge graph, alongside its 3D-asset registry and image archive, is indexed by a retriever. Subsequently, multimodal interface specifications combining text, 3D scene descriptions, and audio cues are produced by the generator agent. Scholarly validation follows: provenance for every entity reference is verified by a validator agent, which flags claims exceeding the retrieved sources. The associated failure mode involves the generator producing plausible claims unsupported by sources, such as misattributing a fresco absent in the CIDOC-CRM record, or overstating a reconstruction's certainty level beyond its provenance chain. Finally, accessibility is addressed: language, modality, and contrast are adjusted by an adapter agent to match the specific visitor profile. Consequently, audience heterogeneity is practically managed, allowing identically generated content to render differently for visiting scholars and school groups.

Within the proposed architecture (Figure 3c), the visitor is positioned at the top. Behind the visitor-facing renderer, the four aforementioned agents are arranged in sequence, while the CIDOC-CRM knowledge graph, 3D asset registry, and image archive constitute the fundamental retrieval substrate. Depending on the specific deployment context, the renderer outputs are utilized to feed VR/AR headsets, web browsers, or in-gallery displays.

It should be emphasized that one open issue remains unresolved. Scholarly-validation agents depend strictly on high-quality, structured heritage data, whereas many smaller institutions lack the resources required for its production. The proposed architecture functions effectively in cases where institutions have invested in CIDOC-CRM-compatible knowledge graphs and complete provenance chains. However, for institutions lacking such infrastructure, it remains unclear whether the same pipeline can be productively scaled down or whether a

smaller-footprint variant necessitates an entirely separate design, leveraging existing architectures optimized for smaller documentary heritage databases [19].

Decision guidance. Three deployment characteristics drive the strategy selection process. The first factor is the criticality of action consequences. For instance, a smart-home automation triggering an incorrect light scene differs significantly in consequence from an industrial dashboard misreporting an alarm or a virtual-museum interface fabricating an artifact's provenance. Strategies providing hard structural guarantees (such as constrained generation and model-driven hybrids) and explicit validation (multi-agent) justify their implementation cost when consequences remain high. The second characteristic is the density of domain knowledge in structured form. In cases where a deployment incorporates a curated knowledge graph, an asset registry, or a standardized ontology, RAG serves as an essential strategy, and provenance is inherently established. The third dimension is the tolerance for inconsistency. While certain deployments value exploratory creativity (such as early-stage prototyping or design exploration), others require identical inputs to produce structurally equivalent outputs across all invocations. Consequently, constrained generation and template-augmented strategies represent the consistency-first choices.

The selection logic is represented as a flowchart in Figure 4, which is anchored primarily on the criticality of action consequences. High-consequence deployments are directed toward constrained generation (paired with multi-agent decomposition if auditable validation is required), moderate-consequence instances utilize template-augmented generation (optionally with RAG), while low-consequence scenarios rely on direct prompting. Furthermore, secondary branches incorporate RAG where structured domain knowledge exists, and the model-driven hybrid approach where auditable provenance is necessitated. Although practical deployments frequently combine three or more strategies, the primary anchor is identified by the flowchart.

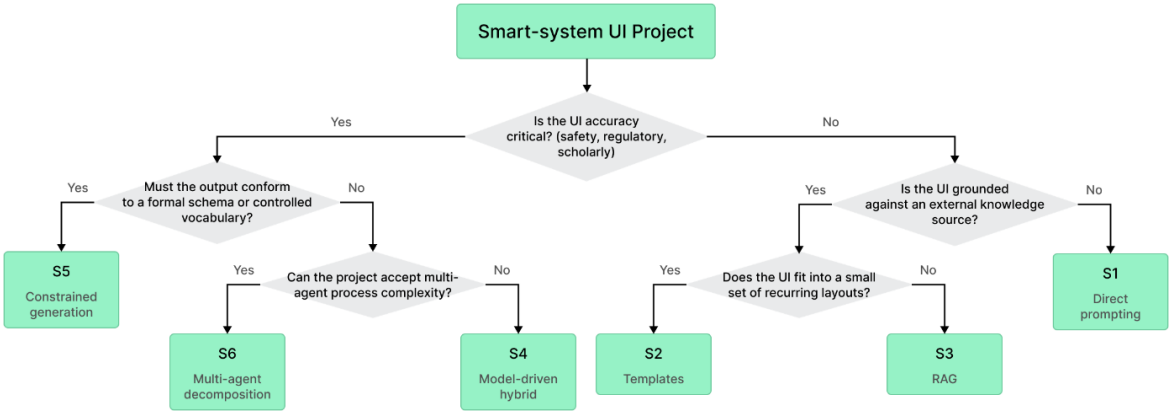

Figure 4. Decision flowchart for strategy selection.

The decision logic can be summarized by five heuristics serving as guidelines rather than invariants, since practical deployments frequently combine strategies with higher-criticality concerns dominating. (1) Direct structured-output prompting alone should be avoided where action consequences are physical, financial, or regulatory. (2) Template-augmented generation is preferred when the component vocabulary is bounded and consistency is prioritized over creative range. (3) RAG should be considered when structured domain knowledge is available to improve grounding and provenance, with costs depending on retrieval infrastructure, data quality, and maintenance. (4) Constrained generation is applied when schema compliance is non-negotiable, paired with semantic validation, since constraint-based decoding guarantees structural validity rather than semantic correctness. (5) Multi-agent decomposition is selected when validation and explanation represent first-class requirements, whereas less complex strategies are preferred where they suffice.

Three recurring pitfalls can be identified in early deployments. The first involves treating constrained generation as a substitute for validation, since schema-conformant outputs can still be erroneous, passing the schema check despite incorrect device references, alarm thresholds, or scholarly attributions. The second pitfall is expecting RAG to resolve domain reasoning. RAG grounds entity references rather than action selection, meaning a model aware of existing devices can still propose unperformable actions. The third issue is under-engineering the multi-agent orchestrator. Since multi-agent decomposition treats coordination as a fundamental engineering problem, deployments chaining multiple LLM calls via informal handoffs inevitably inherit the coordination failure modes documented in the agentic-LLM literature.

4. CONCLUSION

LLM-driven user interface generation has matured to the point where deployment decisions carry real consequences. Current comparative literature tracks a narrower problem: web and chat contexts, where output quality is judged at delivery and an imperfect interface cost little beyond user annoyance. Smart systems sit on the other side of that divide. Every generated UI element must bind to an entity the deployment registry exposes, controls carry physical or process-level consequences, and non-developer end users cannot debug probabilistic outputs without help. The literature lacks systematic guidance for this context.

This paper addresses the gap through three contributions. Six strategies recur across recent work on LLM-driven UI generation: direct structured-output prompting, template-augmented generation, retrieval-augmented generation, model-driven hybrid pipelines, constrained generation, and multi-agent decomposition. They differ along three axes: the LLM's target output, elements fixed before generation, and intermediate representations between prompt and rendered artefact. The strategies are not mutually exclusive; real deployments often combine them, but each is internally coherent enough to compare as a discrete design-space point. Establishing this shared vocabulary is a distinct contribution.

The comparison matrix applies seven criteria shaped by smart-system constraints: output consistency, safety and accuracy validation, explainability, latency, implementation complexity, generalisability, and failure handling. Three patterns emerge. Constrained generation and model-driven hybrid pipelines lead on safety and accuracy validation by verifying the produced artefact before rendering. RAG leads on explainability through provenance, tracing entity references back to a retrieved source. Direct structured-output prompting offers implementation simplicity but compromises consistency and failure handling. No strategy dominates across all seven criteria; choice cannot be reduced to a single technical preference.

The three domain adaptations cover smart-home consumer IoT, smart-industrial supervisory dashboards, and smart cultural heritage. Each translates the abstract comparison into a concrete architecture, identifies the primary strategy required by the domain's constraint profile, and specifies the modifications needed for deployment. The smart-home case requires goal-oriented reasoning above a curated template library to handle non-developer users and heterogeneous device vocabularies. The industrial case pairs schema-constrained generation with explicit validator and explanation agents to meet ISA-101 and audit-trail requirements. The cultural heritage case uses RAG against CIDOC-CRM knowledge graphs with a dedicated scholarly-validation agent to prevent fabricated provenance. A decision flowchart and five selection heuristics consolidate this reasoning, giving practitioners a structured basis for strategy selection.

Three problems remain unresolved. Verification methodology for accuracy-critical generated interfaces is the most pressing. Constrained generation guarantees structural conformance and multi-agent decomposition adds dynamic validation, but neither yields formal verification. Existing evaluation approaches compound this gap: web-page benchmarks measure aesthetics and information delivery, while adaptive-UI evaluation methods address static or rule-adapted

interfaces rather than LLM-generated artefacts. A smart-system-tuned evaluation framework covering registry grounding, action-consequence safety, and auditable provenance remains to be built. The second problem is the relationship between static generation and runtime adaptation. The third is on-device deployment, where local inference on smaller models constrains achievable accuracy, leaving the human-computer interaction consequences unstudied.

The developed taxonomy and decision guidance provide a concrete starting point. The field now requires empirical grounding: evaluation protocols built for smart-system constraints rather than repurposed from adjacent domains, and field studies testing these reference architectures under real deployment conditions.

Author Contributions: Conceptualization: [Halyna Lypak, Oleksij Duda]; Formal analysis: [Taras Lypak]; Writing – original draft preparation: [Taras Lypak, Taras Kramar]; Writing – review and editing: [Taras Kramar]; Supervision: [Oleksij Duda, Halyna Lypak].

Funding: This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Conflicts of Interest: The authors declare no conflict of interest.

Data Availability Statement: All data generated or analyzed during this study are included in this published article.

Declaration of Generative AI and AI-assisted technologies in the writing process: The authors declare that no generative AI or AI-assisted technologies were used in the writing of this manuscript.

REFERENCES

[1] Cao Y., Jiang P., Xia H. (2025). Generative and malleable user interfaces with generative and evolving task-driven data model, in: Proc. of the 2025 CHI Conference on Human Factors in Computing Systems, ACM, New York, 2025. DOI: 10.1145/3706598.3713285

[2] Hojo N., Shinoda K., Yamazaki Y., Suzuki K., Sugiyama H., Nishida K. (2025). Saito K., GenerativeGUI: dynamic GUI generation leveraging LLMs for enhanced user interaction on chat interfaces, in: Extended Abstracts of the 2025 CHI Conference on Human Factors in Computing Systems (CHI EA '25), ACM, New York, 2025. DOI: 10.1145/3706599.3719743

[3] Carrera-Rivera A., Larrinaga F., Lasa G., Martinez-Arellano G., Unamuno G. (2024) AdaptUI: a framework for the development of adaptive user interfaces in smart product-service systems. User Modeling and User-Adapted Interaction, 34, pp. 1929–1980. DOI: 10.1007/s11257-024-09414-0.

[4] King E., Yu H., Lee S., Julien C. (2024) Sasha: creative goal-oriented reasoning in smart homes with large language models, Proceedings of the ACM on Interactive, Mobile. Wearable and Ubiquitous Technologies, 8. DOI: 10.1145/3643505

[5] Amershi S., Weld D., Vorvoreanu M., Fourney A., Nushi B., Collisson P., Suh J., Iqbal S., Bennett P. N., Inkpen K., Teevan J., Kikin-Gil R., Horvitz E. (2019). Guidelines for human-AI interaction, in: Proc. of the 2019 CHI Conference on Human Factors in Computing Systems, ACM, New York, pp. 1–13. DOI: 10.1145/3290605.3300233

[6] Brdnik S., Heričko T., Šumak B. (2022) Intelligent user interfaces and their evaluation: a systematic mapping study. Sensors, 22, 5830. DOI: 10.3390/s22155830

[7] Laitaruk I., Mamchych I. (2025) Some methodological conclusions about the generative AI model data analyst. Scientific Journal of TNTU, vol. 120, no. 4, pp. 130–140. DOI: doi.org/10.33108/visnyk_tntu2025.04.130.

[8] Calvary G., Coutaz J., Thevenin D., Limbourg Q., Bouillon L., Vanderdonckt J. (2003) A unifying reference framework for multi-target user interfaces. Interacting with Computers, 15, pp. 289–308. DOI: 10.1016/S0953-5438(03)00010-9.

[9] Todi K., Bailly G., Leiva L. A., Oulasvirta A. (2021). Adapting user interfaces with model-based reinforcement learning, in: Proc. of the 2021 CHI Conference on Human Factors in Computing Systems, ACM, New York, 2021. DOI: 10.1145/3411764.3445497.

[10] Liu M. X., Yang J., Li T. J.-J., Reza K. K., Terry M. (2024). We need structured output: towards user-centered constraints on large language model output, in: Extended Abstracts of the 2024 CHI Conference on Human Factors in Computing Systems (CHI EA '24), ACM, New York, 2024. DOI: 10.1145/3613905.3650756.

[11] Dong Y., Ruan C. F., Cai Y., Lai R., Xu Z., Zhao Y., Chen T. (2025). XGrammar: flexible and efficient structured generation engine for large language models, in: Proc. of Machine Learning and Systems (MLSys 2025), 2025. https://doi.org/10.48550/arXiv.2411.15100

[12] Vasic I., Fill H.-G., Quattrini R., Pierdicca R. (2025). Knowledge graphs vs. large language models: competitors or partners in supporting virtual museums, ACM Journal on Computing and Cultural Heritage. DOI: 10.1145/3756016.

[13] Fakih M., Dharmaji R., Moghaddas Y., Quiros Araya G., Ogundare O., Faruque M. A. Al (2024). LLM4PLC: harnessing large language models for verifiable programming of PLCs in industrial control systems, in: Proc. of the 46th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '24), ACM, New York, pp. 192–203. DOI: 10.1145/3639477.3639743.

[14] Xia Y., Jazdi N., Zhang J., Shah C., Weyrich M., Control Industrial Automation System with Large Language Model Agents, in 2025 IEEE 30th International Conference on Emerging Technologies and Factory Automation (ETFA) (pp. 1–8). IEEE. DOI: 10.48550/arXiv.2409.18009.

[15] Birkmose R., Reece N. M., Norvin E. H., Bjerva J., Zhang M. (2025). On-device LLMs for home assistant: dual role in intent detection and response generation, in: Proc. of the Tenth Workshop on Noisy and User-generated Text (W-NUT 2025), ACL, 2025, pp. 57–67. https://doi.org/10.18653/v1/2025.wnut-1.7.

[16] Gallo S., Paternò F., Malizia A. (2024). A conversational agent for creating automations exploiting large language models, Personal and Ubiquitous Computing, 2024. DOI: 10.1007/s00779-024-01825-5.

[17] Mountantonakis M., Tzitzikas Y. (2025) Generating SPARQL queries over CIDOC-CRM using a two-stage ontology path patterns method in LLM prompts. ACM Journal on Computing and Cultural Heritage, 18 (1), pp. 1–20. DOI: 10.1145/3708326.

[18] Lypak H., Kunanets N., Veretennikova N., Matsiuk H., Kramar T., Duda O. (2023). An Information System Project Using Augmented Reality for a Small Local History Museum, in: 2023 IEEE 18th International Conference on Computer Science and Information Technologies (CSIT), Lviv 2023, pp. 1–4. ISSN 27663655. DOI: 10.1109/CSIT61576.2023.10324194

[19] Lypak H., Lypak T., Kunanets N. (2024) Designing a Machine Learning-Based Information System for Preserving and Classifying Documentary Heritage Artifacts, Herald of Khmelnytskyi National University. Technical Sciences, vol. 339, no. 4, pp. 176–182. ISSN 2307-5732. DOI: 10.31891/2307-5732-2024-339-4-29.