Journal of Computer Science and Technology Studies
ISSN: 2709-104X
DOI: 10.32996/jcsts
Journal Homepage: www.al-kindipublisher.com/index.php/jcsts

Agent-Generated User Interfaces: An Evidence-Classified Mapping of A2UI, GenUI, and Interoperable Agent-User Interaction

Netra Nandkumar Awate

Northeastern University, Boston, Massachusetts, USA

Corresponding Author: Netra Nandkumar Awate**, E-mail:** netra.awate1@gmail.com

ABSTRACT

Large language model agents create user interfaces for interaction with users at runtime. Evidence for these systems is spread across research papers, protocol specifications, engineering specifications, and other documents. This paper maps what is known about A2UIs, specifically A2UI and GenUI, and the strength of the evidence supporting each claim. A validated corpus of peer-reviewed papers, preprints, and grey literature was systematically mapped for each layer of technology, generation time, contribution to the A2UI, type of research performed, and class of evidence. The application dimension includes content automation and commerce. Technical and human evaluations focus mainly on code-generating interfaces. Declarative, catalog-based protocols have been evaluated mainly with regard to their specifications. There is limited evidence for incident response, multi-agent composition, mobile degradation, and cross-renderer interoperability. The mapping produced a repeatable coding scheme and a research agenda for semantic authorization, backend-to-UI state contracts, and governed commerce surfaces.

KEYWORDS

Agent-Generated User Interfaces, Generative UI, A2UI, Multivocal Systematic Mapping, Interface Governance

ARTICLE INFORMATION

ACCEPTED: 14 October 2026 **PUBLISHED:** 14 November 2026 **DOI:** 10.32996/jcsts.2026.8.15

1. Introduction

Traditional software interfaces are largely authored in advance: designers define the available screens and interactions, and users navigate within those predefined structures. LLM-based agents can disrupt that relationship by planning, calling tools, and acting on intermediate results within a single session [1]. Agent-generated interfaces can instead be composed at runtime according to the task, tool results, and current interaction state. Text-only interaction can be less efficient for tasks requiring comparison, filtering, structured entry, monitoring, or explicit approval controls, which helps explain why runtime interface generation moved rapidly from demonstration to engineering practice [2], [3].

Details are not yet settled and may differ between communities, but a generative UI could involve a model generating HTML and scripts, an agent picking widgets from a component catalog, or a host app exposing agent-generated surfaces in otherwise static screens. Some systems treat the generated surface as a task-oriented data model, not as markup [4]. Others evaluate the generation of interfaces from queries [3] against conversational baselines. Since these approaches differ in their assumptions about who is responsible for rendering, these different systems have very different security, portability, and maintenance properties.

A2UI provides a useful case study for distinguishing assumptions, as the protocol encodes user-interface intent as JSON messages that a client renders using a trusted catalogue of components, and as such, the agent need not deliver executable interface code [5]. The protocol's current production version, per the project's own specification labeling, is 0.9.1, with v1.0 as a release candidate adding typed function calls across the agent and renderer boundary [6]. A2UI sits within a broader agent protocol ecosystem rather than being one of a fixed set of equivalent interfaces: A2A supports agent-to-agent communication [7], AG-UI supports runtime event and state exchange between agents and user-facing applications in a web browser or on a mobile device [8], and A2UI itself represents structured user-interface intent for client-side rendering. A cross-platform renderer also exists through the GenUI SDK for Flutter, an alpha-stage SDK that composes interfaces from an application's own widget catalog [9].

Several included records examine high-interaction or transactional interfaces, including commerce-related applications. Prior work has studied how high-traffic consumer content platforms merge media, product catalogs, and interactive purchasing into a single interface [10], and has examined AI-driven presenters and interaction tools used to support purchase intention [11]. An agent

Copyright: © 2026 the Author(s). This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC-BY) 4.0 license (https://creativecommons.org/licenses/by/4.0/). Published by Al-Kindi Centre for Research and Development, London, United Kingdom.

composing an interface around a product, a content workflow, or a user's question would need to inherit the reliability, governance, and consistency obligations of the underlying transactional system.

Engineering work on the protocols has outpaced the evidence; specifications, repositories, and SDK documentation describe things that can be expressed with these protocols but do not indicate that a generated surface is authorized, reproducible, or semantically equivalent across renderers. Academics have frequently posed interface generation problems assuming no knowledge of the protocol, transport, and state layers that production systems assume. However, no clear evidence-classified mapping yet exists of protocol to runtime protocol, renderer, backend state, and application settings for each protocol-renderer pair.

This paper addresses that gap through a multivocal systematic mapping of agent-generated user interfaces, A2UI and GenUI, to characterize the state of evidence in this area. The mapping does not claim superiority of one protocol or introduce a new user experiment. Instead, the mapping records what has been proposed, built, evaluated, and documented; it grades the strength of that evidence; and it identifies where primary research is most needed. The work makes four contributions to this line of research.

(i) A distinction between runtime agent-driven interfaces, AI-assisted interfaces created at design time, and agents that operate human-facing graphical user interfaces.

(ii) A multivocal mapping protocol in which specifications and engineering documentation are separated from peer-reviewed evidence.

(iii) A reproducible coding scheme across technical layer, generation time, backend delivery, state treatment, application dimension, and evidence class applied to a verified corpus, shared as supplementary material.

(iv) A research agenda on evidence-graded governance, observability, multi-agent provenance, mobile degradation, cross-renderer conformance, backend-to-UI state contracts, and governed commerce surfaces.

2. Background and Related Work

2.1 Agent-Generated UI and GenUI

Agent-generated UIs are those that an LLM or agent generates, selects, or composes after the start of an interaction. UIs found in the corpus and related literature largely conform to one of three architectural patterns. The code generator pattern makes the model generate HTML, scripts, or framework components and offers the greatest expressiveness, leaving validation, sandboxing, and reproducibility to the host. The catalog-driven pattern makes the model emit a structured description that a renderer can handle using a set of components already vetted by the host. The hybrid pattern makes the host application a fixed point with agent surfaces, state updates, or action handlers only at specific locations.

The corpus provides some supporting evidence for runtime generation. In an LLM-generated interface evaluated against an expert-built interface at scale, the expert-built interface was preferred overall, but the generated interface was "at least comparable in 50% of cases" [2]. A separate evaluation of generated interfaces and conversational messages reports "up to a 72% improvement in human preference" [3]. These results indicate that runtime interfaces are promising but do not directly evaluate the safety of a specific delivery model, since both studies generate markup or interactive pages. Such distinctions are important because a protocol, a renderer, an orchestration layer, and a host application resolve different problems, and a finding for one may not hold for another.

2.2 The A2UI Declarative Protocol

In A2UI, a scenario is defined by a sequence of messages that create a surface, set its components, update its data model, and destroy it when it is no longer needed [5]. Components are selected from a catalog agreed to by both client and server developers, with strict typing rules for such catalogs specified in the standard [5]. Since the client owns the final render, it can map the abstract components to native OS widgets, retaining control over branding, layout, accessibility, and action execution. The v1.0 candidate illustrates this need: calls may be made on the renderer, the agent, or both, and the renderer is responsible for enforcing this boundary at runtime [6].

While the declarative model resolves risks from arbitrary executable code passed over the interface, the agent can still be misled or coerced into requesting the wrong approval label, binding a data control to the wrong record, or displaying sensitive information to unauthorized users. Indirect prompt injection research on LLM-integrated applications, offered here as adjacent rather than A2UI-specific evidence, shows that tool output can steer an application toward an attacker's goal without requiring a compromise of the application code [12]. This suggests an analogous risk for agent-generated interfaces, where data contained within the surface itself could become an attack channel, with code-execution safety and interaction safety treated as separate dimensions of the security space. With these considerations in mind, the coding scheme in Section 3 can record these dimensions separately.

2.3 Related Protocol Layers

A2UI operates within a broader ecosystem of agent, tool, event, and transport protocols: A2A defines how agents discover and communicate with each other. The first stable A2A release is v1.0.0 [7]. AG-UI outlines how agents interact with the applications the user is communicating with. It relays streamed messages, tool calls, interrupts, and state events [8]. Agents interact with tools and contexts via the MCP using a plugin architecture. A recent lifecycle analysis delineates the risks in creating, deploying, operating, and maintaining the servers [13]. A2UI may run over several of these layers, which serve distinct architectural purposes:

A2A for agent-to-agent coordination, AG-UI for agent/application event and state exchange, and MCP for tool and context access, with delivery carried over transports such as server-sent events, JSON-RPC, WebSockets, and REST [5]. Layering protocols in this way introduces a problem that none of these specifications addresses. When UI descriptions, state changes, permissions, and actions cross protocol or framework boundaries, they can be semantically incoherent while remaining syntactically valid.

2.4 Backend APIs, Data Binding, and State Synchronization

Although A2UI separates the renderer from the systems providing application data, the binding semantics mean that how a backend provides information considerably affects runtime behavior. Component properties are tied to paths in the surface data model using JSON pointers [5]. An update message replaces a value at a given path or, if a path is not specified, the entire model. The various input components, such as text fields, sliders, checkboxes, and date pickers, automatically configure two-way binding so that user changes on the renderer side immediately update the model. The state layer is therefore not merely a presentation-level representation but also the boundary for synchronizing upstream application records, agent reasoning, local user edits, and rendered controls.

This boundary is affected by the delivery mechanism selected for the backend. The A2UI specification notes that REST is appropriate for simple cases but does not support streaming, and messages must be delivered in the order in which they were created [5]. AG-UI identifies snapshot events, which send a complete agent state for cases like session start and interruption recovery, and delta events, which send JSON Patch operations in a sequence for small changes needed frequently [8]. A request-response pattern suits bounded operations such as those requiring an authoritative snapshot before a surface change. Event streams are a natural fit for long-lived computations with intermediate results, though they raise questions of ordering, convergence, and reconciliation. The replicated-storage research community has a vocabulary to describe these properties, e.g., eventual consistency, where replicas reach the same state if they receive the same updates [14]. The current specifications leave it to the implementers to decide whether A2UI-style surfaces require such guarantees or can use just a single authoritative backend.

The protocol semantics suggest, as a design implication rather than an established finding, that bindings and surfaces may be more resilient to incremental updates when they use stable identifiers for entities, predictable semantics for fields, and backend update granularity matching that of the surface; comparative evidence for these claims is still needed. Open questions that follow from this implication include how authorization and business validation are enforced on the server when controls are dynamically composed by an agent, how systems should resolve local edits that go out of sync with remote ones, how replayed events are handled after a connection is reestablished, and which component takes priority over which field.

2.5 Adjacent Reviews and the Remaining Gap

The closest secondary study on intelligent user interfaces followed a broader definition, finding, across 211 studies, that "81% reported partial or complete improvement in usability or UX" but identifying no evaluation tools or metrics for IUI. Early work on model-based IUIs focused on explicitly modeling the interface adaptation process with models and adaptation rules. This approach anticipated the catalog-plus-data-model approach for contemporary protocols but did not deal with LLM agents. One peer-reviewed study compared LLM-driven GUI layout search with practitioners' rankings in a controlled environment [17]. Among the secondary studies identified in this search, none jointly examined the runtime protocol, renderer, state, and governance layers that A2UI introduces. The mapping therefore covers only the overlap between A2UI and these earlier systems.

3. Materials and Methods

3.1 Study Design and Research Questions

Effect-oriented systematic reviews were considered inappropriate for this nascent research area given the heterogeneity of evidence. In contrast, mapping studies focus on the distribution of research and evidence rather than aggregating effect sizes. The procedures followed the software-engineering mapping guidelines for scoping, search, screening, keywording, and classification of the material [18]. Much of the material is grey literature, including protocol specifications, SDK documentation, and engineering reports. A multivocal design was adopted for this work, following guidelines that allow the retention of grey literature in a systematic review if it is identified, assessed, and reported separately from peer-reviewed literature [19], [20]. No new user experiment, prototype evaluation, or participant data collection was performed.

The analysis addressed six research questions adapted from standard mapping dimensions.

RQ1. Publication landscape: How has agent-generated UI work changed over the years, by source type, and by generation time?

RQ2. Technical focus: To what extent does the work discuss the various technical layers (declarative protocols, transport and orchestration, renderers, code generation, governance, and backend data integration)?

RQ3. Contribution type: What types of contributions exist in the area, such as protocols, SDKs, frameworks, empirical evaluations, benchmarks, case studies, or conceptual contributions?

RQ4. Research type: Which research strategies are represented in the corpus, including solution proposals, validation research, evaluation research, experience reports, and methodological studies?

RQ5. What types and levels of empirical validation support claims about usefulness, safety, interoperability, and production readiness, and how reproducible is that evidence?

RQ6. Research gaps: What are the most important research gaps in the literature regarding enterprise workflows, DevOps and incident response, multi-agent systems, mobile applications, cross-framework interoperability, and backend-fed data and commerce interfaces?

3.2 Search Strategy and Sources

The search combined technology, mechanism, and context terms using the following core search string:

("A2UI" OR "Agent-to-UI" OR "Agent to User Interface") OR ("GenUI" OR "generative UI" OR "generative user interface") OR ((LLM OR "large language model" OR "AI agent") AND ("user interface" OR UI) AND (generat* OR dynamic OR "agent-driven" OR malleable))

Supplementary search strings were also tested, covering prompt injection, deceptive design, component catalog, rendering, accessibility, dashboard, enterprise approval, incident response, multi-agent system, mobile application, interoperability, backend delivery, streaming, data binding, state synchronization, e-commerce, and live commerce.

Literature reviews were conducted using the ACL Anthology, the ACM Digital Library, the IEEE Xplore database, SpringerLink, ScienceDirect, the NeurIPS proceedings, arXiv, and Google Scholar. For publisher pages that could not be accessed, the metadata were cross-checked with Crossref or author uploads. Engineering sources were identified from protocol websites, SDK documentation, and engineering documents describing a single technical design. Backward and forward snowballing from anchor papers and specifications was conducted. The search window spanned January 2023 to 25 September 2026, covering the period during which tool-using LLM agents and runtime protocols emerged. Sources published before this window and cited elsewhere in the paper (e.g., foundational intelligent-UI and mapping-methodology references) are treated as contextual or methodological background and are not counted in the corpus statistics reported in Section 4.

3.3 Inclusion, Exclusion, and Verification

A record was included if it primarily addressed LLM- or agent-generated, selected, or composed user interfaces, a protocol or SDK that uses such user interfaces, or the evaluation of such user interfaces in one or more LLMs. The record was also required to be in English, have publicly named authors or owners, and provide either the full text or a stable technical artifact. Studies evaluating code generation for design-time interfaces were retained as a separately coded adjacent class, since they inform interface-generation capability but are not direct evidence for runtime agent protocol behavior.

The corpus excluded agents that act on human-generated graphical user interfaces or that generate code without a user-visible interface, non-agent UI adaptation, agent security, or agent coordination without a user-visible interface, duplicates of a product, and marketing claims without a technical basis. Protocol surveys and security studies without an interface (otherwise used as part of the contextual material) were not included in the corpus.

The study selection process proceeded in five stages (Figure 1). Of 100 candidates located through database search and snowballing, 15 were excluded because their metadata could not be verified at a stable landing page: 10 titles appeared only on a third-party generative-UI listing site, and the remainder comprised DashChat, PromptInfuser, a vendor architecture blog, an accessibility preprint, and an agent-design guidance page. The remaining 85 candidates were verified at their publisher page, Crossref record, arXiv abstract page, or documentation page. Two specification-version-specific entries for A2UI (v0.9.1 and the v1.0 release candidate) were then merged into the single family record already represented in the corpus, reducing the pool to 83. Full eligibility screening against the inclusion and exclusion criteria in Section 3.3 then removed 24 records, for reasons recorded in the exclusion log, leaving a final included corpus of 59 records: 24 peer-reviewed, 20 preprints, and 15 grey-literature records. The full corpus, exclusion log, and all coding decisions are available in Supplementary Table S1. The two merged A2UI specification versions (v0.9.1 and the v1.0 release candidate) are treated as a single artifact family for corpus-counting purposes; the production version (v0.9.1) determines the record's year and feature coding, while version-specific differences (such as the v1.0 candidate's typed function-call boundary) are preserved in the coding notes rather than the main count.

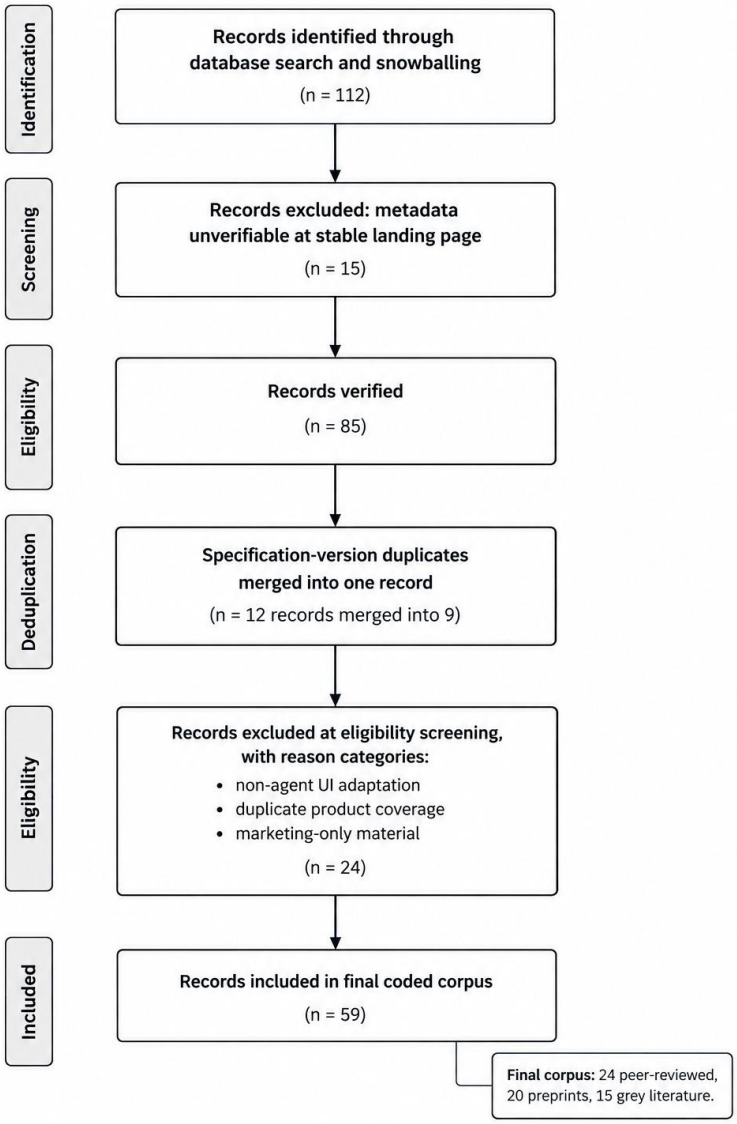


Figure 1: PRISMA Study-Flow Diagram

3.4 Evidence Classification

Given the mixture of source types, evidence class was coded separately from contribution type (see Table 1). Each record was assigned the evidence class representing the type and scope of validation provided; these classes are not interpreted as a universal ordinal quality ranking, since different classes answer different questions. Three rigor flags were also coded: whether the paper was peer-reviewed, whether the authors provided code or data, and whether they discussed threats to validity. Deployment claims were coded E2 only if the deployment was described in an inspectable report.

Table 1. Evidence classes used in the mapping

Class	Typical sources	What it can support	What it cannot establish
E0: Specification or position	Protocol specifications, vision and position papers	Design intent, protocol concepts, stated capabilities	Usability, safety in practice, production success
E1: Illustrative demonstration	SDK documentation with examples, prototypes, demos	Feasibility and illustrative workflows	General performance or adoption
E2: Case study or experience report	Documented deployments, engineering reports	Contextual implementation lessons	Generalizable causal conclusions
E3: Technical evaluation	Benchmarks; quantitative evaluations of fidelity, validity, security or cost	Measured performance on defined tasks	Human benefit unless externally validated

Class	Typical sources	What it can support	What it cannot establish
E4: Human or UX evaluation	User, preference, accessibility and trust studies	User-facing effectiveness and experience	Generalization beyond the studied sample and task

Evidence classes were interpreted relative to the claim under discussion: for example, a human evaluation (E4) supports claims about user-facing outcomes but does not by itself establish protocol safety or production reliability, just as a technical evaluation (E3) supports performance or conformance claims but not user benefit.

3.5 Data Extraction and Coding Scheme

Five core analytical facets were applied to each record (Table 2), supplemented by metadata and treatment flags. For particular records, multiple codes were assigned within a facet because the record spanned several layers or dimensions. Facet A is technical, and facet B is application-oriented. The six original protocol dimensions of design were retained. One was added for content automation and commerce interfaces. The latter reflects an expansion of agent-composed surfaces into shopping, creator, and live-commerce contexts where errors in interfaces have financial downsides and user behavior and transaction risks are closely tied. Facet C records the contribution type, facet D the type of research according to software-engineering mapping practice [18], and facet E the class of evidence. Additional fields record the year, source type, generation time, and treatment of accessibility, security or governance, and state synchronization. Three further fields (synchronization mechanism, agent topology, and portability) are proposed extensions for future use of the scheme and were not populated in this mapping; they are listed in Table 2 for completeness but are not part of the current analysis.

The six original protocol-design technical-layer and application-dimension categories were defined a priori from the protocol specifications under review; the content automation and commerce application dimension emerged during keywording and was added after initial screening, reflecting an expansion of agent-composed surfaces observed in the corpus.

Table 2. Coding scheme and its application in this mapping

Facet or field	Categories	Status
A. Technical layer	A1 declarative UI protocol; A2 transport or orchestration; A3 renderer or SDK; A4 code-generating UI; A5 hybrid, harness or governance; A6 backend data or API integration	Applied
B. Application dimension	enterprise approvals and governance; DevOps and incident response; multi-agent systems; mobile applications; cross-framework interoperability; backend-fed data; content automation and commerce; general	Applied
C. Contribution	protocol or specification; framework; SDK; benchmark or dataset; empirical evaluation; case study; conceptual	Applied
D. Research type	solution proposal; validation; evaluation; experience report; opinion or vision	Applied
E. Evidence class	E0 to E4 (Table 1)	Applied
Generation time	runtime; design-time; both	Applied
Rigor flags	peer-review status; code or data released; threats to validity discussed	Applied
Treatment flags	accessibility; security or governance; state synchronization	Applied
Synchronization mechanism	snapshot replacement; path update; delta or patch; two-way local binding; mixed	Proposed extension
Agent topology	single agent; orchestrator with sub-agents; multi-agent handoff	Proposed extension
Portability	single framework; cross-platform; cross-renderer	Proposed extension

Multi-label coding rules: every technical layer and application dimension that a record substantively addressed was assigned; no single primary label was designated, and per-cell counts in Table 3 should be treated as non-exclusive.

3.6 Synthesis Method

Descriptive counts summarize each coding dimension, and cross-tabulations identify under-represented areas of coverage (e.g., declarative protocols with human evaluation, interoperability with peer-reviewed evidence). Finally, qualitative synthesis compares claims, assumptions, and limitations among applications within each dimension. Counts are not always useful; a layer coded frequently may still be ineffective, and a missing cell may not indicate a lack of work but may instead reflect terminology or indexing. All records were coded by one reviewer; this introduces classification subjectivity and prevents estimation of inter-rater reliability. All coding decisions were recorded in the coding sheet that was released with the data to support auditing and re-coding by others.

4. Results and Discussion

4.1 Publication Landscape (RQ1)

The corpus is very recent and still growing, with 34/59 records from 2026, 17 from 2025, and only one from before 2024. Much of the corpus has grown through preprints and peer-reviewed conferences in human-computer interaction and language technology. Generation-time distribution shows 41 records involving runtime generation only, 15 design-time only, and 3 both (Figure 2). Maturity is better assessed from research type and evidence class: solution-proposal and specification records remain common (see Section 4.3), while independent evaluations of declarative runtime protocols are limited, suggesting an immature evidence base for that specific class of system even as code-generation research matures.

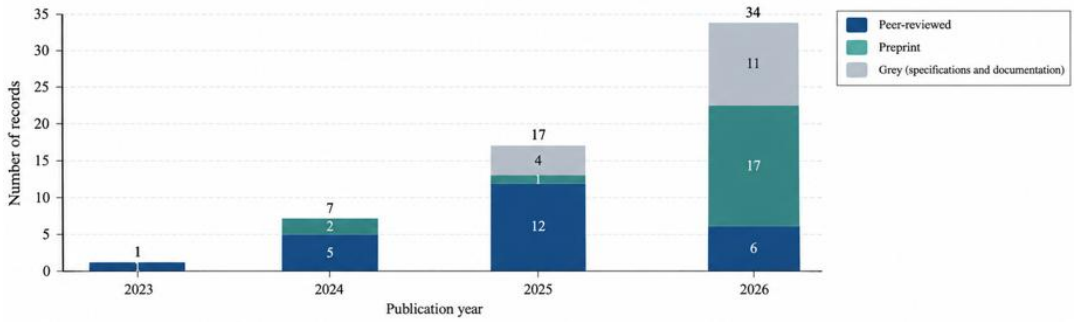


Figure 2. Records in the mapping corpus by publication year and source type (search cutoff 25 September 2026).

The mix of sources used to answer this question is itself a finding: the grey sources are not marginal to this field. Most information about how runtime interfaces are moving between agents and clients comes from protocol specifications, SDK documentation, and engineering reports. If restricted to peer-reviewed work, this mapping would describe interface generation rather than the protocols used by production systems in practice.

4.2 Technical Focus and Contributions (RQ2, RQ3)

Focus is highest on the code-generating UI layer, with 37 records for the code-generating UI and 11 for the declarative UI protocols (of which just one is peer-reviewed). Renderers and SDKs, along with hybrid and governance harnesses, receive an intermediate level of attention, while transport/orchestration and backend data integration receive the least (Table 3).

Table 3. Technical layer by evidence class (multi-label; number of records)

Layer	E0	E1	E2	E3	E4	Total	Peer-reviewed
A1 Declarative UI protocol	4	1	1	4	1	11	1
A2 Transport or orchestration	4	2	0	0	0	6	0
A3 Renderer or SDK	2	6	1	2	6	17	5
A4 Code-generating UI	1	3	2	15	16	37	20
A5 Hybrid or governance	6	4	3	2	7	22	7
A6 Backend data or API	2	0	0	4	0	6	2

Note: because records can be assigned to more than one technical layer, column and row totals in Table 3 are not mutually exclusive and should not be summed against the 59-record corpus total.

Outside of the research/engineering divide, the corpus of peer-reviewed contributions consists mainly of frameworks, benchmarks, and empirical work. Core protocol definitions are concentrated in specifications and engineering documentation, while renderer and SDK framework evaluations are better represented in the empirical literature, appearing across E1 through E4. The research community is measuring how well models can write interface code, while the engineering community is standardizing how interfaces are described without code. They seem to develop largely in parallel, and very few records test both a model and the protocol that would be used to carry its output.

The renderer layer deserves special comment because it is where agent intent meets platform reality. The renderer and SDK records are separated out by illustrative documentation and human evaluations of specific systems (Table 3), but there is little technical evaluation in between. This layer has practical implications: A renderer must decide how to display unresolved bindings and validation failures, change focus when a component appears, and queue actions while a stream is open. These decisions determine whether two conformant clients feel equivalent to a user, but they are rarely specified or assessed. On high-traffic consumer interfaces, the renderer is used for performance budgets, caching, and telemetry, so leaving the renderer out of a technical evaluation leaves a practical blind spot.

4.3 Research Type and Evidence Strength (RQ4, RQ5)

The corpus contains 10 E0, 10 E1, 3 E2, 18 E3, and 18 E4 records, but the higher-validation classes are unevenly distributed across technical layers. Of the 36 records that were at E3 or E4, only 5 can be said to have experimentally tested a declarative or schema-based representation; the others tested generated markup, pages, or components. Most empirical evidence on failure modes concerns generated markup or components, whereas the declarative, catalog-based protocols intended to constrain such failures have received comparatively little independent evaluation.

The most common are validation studies, followed by evaluation studies and solution proposals. Experience reports are relatively rare. There are moderate signals for reproducibility: thirty-one records released code or data, while 14 explicitly discussed threats to validity based on the material inspected. However, the latter number should be treated with caution, as validity flags were coded from abstracts, landing pages, and documentation rather than a full-text audit.

This finding has two implications for RQ5 and how evidence is cited. First, claims about the usefulness of runtime interfaces now have credible E3 and E4 support, at least for generated pages and components in bounded tasks. Second, safety, interoperability, and production readiness arguments for declarative protocols mainly exist at E0 and E1. This gap is sharpest for A2UI specifically: of the declarative/schema-constrained records in the corpus that reach E3 or E4, independent evaluation concentrates on broader catalog-based or schema-driven UI generation rather than on A2UI's own behavior, which remains supported mainly by its specifications. For example, using increases in preference for generated pages as an argument for a catalog-based protocol would require evidence that is currently not found within the corpus.

There are several reasons why declarative representations may not have been evaluated. Catalog-based protocols tightly couple generation to the set of components in a catalog, limiting the transfer of results across catalogs. Their outputs are less visually prominent than pages generated by the techniques, making them harder to evaluate in preference studies. Protocols are also fairly new: the A2UI specification for production was from late 2025, and a lot of the technical work that could be used to test it happened too late for the search. None of these reasons makes the gap less consequential, since production systems are increasingly being implemented with precisely those representations that have been tested least.

4.4 Application Dimensions (RQ6)

The majority of records are on non-specific applications, with little coverage of DevOps and incident response (0 records), multi-agent systems (3), mobile applications (4), commerce (6, of which 4 are peer-reviewed), enterprise (7, of which 1 is peer-reviewed), interoperability (9, all grey), or backend-fed data (9, of which 2 are peer-reviewed). Cross-cutting concerns are also sparse. There are 5 publications on accessibility (2/5 at E3 or E4), 10 on security or governance issues (2/10 at E3 or E4), and 8 on state-synchronization concerns (3/8 at E3 or E4).

4.4.1 Enterprise Approvals and Governance

The approval process will be applicable to agent-generated interfaces because required fields, evidence, and policy findings are not the same for each approval request. The critical question is not only whether the interface can be rendered, but also whether it is authorized, semantically valid, auditable, and correctly separated from action execution. Enterprise records in the corpus mostly include specifications and documentation of declarative card templates bound to API responses and agent frameworks with human-approval steps; one example is the A2UI requirement that orchestrators set or validate attribution fields so that agents cannot impersonate trusted services [5].

While a catalog may restrict content that would normally be present, a syntactically valid surface might still have a business-level misuse case; for example, a user may be presented with an approval control but not be permitted to submit it. Syntactic validity is not semantic validity. Adjacent MCP security evidence shows that tool-layer vulnerabilities can affect upstream data supplied to agent-generated surfaces: a review of 1,899 open-source MCP servers found 7.2% with general vulnerabilities and 5.5% with MCP-

specific tool poisoning [21]. Because output from these tools supplies the data models that approval surfaces use to display information, such weaknesses could propagate into the surface, though this corpus does not directly evaluate that propagation.

4.4.2 DevOps and Incident Response

No included record directly evaluates agent-generated interfaces for incident response. The closest evidence comes from systems that generate or compose analytics dashboards from live data, which are not evaluated in a time-critical operational environment. This step is important because incident tooling involves traces, logs, ownership, runbooks, and remediation actions that differ between incidents. Adapting a dynamic surface is difficult due to the need to consider the model version, prompt context, tool results, permission scopes, and data. Without surface snapshots, event traces, catalog versions, and action provenance, it may not be possible to recreate what the operator had actually seen before acting. These gaps motivate treating generated surfaces as versioned, observable operational artifacts in future systems and evaluations, though this issue remains a research and engineering implication rather than an established result in the corpus.

4.4.3 Multi-Agent Systems

Only three publications describe multi-agent implementations: the A2A specification, documentation on the server-pushed interfaces emitted from agent graph nodes, and a workshop paper on using cooperating agents to create an enterprise dashboard. Composition is architecturally possible because A2UI includes A2A transport, and A2A contains a mechanism for the agent graph to delegate tasks to other agents [7]. The reviewed literature does not establish how a sub-agent's interface fragment should be propagated by an orchestrator, policy-filtered, coalesced with other interface fragments, and rendered. Similarly, the specifications and research literature do not provide easy answers to issues of ownership, provenance, permission scope, and action routing across a trust boundary.

4.4.4 Mobile Applications and Accessibility

Mobile rendering also has some limitations that tend to be hidden in browser demos, such as screen size, network availability, interruption of the lifecycle, native accessibility semantics, and performance budget constraints. The 4 mobile records include the alpha version of a cross-platform SDK [9], a typed state abstraction for multimodal mobile interaction, a study of personalized generated mobile surfaces, and a code-generation approach for a native mobile surface. None of the four mobile records in the mapped corpus evaluates how the quality of the generated surface degrades when the network, model stream, or renderer fails.

Accessibility is another parallel weakness made visible in the mobile context. A journal-scale evaluation of LLM-generated code found that models also tend to support basic accessibility properties, such as color contrast and alt-text, but more complex properties, such as ARIA attributes, are often missing [22]. Runtime surfaces also follow a different set of rules since the focus order, labels, and announcements may change every turn but must have the same meaning in each platform's accessibility tree.

4.4.5 Cross-Framework Interoperability

Interoperability has the largest ambition-evidence gap. Of all interoperability records, all but one are specifications, documentation, or engineering reports. The exception is a preprint on catalog-constrained generation with small models. Framework-agnostic payloads and multiple renderer targets allow portable interfaces, but syntax portability does not imply portable behavior. Layout constraints, default semantics for components, validation rules, focus behavior, error presentation, and accessibility trees may differ between renderers for different implementations of the same interface.

There is also a distinction between structural interoperability, in which renderers accept the same payload, and semantic interoperability. Semantic interoperability means that the surfaces have the same visible semantics (meaning), affordances (available actions), state changes, and safety properties. No included study empirically evaluates semantic conformance (as opposed to structural or syntactic conformance) across multiple renderers. A second axis of divergence for grey literature is between declarative catalog protocols and sandboxed-markup protocols, where the host renders the agent-published HTML in a separate frame and mediates all communications through logged messages (Supplementary Table S1). However, this divergence tends to be a trade-off between expressiveness and reasoning about what a surface can do.

4.4.6 Backend-Fed Data and State Convergence

However, backend integration is often understudied. Technical case studies include dashboard generators that separate analysis from deterministic rendering via an intermediate representation and dashboards evaluated by their performance over real data. The protocol structure suggests a potential advantage for view-oriented contracts between the backend and the surface, but the corpus does not contain controlled comparisons establishing that advantage. Directly exposing a database-shaped object graph to the surface is fragile for bindings; a smaller state surface is more resilient to upstream changes. Separating component structure from application data, for example, in A2UI's notion of updating certain paths, is another step in this direction (though limited).

However, there are no comparisons of stale-state exposure, lost updates, reconnect behavior, the ordering of updates, or time to a semantically complete surface between request-response and streaming backends to quantify their differences in more detail. Because AG-UI already distinguishes the procedure of snapshot recovery from the sequential deltas in [8], this area is also a concrete systems gap, rather than a hypothetical one.

4.4.7 Content Automation and Commerce Interfaces

Commerce is a high-consequence application dimension because interface variation can influence user decisions and transactions. Four of the six commerce records are peer-reviewed, including the redesign of end-user shopping and booking interfaces, on-demand generated controls for content-creation tools, and an engineering paper describing interactive product dimensions delivered through a tool protocol. The strongest warning comes from a survey of 1000 LLM-generated e-commerce components finding that "55.8% of components contained at least one deceptive design, and 30.6% contained two or more" [23].

Interface variation has a demonstrable effect in at least one documented setting: one pilot study of segmented interface variants for an online retailer reported a 46% conversion increase and an 11% higher average order value in its analyzed group; this result is context-specific and should not be generalized beyond that study [24]. Generative models are also reaching the content layer, as multivariant storefronts have generated and tested personalized product descriptions [25]. Considering that creator media, catalog, and purchase interfaces are co-located in some high-traffic commerce surfaces [10], an agent composing interfaces around transactional and commercial workflows in that setting would inherit both a strong lever on user behavior and the demonstrated, measurable prevalence of deceptive-design patterns in prior empirical work. Governed commerce surfaces are a priority for primary research because they constrain what an agent may render around a transaction through catalogs, policy checks, and audit trails.

4.5 Research Gap Matrix

Table 4 summarizes identified gaps for all seven dimensions plus accessibility. Priority is based on the thinness of evidence and the consequence of an evidence failure across settings (Figure 3).

Table 4. Research gap matrix

Dimension	Records (peer-reviewed)	Existing emphasis	Underrepresented evidence	Priority
Enterprise approvals and governance	7 (1)	card templates, approval steps, attribution rules	semantic authorization, auditability, action provenance	High
DevOps and incident response	0 (0)	none in the corpus	reproducible surface state, observability, forensic reconstruction	High
Multi-agent systems	3 (1)	coordination protocols, dashboard pipelines	fragment provenance, merge policy, cross-agent authorization	High
Mobile applications	4 (2)	cross-platform SDKs, personalization	degraded-mode behavior, lifecycle handling, native accessibility	High
Cross-framework interoperability	9 (0)	multiple renderers and transports	conformance, semantic equivalence, version compatibility	Very high
Backend-fed data	9 (2)	intermediate representations, dashboard benchmarks	convergence, ordering, conflict handling, reconnect recovery	Very high
Content automation and commerce	6 (4)	adaptive storefronts, generated components	governed rendering around transactions, deceptive-design safeguards	Very high
Accessibility (cross-cutting)	5 (2)	code-level accessibility audits	dynamic focus, announcements, renderer consistency	High

Note: Priority ratings reflect a reproducible rule: Very High where evidence consists of 0–2 peer-reviewed records and the application has potentially serious consequences (user harm, financial harm, or irreversible action); High where evidence is limited but the consequences are moderate.

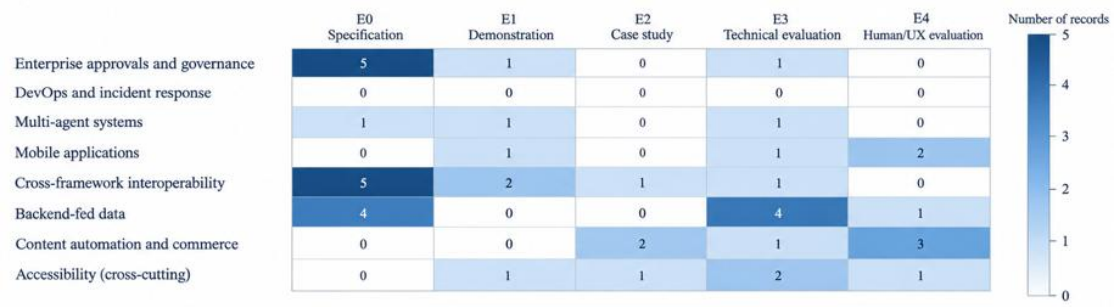


Figure 3. Number of corpus records per application dimension and evidence class, showing where claims rest on specifications rather than evaluation.

4.6 Research Agenda

Semantic authorization means that a generated control is not only schema-valid but also appropriate for the current identity, permission state, workflow state, data context, and permitted action. Achieving this for approval surfaces requires separating allow-listing, identity and permission checks, policy evaluation, action authorization, audit logging, and human confirmation. It remains an open question whether a catalog-based renderer with policy-as-code can prove that an approval surface is semantically authorized for the current state of a workflow, rather than merely valid according to a schema.

Observability and forensics. Surfaces should carry a versioned payload, including a model identifier, task context, catalog version, data bindings, policy decisions, and action events for incident reconstruction. Currently, a general observability model for interface generation across agent protocols is not available.

Multi-agent provenance. The provenance of the component, data value, recommendation, and action should be retained even when contributions are merged by the orchestrator and policies are applied on the renderer side. This problem is particularly acute when agents cross organizational trust boundaries.

Degraded-mode semantics: Protocol designers may need to specify, through explicit fields or metadata, what must be preserved when a renderer is missing a component, a stream is interrupted, data is unavailable, or a user returns to a historical surface. Candidate levels of preservation include content semantics, action-meaning semantics, state semantics, accessibility semantics, and, as a fallback, a best-effort visual representation.

This defines what cross-renderer conformance tests can cover: payload acceptance, semantic components, data binding, events, error states, and accessibility output across renderers. The main challenge is finding tests for semantic equivalence and not just schema validity.

Backend-to-UI state contracts: A useful experiment would compare snapshot delivery against event-stream deltas using the same renderer and task. Furthermore, the following would be measured: time to first useful surface, time to semantically complete state, time exposed to stale values, time to duplicate/reordered updates, time to resolve edit conflicts, time to recover from disconnects, and the recomposition cost. The contracts used for the protocols, such as view-oriented schemas, version negotiation, streamed value provenance, and convergence rules, would be considered.

For transactional and commercial interfaces generally, future work could evaluate whether catalog constraints, deceptive-pattern detection, and policy checks can operate at render time without sacrificing the adaptation that makes generated interfaces valuable in the first place.

5. Threats to Validity

Search validity. Terminology: some authors prefer terms such as adaptive UI, malleable software, schema-driven generation, or interactive agent UI. The use of exact protocol terms, broad concept terms, and snowballing improved coverage but did not eliminate the risk of missing records.

Multivocal and publication bias. Without grey literature, protocol information is missing, while including grey literature may overrepresent vendor claims. Separate source-type and evidence-class labels lower the risk, and no grey source was coded above E2.

Maturity bias. The protocols studied here are relatively new and likely still evolving, with specification versions changing within the review window. Separately, several empirical records in the corpus were preprints published only a few weeks before the search cutoff, meaning their peer-review status may change after this mapping is complete. These conclusions are dated 25 September 2026 and should be revisited as specifications are finalized.

All records were coded by one reviewer, which introduces classification subjectivity and prevents estimation of inter-rater reliability. Some records fit multiple categories, which is addressed through the multi-label coding scheme described in Section 3.5. Multi-label coding and the release of the decision sheet enable re-coding, though inter-rater agreement was not measured.

Depth of rigorous coding. All accessible scholarly records (peer-reviewed papers and preprints) were coded from full text where available; grey-literature records were coded from the most complete stable technical artifact available. Landing-page or

documentation-only coding was used only when full text genuinely could not be accessed, and this is recorded per source in the coding sheet. In Supplementary Table S1, 'N' denotes that the treatment was not identified in the material inspected, not a confirmed absence in the source.

Verification constraints. Some publisher pages were not publicly accessible, and Crossref or author-hosted copies were used to verify the metadata for these records.

No new empirical validation. The mapping characterizes existing evidence but does not establish benefits for the user, security guarantees, production readiness, or performance advantages. This gap is a motivation for experiments, not proof.

Scope of coverage. Attention was restricted to backend APIs and streams that were relevant to data binding, synchronization, latency, or correctness. Persistence technologies were out of scope, as the protocols themselves do not define persistence semantics.

6. Conclusion

The goal of the mapping was to assess what is known about agent-generated user interfaces and how well that knowledge is understood. The answer to this question depends on the layer. There is both technical evidence and human evidence for the existence of runtime interfaces for pages and components in bounded tasks. For A2UI specifically, independent evaluation remains limited to [X] included records beyond its specifications; broader catalog-based and schema-constrained declarative interfaces have somewhat more independent evaluation, concentrated in code generation and benchmark studies rather than protocol-level behavior. This is problematic for researchers and practitioners because test evidence generated for code-generating interfaces cannot be transferred directly to catalog-driven protocols without protocol-specific evaluation. Several high-consequence evidence gaps remain weakly evaluated: approval governance, incident response, multi-agent composition, degraded mobile behavior, cross-renderer equivalence, backend-to-UI state convergence, and commerce surfaces exposed to deceptive design. A promising direction for future work may be to treat each generated surface as an operational artifact with versioned payloads, provenance, and observable state. The operational scope, multivocal mapping, reproducible coding, and the evidence-graded research agenda are all provided with the corpus to permit auditing, extension, or recoding of the categorical classification. The maturity of the protocols, the renderers, state integration, the governance, the evidence, and the uses of the production system may be assessed separately, since improvements achieved in one domain do not guarantee improvements in others. Once the specifications have matured, the same mapping could be reapplied to show whether primary research is closing these gaps.

6.1 Ethics Statement

6.1.1 Statement of Human and Animal Rights

The author performed no studies involving human participants or animals for this article.

6.1.2 Informed Consent

Not applicable, as no human participants were involved.

Acknowledgment

The author received no specific funding for this work.

Data Availability

The corpus, exclusion log and coding decisions are provided as Supplementary Table S1 (in the Appendix).

Funding: This research received no external funding.

Conflicts of Interest: The author declares no conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers.

References

[1] Lei Wang et al., "A survey on large language model based autonomous agents", Frontiers of Computer Science, vol. 18, no. 6, December 2024. Available: https://doi.org/10.1007/s11704-024-40231-1

[2] Yaniv Leviathan et al., "Generative UI: LLMs are Effective UI Generators", arXiv preprint 2604.09577, February 2026. Available: https://arxiv.org/abs/2604.09577

[3] Jiaqi Chen et al., "Generative Interfaces for Language Models", Findings of the Association for Computational Linguistics: ACL 2026, July 2026. Available: https://doi.org/10.18653/v1/2026.findings-acl.74

[4] Yining Cao et al., "Generative and Malleable User Interfaces with Generative and Evolving Task-Driven Data Model", Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems, April 2025. Available: https://doi.org/10.1145/3706598.3713285

[5] A2UI Project, "A2UI (Agent to UI) Protocol v0.9.1", Protocol Specification (Current Production), December 2025. Available: https://a2ui.org/specification/v0.9.1-a2ui/

[6] A2UI Project, "A2UI (Agent to UI) Protocol v1.0", Protocol Specification (Release Candidate), June 2026. Available: https://a2ui.org/specification/v1.0-a2ui/

[7] A2A Project, "Agent2Agent (A2A) Protocol Specification, Version 1.0.0", Linux Foundation, 2026. Available: https://a2a-protocol.org/v1.0.0/specification

[8] AG-UI Protocol, "State Management", Agent User Interaction Protocol Documentation, 2026. Available: https://docs.ag-ui.com/concepts/state

[9] Flutter Team, "GenUI SDK for Flutter", Flutter Documentation, August 2026. Available: https://docs.flutter.dev/ai/genui

[10] Xi Luo et al., "Live Streaming Commerce: A Review and Research Agenda", Journal of Computer Information Systems, vol. 65, no. 3, December 2023. Available: https://doi.org/10.1080/08874417.2023.2290574

[11] Hao Liu et al., "Impact of AI-generated virtual streamer interaction on consumer purchase intention: A focus on social presence and perceived value", Journal of Retailing and Consumer Services, vol. 85, July 2025. Available: https://doi.org/10.1016/j.jretconser.2025.104290

[12] Kai Greshake et al., "Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection", Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security, November 2023. Available: https://doi.org/10.1145/3605764.3623985

[13] Xinyi Hou et al., "Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions", ACM Transactions on Software Engineering and Methodology, vol. 35, no. 10, September 2026. Available: https://doi.org/10.1145/3796519

[14] Paulo Sérgio Almeida, "Approaches to Conflict-free Replicated Data Types", ACM Computing Surveys, vol. 57, no. 2, November 2024. Available: https://doi.org/10.1145/3695249

[15] Saša Brdnik et al., "Intelligent User Interfaces and Their Evaluation: A Systematic Mapping Study", Sensors, vol. 22, no. 15, August 2022. Available: https://doi.org/10.3390/s22155830

[16] Silvia Abrahão et al., "Model-based intelligent user interface adaptation: challenges and future directions", Software and Systems Modeling, vol. 20, no. 5, October 2021. Available: https://doi.org/10.1007/s10270-021-00909-7

[17] Paul Brie et al., "Evaluating a Large Language Model on Searching for GUI Layouts", Proceedings of the ACM on Human-Computer Interaction, vol. 7, no. EICS, June 2023. Available: https://doi.org/10.1145/3593230

[18] Kai Petersen et al., "Guidelines for conducting systematic mapping studies in software engineering: An update", Information and Software Technology, vol. 64, August 2015. Available: https://doi.org/10.1016/j.infsof.2015.03.007

[19] Vahid Garousi et al., "Guidelines for including grey literature and conducting multivocal literature reviews in software engineering", Information and Software Technology, vol. 106, February 2019. Available: https://doi.org/10.1016/j.infsof.2018.09.006

[20] Barbara Kitchenham et al., "How Should Software Engineering Secondary Studies Include Grey Material?", IEEE Transactions on Software Engineering, vol. 49, no. 2, February 2023. Available: https://doi.org/10.1109/TSE.2022.3165938

[21] Mohammed Mehedi Hasan et al., "Model Context Protocol (MCP) at First Glance: Studying the Security and Maintainability of MCP Servers", ACM Transactions on Software Engineering and Methodology, May 2026. Available: https://doi.org/10.1145/3814959

[22] Hyunjae Suh et al., "An Empirical Study on Evaluating Accessible Code Generation Capabilities of LLMs", ACM Transactions on Software Engineering and Methodology, June 2026. Available: https://doi.org/10.1145/3820782

[23] Ziwei Chen et al., "Deception at Scale: Deceptive Designs in 1K LLM-Generated E-Commerce Components", Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems, April 2026. Available: https://doi.org/10.1145/3772318.3791063

[24] Adam Wasilewski and Elżbieta Pawełek-Lubera, "Practical assessment of the e-commerce multivariant user interface", Information and Software Technology, vol. 181, May 2025. Available: https://doi.org/10.1016/j.infsof.2025.107684

[25] Adam Wasilewski, "*Harnessing generative AI for personalized E-commerce product descriptions: A framework and practical insights*", Computer Standards & Interfaces, vol. 94, August 2025. Available: https://doi.org/10.1016/j.csi.2025.104012

[26] Appendix: Supplementary Table

[27] Table S1: Corpus_Coding

ID	Source	Year	Source type	Generation time	Technical layer (Facet A)	Contribution (Facet C)	Research type (Facet D)	Evidence class (Facet E)	Application dimension (Facet B)	Code/data available	Threats to validity discussed	Accessibility treated	Security/governance treated	State-sync mechanism treated	Stable URL
C01	Leviathan et al., Generative UI (arXiv)	2026	Preprint	Run time	A4 A3	framework	validation	E4	general	Y	N	N	N	N	https://arxiv.org/abs/2604.09577
C02	Chen et al., Generative Interfaces for LMs (ACL Findings)	2026	Peer-reviewed	Run time	A4	framework	validation	E4	general	Y	Y	N	N	N	https://doi.org/10.18653/v1/2026.findings-acl.74
C03	Wu et al., UICoder (NAACL)	2024	Peer-reviewed	Design-time	A4	framework	validation	E3	mobile	N	N	N	N	N	https://doi.org/10.18653/v1/2024.naacl-long.417
C04	Liu et al., AlignUI (arXiv)	2026	Preprint	Run time	A4 A5	framework	validation	E4	general	N	N	N	N	N	https://arxiv.org/abs/2601.17614
C05	Hojo et al., GenerativeGUI (CHI EA)	2025	Peer-reviewed	Run time	A4 A3	framework	validation	E4	general	N	N	N	N	N	https://doi.org/10.1145/3706599.3719743
C06	Si et al., Design2Code (NAACL)	2025	Peer-reviewed	Design-time	A4	benchmark	evaluation	E3	general	Y	N	N	N	N	https://doi.org/10.18653/v1/2025.naacl-long.199
C07	Cao et al., Generative and Malleable UIs (CHI)	2025	Peer-reviewed	Run time	A1 A4	framework	validation	E4	general	N	N	N	N	Y	https://doi.org/10.1145/3706598.3713285
C08	Dong et al., SchemaGUI (arXiv)	2026	Preprint	Run time	A1 A4	benchmark	evaluation	E3	general	Y	Y	N	N	N	https://arxiv.org/abs/2608.22390
C09	Vaithilingam et al., DynaVis (CHI)	2024	Peer-reviewed	Run time	A4 A3	framework	validation	E4	general	N	Y	N	N	N	https://doi.org/10.1145/3613904.3642639

ID	Source	Year	Source type	Generation time	Technical layer (Facet A)	Contribution (Facet C)	Research type (Facet D)	Evidence class (Facet E)	Application dimension (Facet B)	Code/data available	Threats to validity discussed	Accessibility treated	Security/governance treated	State-sync mechanism treated	Stable URL
C10	Li et al., Sketch2Code (NAACL)	2025	Peer-reviewed	Design-time	A4	benchmark	evaluation	E4	general	Y	N	N	N	N	https://doi.org/10.18653/v1/2025.naacl-long.198
C11	Yun et al., Web2Code (NeurIPS)	2024	Peer-reviewed	Design-time	A4	benchmark	evaluation	E3	general	Y	N	N	N	N	https://doi.org/10.52202/079017-3560
C12	Cheng et al., BISCUIT (VL/HCC)	2024	Peer-reviewed	Run time	A4 A3	framework	validation	E4	general	N	N	N	N	N	https://arxiv.org/abs/2404.07387
C13	Xiao et al., Interaction2Code (ASE)	2025	Peer-reviewed	Design-time	A4	benchmark	evaluation	E3	general	Y	N	N	N	N	https://doi.org/10.1109/ASE63991.2025.00028
C14	Laurencon et al., WebSight (arXiv)	2024	Preprint	Design-time	A4	benchmark	solution proposal	E3	general	Y	N	N	N	N	https://arxiv.org/abs/2403.09029
C15	Lu et al., WebGen-Bench (NeurIPS)	2025	Peer-reviewed	Design-time	A4 A6	benchmark	evaluation	E3	backend	Y	N	N	N	N	https://proceedings.neurips.cc/paper_files/paper/2025/hash/6841eed8bb6a2ec49e49235c8115efee-Abstract-Datasets_and_Benchmarks_Track.html
C16	Zhang et al., ArtifactsBench (arXiv)	2025	Preprint	Run time	A4	benchmark	evaluation	E3	general	Y	N	N	N	N	https://arxiv.org/abs/2507.04952
C17	Kolthoff et al., Zero-shot GUI generation (arXiv)	2024	Preprint	Design-time	A4	empirical	validation	E4	general	Y	N	N	N	N	https://arxiv.org/abs/2412.11328
C18	Sawicki et al., Qualitative evaluation of LLM-designed GUI (arXiv)	2026	Preprint	Design-time	A4	empirical	evaluation	E4	general	N	N	Y	N	N	https://arxiv.org/abs/2601.22759

ID	Source	Year	Source type	Generation time	Technical layer (Facet A)	Contribution (Facet C)	Research type (Facet D)	Evidence class (Facet E)	Application dimension (Facet B)	Code/data available	Threats to validity discussed	Accessibility treated	Security/governance treated	State-sync mechanism treated	Stable URL
C19	Lee, Working definition of designing GenUI (DIS Companion)	2025	Peer-reviewed	Both	A5	conceptual	opinion/vision	E0	general	N	N	N	N	N	https://doi.org/10.1145/3715668.3736365
C20	Min et al., Malleable overview-detail interfaces (CHI)	2025	Peer-reviewed	Run time	A3 A5	framework	validation	E4	commerce	N	N	N	N	N	https://doi.org/10.1145/3706598.3714164
C21	Min et al., Gradual generation of UIs (arXiv)	2026	Preprint	Both	A4 A5	conceptual	opinion/vision	E1	general	N	N	N	N	N	https://arxiv.org/abs/2601.17975
C22	A2UI protocol specification family v0.9/v0.9.1/v1.0 candidate	2026	Grey (spec/docs/blog)	Run time	A1 A2 A5	protocol	solution proposal	E0	interop enterprise backend	Y	N	N	Y	Y	https://a2ui.org/specification/v0.9.1-a2ui/
C23	A2UI public announcement (developer blog)	2025	Grey (spec/docs/blog)	Run time	A1 A5	protocol	opinion/vision	E0	enterprise	Y	N	N	Y	N	https://developers.googleblog.com/introducing-a2ui-an-open-project-for-agent-driven-interfaces/
C24	A2A protocol specification v1.0.0	2026	Grey (spec/docs/blog)	Run time	A2	protocol	solution proposal	E0	multiagent interop enterprise	Y	N	N	N	N	https://a2a-protocol.org/latest/specification/
C25	AG-UI protocol documentation	2026	Grey (spec/docs/blog)	Run time	A2 A6	protocol	solution proposal	E0	interop backend	Y	N	N	N	Y	https://docs.ag-ui.com/introduction

ID	Source	Year	Source type	Generation time	Technical layer (Facet A)	Contribution (Facet C)	Research type (Facet D)	Evidence class (Facet E)	Application dimension (Facet B)	Code/data available	Threats to validity discussed	Accessibility treated	Security/governance treated	State-sync mechanism treated	Stable URL
C26	MCP Apps extension specification and announcement	2026	Grey (spec/docs/blog)	Run time	A3 A4 A5	protocol	solution proposal	E0	enterprise interop	Y	N	N	Y	N	https://modelcontextprotocol.io/extensions/apps/overview
C27	GenUI SDK for Flutter (package and docs)	2026	Grey (spec/docs/blog)	Run time	A3	sdk	solution proposal	E1	mobile interop	Y	N	N	N	Y	https://docs.flutter.dev/ai/genui
C28	Generative UI module of a web AI SDK (docs)	2026	Grey (spec/docs/blog)	Run time	A3	sdk	solution proposal	E1	general	Y	N	N	N	N	https://ai-sdk.dev/docs/ai-sdk-ui/generative-user-interfaces
C29	Agent UI framework with human approvals (docs)	2026	Grey (spec/docs/blog)	Run time	A3 A5	sdk	solution proposal	E1	enterprise	Y	N	N	Y	N	https://docs.copilotkit.ai/
C30	Kong et al., Macaron-A2UI (arXiv)	2026	Preprint	Run time	A1 A3	benchmark	validation	E3	general	Y	N	N	N	N	https://arxiv.org/abs/2605.24830
C31	Brie et al., LLM search for GUI layouts (PACM HCI)	2023	Peer-reviewed	Design-time	A4	empirical	validation	E4	general	N	N	N	N	N	https://doi.org/10.1145/3593230
C32	Gurita and Vatavu, LLM code accessibility barriers (W4A)	2025	Peer-reviewed	Design-time	A4	empirical	evaluation	E3	general	N	N	Y	N	N	https://doi.org/10.1145/3744257.3744266

ID	Source	Year	Source type	Generation time	Technical layer (Facet A)	Contribution (Facet C)	Research type (Facet D)	Evidence class (Facet E)	Application dimension (Facet B)	Code/data available	Threats to validity discussed	Accessibility treated	Security/governance treated	State-sync mechanism treated	Stable URL
C33	Suh et al., Accessible code generation (TOSEM)	2026	Peer-reviewed	Design-time	A4	empirical	evaluation	E3	general	N	N	Y	N	N	https://doi.org/10.1145/3820782
C34	Kolthoff et al., GUIDE (ICSE Companion)	2025	Peer-reviewed	Design-time	A4	framework	solution proposal	E1	general	Y	N	N	N	N	https://doi.org/10.1109/ICSE-Companion66252.2025.00010
C35	Yang et al., React Genie (CHI)	2024	Peer-reviewed	Run time	A3 A5	framework	validation	E4	mobile	N	N	N	N	Y	https://doi.org/10.1145/3613904.3642517
C36	Generative UI deployment report (research blog)	2025	Grey (spec/docs/blog)	Run time	A4 A5	case study	experience report	E2	general	N	N	N	N	N	https://research.google/blog/generative-ui-a-rich-custom-visual-interactive-user-experience-for-any-prompt/
C37	MCP-UI project documentation	2025	Grey (spec/docs/blog)	Run time	A1 A2 A3	protocol	solution proposal	E0	interop	Y	N	N	N	N	https://mcpui.dev/about
C38	Interactive MCP UI components in commerce (engineering blog)	2025	Grey (spec/docs/blog)	Run time	A1 A3 A5	case study	experience report	E2	commerce interop	Y	N	N	Y	N	https://shopify.engineering/mcp-ui-breaking-the-text-wall
C39	Chat host app SDK custom UX (docs)	2026	Grey (spec/docs/blog)	Run time	A2 A3	sdk	solution proposal	E1	interop	Y	N	N	Y	N	https://developers.openai.com/apps-sdk/build/custom-ux/
C40	Agent graph generative UI (docs)	2026	Grey (spec/docs/blog)	Run time	A2 A3	sdk	solution proposal	E1	multiagent	Y	N	N	N	Y	https://docs.langchain.com/langsmith/generative-ui-react

ID	Source	Year	Source type	Generation time	Technical layer (Facet A)	Contribution (Facet C)	Research type (Facet D)	Evidence class (Facet E)	Application dimension (Facet B)	Code/data available	Threats to validity discussed	Accessibility treated	Security/governance treated	State-sync mechanism treated	Stable URL
C41	Adaptive card response templates for enterprise copilot plugins (docs)	2026	Grey (spec/docs/blog)	Run time	A1 A6	protocol	solution proposal	E0	enterprise backend	N	N	N	Y	Y	https://learn.microsoft.com/en-us/microsoft-365/copilot/extensibility/api-plugin-adaptive-cards
C42	Open UI generative UI toolkit (docs)	2026	Grey (spec/docs/blog)	Run time	A1 A3	sdk	solution proposal	E1	general	N	N	N	N	N	https://www.openui.com/docs
C43	Peng et al., Personalization of generative UIs (arXiv)	2026	Preprint	Run time	A4 A5	empirical	validation	E4	mobile	N	Y	N	N	N	https://arxiv.org/abs/2604.09876
C44	Shi et al., NL2Dashboard (arXiv)	2026	Preprint	Run time	A1 A6	framework	validation	E3	backend	N	N	N	N	Y	https://arxiv.org/abs/2601.06126
C45	Wang and Deng, DashArena (arXiv)	2026	Preprint	Run time	A4 A6	benchmark	evaluation	E3	backend	Y	Y	N	N	N	https://arxiv.org/abs/2608.10567
C46	Zhang and Elhamod, Data-to-Dashboard (KDD workshop)	2025	Peer-reviewed	Run time	A4 A6	framework	validation	E3	multiagent enterprise backend	Y	N	N	N	N	https://arxiv.org/abs/2505.23695

ID	Source	Year	Source type	Generation time	Technical layer (Facet A)	Contribution (Facet C)	Research type (Facet D)	Evidence class (Facet E)	Application dimension (Facet B)	Code/data available	Threats to validity discussed	Accessibility treated	Security/governance treated	State-sync mechanism treated	Stable URL
C47	Ryskeldiev, Generative UI as accessibility bridge in C2C commerce (arXiv/workshop)	2026	Preprint	Runtime	A4 A5	conceptual	experience report	E2	commerce	N	Y	Y	N	N	https://arxiv.org/abs/2604.25455
C48	Lindley et al., What does GenUI mean for HCI practice (CHI EA)	2026	Peer-reviewed	Both	A5	conceptual	opinion/vision	E0	general	N	N	N	N	N	https://doi.org/10.1145/3772363.3778757
C49	Xie and Xie, Software as content (arXiv)	2026	Preprint	Runtime	A4 A5	conceptual	opinion/vision	E1	general	N	Y	N	N	N	https://arxiv.org/abs/2603.21334
C50	Peng et al., EvoGenUI-Bench (arXiv)	2026	Preprint	Runtime	A4	benchmark	evaluation	E3	general	Y	Y	N	N	N	https://arxiv.org/abs/2608.29387
C51	Imteyaz et al., Design Theater (arXiv)	2026	Preprint	Runtime	A4 A5	benchmark	evaluation	E3	general	Y	Y	N	N	N	https://arxiv.org/abs/2607.22928
C52	Krauss et al., Unsolicited deceptive designs in generated websites (CHI)	2025	Peer-reviewed	Design-time	A4 A5	empirical	evaluation	E4	commerce	N	Y	N	Y	N	https://doi.org/10.1145/3706598.3713083

ID	Source	Year	Source type	Generation time	Technical layer (Facet A)	Contribution (Facet C)	Research type (Facet D)	Evidence class (Facet E)	Application dimension (Facet B)	Code/data available	Threats to validity discussed	Accessibility treated	Security/governance treated	State-sync mechanism treated	Stable URL
C53	Chen et al., Deception at scale in LLM e-commerce components (CHI)	2026	Peer-reviewed	Design-time	A4 A5	empirical	evaluation	E3	commerce	Y	Y	N	Y	N	https://doi.org/10.1145/3772318.3791063
C54	Jerry et al., LLM-driven accessible interface, model-based (arXiv)	2026	Preprint	Run time	A5	framework	solution proposal	E1	general	N	Y	Y	Y	N	https://arxiv.org/abs/2601.06616
C55	Yang et al., Declarative UI generation at small-model cost (arXiv)	2026	Preprint	Run time	A1 A3	empirical	validation	E3	interop	N	N	N	N	N	https://arxiv.org/abs/2609.04184
C56	Kim et al., Maru (UIST)	2026	Peer-reviewed	Run time	A4 A5	framework	validation	E4	general	Y	N	N	N	N	https://arxiv.org/abs/2608.25565
C57	Li et al., Spatula (UIST)	2026	Peer-reviewed	Run time	A4	framework	validation	E4	commerce	N	Y	N	N	N	https://arxiv.org/abs/2607.10405
C58	Chen and Pavel, TaskArtisan (arXiv)	2026	Preprint	Run time	A4 A5	empirical	validation	E4	backend	N	N	N	N	N	https://arxiv.org/abs/2607.17394
C59	Reddy, The keyhole effect (arXiv)	2026	Preprint	Run time	A5	conceptual	opinion/vision	E0	backend	N	Y	N	N	N	https://arxiv.org/abs/2602.00947