International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 13 Issue: 02 | Feb 2026 www.irjet.net p-ISSN: 2395-0072

Bidirectional Human AI Alignment for Generative User Interfaces and Design Developer Tooling

Nadim Akhtar[1], Laraib Ahmad Siddiqui[2]

[1]Assistant Professor, Computer Science Engineering, Chandigarh University
[2]MLOps Engineer, Accenture

Abstract - *Generative user interfaces promise substantial gains in speed and breadth of exploration by allowing teams to produce interface concepts, component trees, and production code from natural-language intent. Yet current GenUI workflows are often brittle: they overfit to underspecified prompts, fail to respect product constraints such as accessibility and design-system rules, and provide limited visibility into why specific layout or interaction choices were made. These gaps are not merely model errors; they reflect misalignment between what people mean, what the system infers, and what downstream teams can validate and maintain. This paper frames the problem as bidirectional human AI alignment, in which the tool not only adapts to users but also helps users externalize intent as explicit, checkable commitments. We propose a mixed-initiative GenUI pipeline that converts multi-modal intent (text, screenshots, sketches, and existing UI code) into constraint artifacts, generates diverse UI variants linked to editable code, and performs verification against constraints including component policies, interaction invariants, and accessibility targets. Rather than relying on opaque prompting, the system surfaces traceable rationales, uncertainty markers, and targeted self-critiques so that designers and developers can negotiate trade-offs by adjusting constraints, not rewriting prompts. We outline an evaluation plan spanning controlled comparisons with prompt-only baselines, expert reviews with UX practitioners and front-end engineers, and longer deployments in real tool chains to study trust calibration, collaboration impacts, and maintainability. The goal is a GenUI workflow that is faster, more accountable, and easier to integrate into production design-to-code processes.*

Key Words: Generative user interfaces, bidirectional alignment, mixed-initiative systems, design-to-code, constraint-based design, verification, explainable AI, developer tooling

1.INTRODUCTION

Generative systems have moved from novelty to daily infrastructure in many software teams, supporting ideation, prototyping, copywriting, documentation, and code generation. User interface work is a natural next frontier because UI artifacts are structurally rich and tightly constrained: interfaces must express product intent while obeying platform conventions, design-system standards, accessibility requirements, and a growing set of governance policies. In principle, Generative User Interfaces (GenUI) could compress the time between an early concept and a working prototype, enabling teams to explore more alternatives and to test ideas earlier with users.

In practice, prompt-based UI generation can raise a familiar risk: speed rises while accountability falls. Misalignment often shows up in small choices that accumulate into real cost-a layout that ignores spacing tokens, a component choice that violates interaction policy, or a flow that looks plausible but fails keyboard navigation. These problems are not always visible at first glance. Many are discovered only when engineers integrate generated code into an existing codebase, when QA runs accessibility audits, or when design reviews compare outputs against a system of reusable components. The practical failure mode is therefore not that generation is impossible; it is that generation is easy to produce but hard to verify, explain, and maintain. The central question becomes not only “Can a model generate a UI?” but “Can the output be interrogated, corrected, and handed off responsibly?” Human-centered AI work emphasizes that trustworthy systems require legibility and user agency, not merely high average performance [1].

Alignment is also socio-technical. UI work is distributed across roles, and each role interprets correctness differently. A designer may prioritize coherence, hierarchy, and brand consistency; an engineer may prioritize component reuse, testability, and future refactoring; a product manager may emphasize compliance and measurable outcomes; and an accessibility specialist may focus on semantics, focus order, and contrast. Prompt-only GenUI is typically optimized around one person’s immediate request, expressed in ambiguous language and without the surrounding organizational context that makes an interface “right.” Even when the result is acceptable, limited traceability makes it difficult to know which parts are safe to accept and which deserve scrutiny, producing “hidden rework” when issues surface downstream.

Several research strands suggest a more robust direction. Mixed-initiative interface principles argue that effective systems share control: automation proposes actions while people guide, override, and refine behavior [3]. Guidelines for human–AI interaction emphasize showing system status and confidence, enabling efficient correction, and maintaining user control through iterative refinement [2]. Interactive machine learning likewise highlights that reliable

outcomes often emerge through feedback loops that align a system to a user's evolving notion of success [6].

This paper builds on these ideas by framing GenUI as bidirectional human–AI alignment. The tool should not only adapt to users, but also help users externalize intent as explicit, editable constraints; verify generated artifacts against those constraints (including design-system and accessibility targets); and provide traceable rationales and uncertainty markers that support trade-off negotiation. The workflow shifts from "generate and hope" to "generate, check, and negotiate," strengthening collaboration and reducing late-stage surprises.

1.1 SOURCES OF MISALIGNMENT IN PROMPT DRIVEN GENUI

To design bidirectional alignment mechanisms, it is necessary to understand where misalignment originates in current GenUI workflows. A first source is under specification. Natural-language prompts are typically compact, while UI requirements are detailed and conditional. A prompt such as "make a checkout screen with a clean modern style" does not specify required fields, error handling, keyboard behavior, localization constraints, analytics tags, or compliance requirements. In established teams, these requirements are encoded in design systems and engineering conventions, not in prompts. When a model generates an interface, it must implicitly fill in missing details. Even if the result looks coherent, those implicit choices may not match what the organization expects. The problem is not that the model cannot invent details; it is that invented details are often indistinguishable from requirements.

A second source is ambiguity across stakeholders. UI correctness is multi-dimensional, and different dimensions are prioritized by different roles. Designers may describe intent in terms of hierarchy and tone; engineers may describe it in terms of state machines and component APIs. When a tool is oriented around a single prompt channel, it tends to privilege whichever dimension is easiest to verbalize. This is why GenUI outputs can be visually persuasive but structurally fragile: they satisfy surface-level aesthetic cues while violating architectural constraints that matter for maintenance. In mixed-role teams, this becomes a coordination problem: people spend time reconciling whether the generated artifact is a prototype, a production candidate, or merely a sketch. Without explicit signals, each person interprets the artifact differently, which slows collaboration.

A third source involves hidden constraints in code and design systems. Modern UI development is rarely Greenfield. Teams reuse component libraries, theming tokens, and interaction patterns that embody decisions about accessibility, brand, responsiveness, and performance. Prompt-based generation often produces "one-off" components that look correct but do not correspond to the reusable primitives that teams depend on. This mismatch creates a fork: either engineers rewrite the UI to fit the system, or the system is bypassed, degrading consistency over time. The long-term cost of bypass is substantial because design systems are meant to reduce variance and accelerate delivery. A GenUI tool that does not respect these constraints may create more work than it saves, particularly for teams with mature systems.

A fourth source is the difficulty of verifying UI behavior from static outputs. Many generative outputs are presented as images or static markup, but real interfaces depend on interaction logic: state transitions, form validation, asynchronous loading, and error recovery. In practice, teams discover issues through testing, review, and instrumentation. If GenUI produces code that is not testable, or if it changes behavior in subtle ways, misalignment can persist until late in the pipeline. This is where interpretability matters: if a system can explain what it assumed about interaction states, the team can more quickly judge whether those assumptions match reality. Explanations are not a luxury; they are a mechanism for earlier detection of mismatch [4], [5].

A fifth source concerns the calibration of trust. When a system generates convincing outputs quickly, users may over-trust the artifact, adopting it without sufficient review. Conversely, after encountering several unexpected failures, users may under-trust the tool and treat it as unreliable, even in cases where it could be helpful. Human-AI interaction guidelines emphasize that systems should help users understand when automation is likely to succeed and how to recover when it fails [2]. In GenUI, this means surfacing uncertainty in a way that is actionable. A vague disclaimer that "results may vary" does not support productive work. What helps is specificity: which parts of the interface are confident, which violate constraints, and which are ambiguous because the intent was not captured.

A sixth source is that iteration is often expressed as "prompt churn." Users repeatedly rewrite prompts to push the system toward the desired output. This is time-consuming and unreliable because prompt edits are an indirect control surface. Prompt churn also makes collaboration difficult, because the reasoning behind a sequence of prompt edits is rarely documented in a way that teammates can interpret. In design and engineering teams, progress depends on shared artifacts: annotated screens, decision logs, and versioned constraints. GenUI interaction that lives only in a prompt history is poorly aligned with these norms. This is why bidirectional alignment should include the production of artifacts that can be discussed, reviewed, and audited, not merely outputs that can be copied.

A final source is governance and responsibility. UI decisions can have ethical and legal consequences, especially when they influence user consent, privacy choices, or access to services. A GenUI tool that generates persuasive interfaces without making trade-offs explicit can unintentionally steer outcomes. Human-centered AI research emphasizes responsible design, including transparency about system behavior and the societal implications of automation [1]. In GenUI, responsibility requires more than telling users to "review carefully." It requires building the review into the workflow, with checks for accessibility guidelines such as WCAG, warnings for dark-pattern-like structures, and documentation that supports accountability [10]. In short, misalignment arises because today's GenUI workflows treat generation as the endpoint, while real teams treat generation as the beginning of negotiation.

1.2 BIDIRECTIONAL ALIGNMENTS AS A TOOLING PRINCIPLE

Bidirectional human AI alignment reframes the interface between people and generative systems as a shared workspace rather than a command channel. The starting point is simple: if design intent cannot be fully captured in a prompt, then a useful tool must help externalize intent in other forms. This includes extracting structure from sketches and screenshots, importing constraints from design systems, and interpreting existing UI code as a partial specification. Bidirectionality means the system not only learns from users but also teaches users what it needs to behave predictably. In practical terms, the system should ask targeted questions, propose constraint candidates, and highlight ambiguity. Such interactions align with mixed-initiative principles, which treat uncertainty and negotiation as normal, not as error cases [3].

A key mechanism is representing intent as constraints. Constraints are valuable because they are checkable, composable, and discussable. They can encode layout policies (for example, grid alignment and spacing tokens), component rules (only use approved primitives), interaction invariants (a submit button is disabled until validation passes), and accessibility targets (minimum contrast, keyboard focus order). Constraints can also capture higher-level goals such as "prioritize clarity over density" by translating them into measurable proxies, though this translation must be visible and editable. The constraint representation becomes a stable artifact: unlike a prompt, it can be versioned, reviewed, and reused across screens. This matches how teams already work with design-system tokens and lint rules, suggesting that alignment can be embedded into existing practices rather than imposed as a new habit.

Directionality also changes how the system communicates. Instead of presenting a single "best" output, the tool can present a small set of diverse variants, each annotated with the constraints it satisfies and the trade-offs it makes. Diversity is not only for creativity; it is a diagnostic tool. When variants differ, users can infer which parts of intent are stable and which are still ambiguous. For instance, if all variants agree on component selection but differ on information hierarchy that suggests hierarchy constraints were underspecified. The system can then propose follow-up questions specifically about hierarchy rather than asking generic clarification prompts. This targeted questioning can reduce cognitive load and shorten iteration, aligning with human-AI interaction guidance to make correction efficient and focused [2].

Another principle is traceability. Generated UI artifacts should be accompanied by rationales that are anchored in constraints and in the source context the system used. Traceability is not the same as verbose explanation. The goal is to support accountability: if a screen uses a particular layout, the user should be able to see that the choice was driven by a "mobile-first single column" constraint, not by an opaque stylistic guess. Where the system is uncertain, traceability should reflect that uncertainty explicitly. Explanations that admit uncertainty can improve trust calibration because they help users decide where to spend review effort [4]. Importantly, traceability should extend to the design-to-code handoff: developers should receive a record of which constraints were assumed so they can decide whether those assumptions still hold in production.

Bidirectional alignment also requires supporting repair. In many generative tools, "fixing" means re-running generation with a modified prompt. A constraint-centered approach supports more direct repair strategies. If spacing is wrong, the user should adjust spacing constraints or select a token; if component choice violates the design system, the user should switch the component policy; if accessibility fails, the tool should propose specific repairs (e.g., increase contrast, add labels, correct focus order). This is closely related to the idea of making the system's actions reversible and supporting incremental changes, which are long-standing usability principles and remain critical when automation is involved [2], [7]. Repair becomes less of a conversation with a black box and more of an engineering-like iteration on explicit requirements.

A further dimension is collaboration. Design and development are not solitary activities, and alignment mechanisms must support shared understanding. Constraint artifacts can be used as boundary objects: designers can specify visual and interaction constraints, engineers can add architectural and performance constraints and both can see conflicts when constraints collide. For instance, a designer might want a dense information layout, while an engineer might enforce a minimum tap target size. A bidirectional tool can surface such conflicts and offer options rather than silently picking one. Over time, the constraint library can become part of organizational knowledge, capturing

patterns that are otherwise spread across docs, code comments, and informal mentorship.

Finally, bidirectional alignment must respect responsible generation. In UI work, responsible practice includes accessibility, privacy, and avoidance of manipulative patterns. A GenUI tool should therefore treat audits as first-class steps, not afterthoughts. Verification against accessibility guidelines and internal policies can be integrated into the generation loop, and warnings can be prioritized based on user impact. This approach is consistent with human-centered AI views that emphasize transparency and societal concerns as integral to design, not as external compliance steps [1]. In summary, bidirectional alignment is best understood as a tooling principle: it is a commitment to making intent explicit, making outputs checkable, and making negotiation and repair central to the user experience of GenUI.

2. PROPOSED SYSTEM ARCHITECTURE AND METHODOLOGY

This section describes a practical system concept for bidirectional human AI alignment in GenUI, organized around a pipeline that turns intent into constraints, constraints into candidates, and candidates into verified, explainable artifacts suitable for handoff. The system begins with multi-modal intent capture. Users can provide natural-language goals, reference screenshots, sketch annotations, and optionally an existing codebase context such as a component library. The tool parses inputs into an initial intent model: a structured representation of screen purpose, primary user tasks, information hierarchy, and target platforms. Because many UI requirements are implicit, the tool also proposes inferred constraints derived from context. For example, if a team imports a design system, the tool can infer that spacing tokens must be used, typography must follow a scale, and components must be drawn from an approved set. These inferences are marked as suggestions, not requirements, and the user can accept, edit, or reject them.

The second stage is constraint compilation. The intent model is translated into a constraint set that is both human-readable and machine-checkable. Constraints can be grouped into categories such as layout, components, interaction, content, accessibility, and engineering. Each constraint has a provenance label that records whether it came from the user explicitly, from imported systems, or from model inference. Provenance matters because it shapes how conflicts are resolved. If an inferred constraint conflicts with an explicit user constraint, the system should defer to the explicit constraint while still flagging the risk of policy violation. Conversely, if a constraint comes from an imported design system, the system can treat it as higher priority by default. This ranking is editable, since teams differ in how strictly they enforce design-system compliance.

The third stage is generation of UI variants. Rather than producing a single design, the system generates a small set of candidates that intentionally span different trade-offs while satisfying the high-priority constraints. Candidates include a rendered preview, a component tree, and code in a target framework (for example, React, Flutter, or platform-native code) with clear mapping to reusable primitives. Diversity is guided, not random: the system varies aspects such as hierarchy, density, and navigation patterns within allowable bounds. Each candidate is accompanied by a compact "constraint report" showing which constraints were satisfied, which were violated, and which were only partially satisfied. This supports a workflow where selection is informed by compliance and by design intent, not just visual preference.

The fourth stage is verification. Verification includes deterministic checks (lint rules, token usage, component constraints, and accessibility checks) and probabilistic checks (uncertainty about intent interpretation). Deterministic checks align with existing engineering practice: teams already use linters and tests, and GenUI should leverage that ecosystem rather than bypass it. Accessibility checks can include labels, roles, contrast, focus order, and tap target size aligned to relevant standards [10]. Where checks cannot be fully automated, the tool should still surface risks and recommend human review. For example, content clarity and appropriateness are difficult to verify automatically; the tool can flag these as "requires review" and suggest evaluation heuristics. Verification outputs are integrated directly into the design view, so users see violations in context, not in a separate report that is easy to ignore.

The fifth stage is explanation and repair. Explanations are constraint-linked: users can click a UI element and see which constraints influenced its existence and properties. The system also provides self-critiques: targeted notes about where the candidate is likely weak, framed as repair suggestions rather than generic apologies. For example, if a form is missing inline error feedback, the tool can propose an error-state pattern consistent with the design system. Repair actions can be applied locally (fix this component) or globally (tighten spacing constraints). The key is that repair should be incremental and reversible, helping users converge without restarting. This reflects established principles that interactive systems should support efficient correction and enable users to recover from errors [2], [7].

Table -1: Alignment artefacts and checks (example structure to include in Word as a table

Alignment artifact	What it represents	How it is checked	Typical user action
Intent model	Screen purpose, tasks, hierarchy	Consistency checks across inputs	Approve or refine intent summary

Alignment artifact	What it represents	How it is checked	Typical user action
Constraint set	Explicit requirements and policies	Rule engine and linting	Add, edit, prioritize constraints
Variant set	Diverse UI candidates + code	Constraint report per variant	Compare, select, merge attributes
Verification report	Violations and risk flags	Accessibility, design system, tests	Apply recommended fixes or waive
Rationale trace	Why decisions were made	Link decisions to constraints	Audit decisions, justify handoff

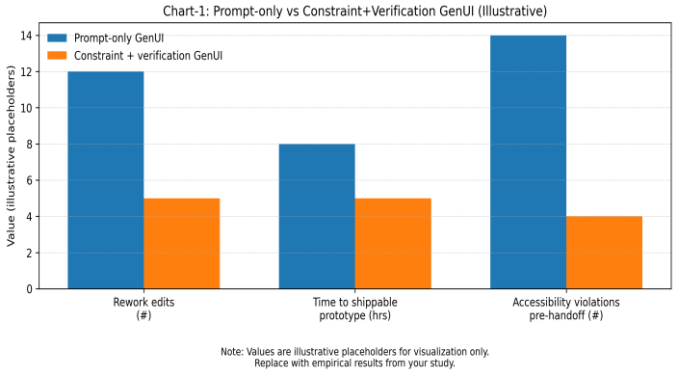

Chart -1: Example evaluation outcome visualization (caption)

Comparison of prompt-only GenUI versus constraint and verification based GenUI on (a) number of rework edits before acceptance, (b) time to reach a shippable prototype, and (c) accessibility violations detected pre-handoff.

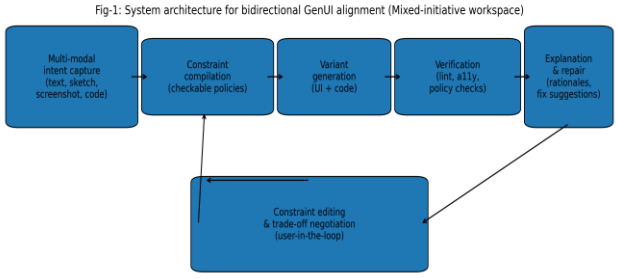

Fig -1: System architecture (caption)

Pipeline showing multi-modal intent capture, constraint compilation, variant generation, verification, and explanation and repair loops integrated into a single mixed-initiative workspace.

Evaluation should measure outcomes that matter to teams, not only aesthetic quality. Controlled studies can compare the proposed workflow against prompt-only baselines on metrics such as time-on-task, number of iterations, perceived agency, and the rate of post-generation fixes required for design-system and accessibility compliance. Surveys and standardized instruments such as SUS can capture perceived usability [8], while qualitative interviews reveal whether users feel more in control of the generative process. Practitioner evaluations with designers and front-end engineers can assess collaboration impacts, including whether constraint artifacts improve communication at the design-to-code boundary. Longitudinal deployments can then examine whether teams continue using the tool once novelty fades, whether constraint libraries become shared resources, and whether maintainability improves over time as measured by component reuse and reduced divergence from design systems.

3. CONCLUSIONS

GenUI will be most valuable when it supports not only rapid creation but also dependable integration into the social and technical realities of product development. Prompt-only generation has shown that plausible interfaces can be produced quickly; the remaining challenge is to make those outputs legible and sustainable. Teams must be able to see what was assumed, how outputs relate to organizational constraints, and how to correct misalignment without repeatedly restarting. This paper has argued that these needs are best addressed by treating GenUI as a bidirectional alignment problem, where both system and user adapt through explicit negotiation. The objective is not to remove iteration, but to make iteration structured, checkable, and accountable.

The proposed approach rests on three commitments. First, intent should be externalized into constraint artifacts that can be reviewed, reused, and versioned, shifting interaction from transient prompting to durable specification. Second, generation should be paired with verification so violations are surfaced early-within the same workspace where design choices are made-reducing downstream “hidden rework.” Third, outputs should include traceable rationales and targeted repair suggestions so users can calibrate trust and focus review effort where it matters most. These commitments align with core human-centered AI and human–AI interaction principles emphasizing transparency, user control, and efficient correction [1], [2].

A constraint-centered workflow also strengthens handoff between design and engineering by expressing intent in a form that maps to both UI structure and engineering rules. Versioned constraints create a decision history that supports later maintenance and reduces reliance on institutional memory as teams and requirements change.

However, bidirectional alignment introduces challenges. Over-constraint can reduce exploration, motivating the use of soft constraints and tooling that helps users decide what is negotiable. Constraint management can also add complexity, requiring progressive disclosure and role-appropriate

controls. Finally, because not all UI quality dimensions are fully verifiable, the system must communicate uncertainty carefully to avoid overstating confidence and to support appropriate trust calibration [4].

Responsible generation remains essential. GenUI tools should integrate accessibility and policy checks as first-class capabilities [10] and preserve provenance so teams can distinguish policy-driven requirements from user instructions and model inferences. Looking ahead, the most practical GenUI systems will likely be hybrid: combining generation with constraints, interactive editing, and established practices such as linting and testing. Future work should explore richer yet usable specification languages, methods for learning constraints from existing products without inheriting legacy errors, and collaborative workflows that support multiple stakeholders editing shared intent representations-validated through longitudinal studies that measure maintainability over months, not minutes [12].

ACKNOWLEDGEMENT

GenUI will be most valuable when it supports not only rapid creation but also reliable integration into the social and technical realities of product development. Prompt-only generation has demonstrated that plausible UIs can be produced quickly. The remaining challenge is to make those outputs legible and sustainable: teams need to understand what was assumed, how the output relates to organizational constraints, and how to correct misalignment without starting over. This paper has argued that these needs are best addressed by treating GenUI as an alignment problem, specifically a bidirectional alignment problem in which both system and user adapt through explicit negotiation. In this framing, the goal is not to eliminate iteration but to make iteration structured, checkable, and accountable.

The proposed approach centers on three commitments. First, intent should be externalized into constraint artifacts that can be reviewed and reused. This changes the unit of interaction from a transient prompt to a durable specification. Second, generation should be paired with verification so that violations are discovered early, in the same workspace where design choices are made. This reduces hidden rework by shifting audits forward in the pipeline. Third, outputs should be accompanied by traceable rationales and targeted repair suggestions, enabling users to calibrate trust and focus attention where it matters. These commitments align with established principles in human-centered AI and human-AI interaction, which emphasize control, transparency, and support for correction [1], [2].
A constraint-based workflow also improves handoff between design and engineering. Many conflicts between designers and developers arise because intent is not represented in a form both can work with. Designers communicate through screens and annotations; engineers communicate through code and tests. Constraints can bridge these representations by expressing intent in a form that maps to both UI structure and engineering rules. When constraints are versioned, they create a history of decisions that helps teams revisit trade-offs without reconstructing past reasoning. This is especially important for UI work, where product requirements evolve and where teams often change over time. A GenUI tool that exports a decision record alongside code can reduce the burden of institutional memory.

Nevertheless, bidirectional alignment introduces its own challenges. One risk is over-constraint: if users encode requirements too rigidly, the system may produce conservative outputs that lack creativity or fail to explore meaningful alternatives. This suggests the need for mechanisms that support “soft” constraints and for UI that helps users decide which constraints are negotiable. Another risk is complexity: if constraint management is too demanding, users may revert to prompt churn. Tool design must therefore emphasize progressive disclosure, providing simple controls for novices while enabling detailed constraint editing for experts. A third challenge is the integration of probabilistic and deterministic checks. Not all aspects of UI quality can be verified automatically, and the system must avoid presenting uncertain judgments as facts. This is where careful communication of uncertainty is essential for trust calibration [4].

Responsible generation remains central. UIs can shape user behavior and may affect access, privacy, and well-being. GenUI tools should integrate accessibility and policy checks as first-class features, making it easier to produce inclusive interfaces by default [10]. They should also support auditing and governance by preserving provenance: users should know whether a constraint came from policy, from user instruction, or from model inference. This is not only a technical feature but an ethical one, because it influences accountability when harmful or biased outcomes occur. As human-centered AI work emphasizes, beneficial systems must be transparent and interrogable, particularly when deployed in contexts where mistakes have real consequences [1].

The future of GenUI is likely to be hybrid rather than purely generative. High-performing workflows will combine generation with formal constraints, interactive editing, and established engineering practices like linting and testing. Program synthesis research has long explored the value of constraints and partial specifications [12], and GenUI provides a contemporary arena in which these ideas can become usable by practitioners. Future work should explore richer specification languages that remain accessible to designers, better methods for learning constraints from existing products without propagating legacy errors, and collaborative workflows that support multiple stakeholders editing a shared intent model. Longitudinal studies will be particularly important to understand whether bidirectional

alignment reduces maintenance cost over months, not just minutes.

In conclusion, bidirectional human AI alignment offers a practical path for making GenUI tools fit the realities of design and development work. By converting intent into checkable constraints, generating diverse candidates, verifying outputs, and providing traceable explanations with incremental repair, GenUI can become more than a rapid sketch generator. It can become a reliable partner in the design-to-code pipeline: fast, yes, but also accountable, legible, and sustainable.

REFERENCES

[1] [B. Shneiderman, *Human-Centered AI*. Oxford University Press, 2022.

[2] S. Amershi, D. Weld, M. Vorvoreanu, A. Fourney, B. Nushi, P. Collisson, J. Suh, S. Iqbal, P. N. Bennett, K. Inkpen, T. Teevan, R. Kikin-Gil, and E. Horvitz, "Guidelines for Human-AI Interaction," in *Proc. CHI*, 2019, doi:10.1145/3290605.3300233.

[3] E. Horvitz, "Principles of Mixed-Initiative User Interfaces," in *Proc. CHI*, 1999, pp. 159–166, doi:10.1145/302979.303030.

[4] A. Holzinger, "From Machine Learning to Explainable AI," in *World Symposium on Digital Intelligence for Systems and Machines*, 2018, doi:10.1109/DISM.2018.8351327.

[5] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why Should I Trust You? Explaining the Predictions of Any Classifier," in *Proc. KDD*, 2016, pp. 1135–1144, doi:10.1145/2939672.2939778.

[6] S. Amershi, M. Cakmak, W. Knox, and T. Kulesza, "Power to the People: The Role of Humans in Interactive Machine Learning," *AI Magazine*, vol. 35, no. 4, 2014, pp. 105–120.

[7] J. Nielsen, "Heuristic Evaluation," in *Usability Inspection Methods*, J. Nielsen and R. L. Mack, Eds. Wiley, 1994.

[8] J. Brooke, "SUS: A Quick and Dirty Usability Scale," in *Usability Evaluation in Industry*, P. W. Jordan, B. Thomas, B. A. Weerdmeester, and I. L. McClelland, Eds. Taylor & Francis, 1996.

[9] T. Beltramelli, "pix2code: Generating Code from a Graphical User Interface Screenshot," *arXiv preprint* arXiv:1705.07962, 2017.

[10] W3C, "Web Content Accessibility Guidelines (WCAG) 2.2," World Wide Web Consortium, 2023.

[11] H. Shen, T. Knearem, R. Ghosh, K. Alkiek, K. Krishna, Y. Liu, and D. Jurgens, "Towards Bidirectional Human-AI Alignment: A Systematic Review for Clarifications, Framework, and Future Directions," *arXiv preprint* arXiv:2406.09264, 2024.

[12] S. Gulwani, O. Polozov, and R. Singh, "Program Synthesis," *Foundations and Trends in Programming Languages*, vol. 4, nos. 1–2, 2017, pp. 1–119.

[13] X. A. Chen, T. Knearem, and Y. Li, "The GenUI Study: Exploring the Design of Generative UI Tools to Support UX Practitioners and Beyond," in *Proc. ACM Designing Interactive Systems Conference (DIS)*, 2025, pp. 1179–1196.

[14] T. Knearem, M. Khwaja, Y. Gao, F. Bentley, and C. E. Kliman-Silver, "Exploring the Future of Design Tooling: The Role of Artificial Intelligence in Tools for User Experience Professionals," in *Extended Abstracts of CHI*, 2023.

[15] T. Miller, "Explanation in Artificial Intelligence: Insights from the Social Sciences," *Artificial Intelligence*, vol. 267, 2019, pp. 1–38.