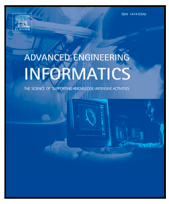
Advanced Engineering Informatics 77 (2027) 105165

Contents lists available at ScienceDirect

Advanced Engineering Informatics

journal homepage: www.elsevier.com/locate/aei



Full length article

GenAlign: Visual knowledge graphs as interactive substrates for property-aligned generative design

Jiin Choi [a,b], Minkyung Seo [a], Kyung Hoon Hyun [a,b,*]

[a] Department of Interior Architecture Design, Hanyang University, Seoul, 04763, Republic of Korea
[b] Human-Centered AI Design Institute, Hanyang University, Seoul, 04763, Republic of Korea

ARTICLE INFO

Dataset link: https://cosjiimos.github.io/GenAlign/

Keywords:
Visual knowledge graph
Segment-based property edit generation
Generative AI
Semantic disambiguation

ABSTRACT

Designers often describe goals using ambiguous adjectives (e.g., *modern*); however, generative AI systems struggle to navigate this ambiguity, creating misalignment between language, intent, and outcomes. When the intended meaning remains implicit and unstructured, design generation cannot be guided precisely, resulting in semantic drift. Rather than logging past user operations, we present *GenAlign*, a graph-backed, human-in-the-loop design support system that externalizes cognitive visual attributes as segment–property bindings in a visual knowledge graph (VKG). *GenAlign* externalizes abstract attributes as segment–property bindings and maintains them as inspectable, reusable, and personalized bindings through real-time VKG updates. In a within-subject study with 24 professional designers, *GenAlign* transformed abstract attributes into reusable property vocabularies, supported personalized design strategies, and reduced per-step semantic drift across iterations. We discuss how VKG functions as an operational boundary object that bridges interpretive and executable design representations rather than eliminates ambiguity.

1. Introduction

Recent work in design support is shifting from generating large numbers of alternatives toward modeling individual designers' evolving intent and preferences to provide tailored assistance [1–3]. From this perspective, a central challenge is how systems can recognize, represent, and reuse subjective intent.

Semantic ambiguity in abstract language complicates intent interpretation: while ambiguity is productive in early stages for divergence and reframing, it benefits from progressive stabilization through shared boundary objects (e.g., mood boards or exemplars) that make mappings between adjectives and properties inspectable and revisable [4–6]. Designers often begin by using adjectives such as modern or classic to sketch a rough stylistic direction before committing to concrete properties [7–11]. When a designer enters a prompt such as "modern sofa", a generative model must select one interpretation among many, ranging from thin metal legs and minimal forms to thick leather cushions and sharp angles. The difficulty is that the same adjective can denote different concrete properties (e.g., material, color, geometry) across designers and even for the same designer depending on context (e.g., sofa vs. table). Recent generative design tools address this ambiguity in different ways: micro-prompts decompose long prompts into tags [12], reference-based systems recombine selected image patches [13], and inpainting tools enable direct local edits via masking [14]. However, these approaches rely on unstructured histories that do not accumulate personalized meaning: prompt conversational logs or image galleries show *what* changed, but not *why* it changed or which attributes were bound to which parts. As a result, interpretations such as a previously defined "modern leg" remain implicit and unreusable, requiring designers to restate intent when switching objects (e.g., from table to chair). Designers therefore resort to repeated prompt refinements (e.g., "more modern", "make the legs metal") to persuade the model, often causing unintended changes and semantic drift across iterations [15–17].

Rather than logging past user operations, we present *GenAlign*, an interactive GenAI system that structures and aligns cognitive visual attributes and properties using a visual knowledge graph (VKG) (Fig. 1): *GenAlign* decomposes images into semantic parts and surfaces each part's visual attributes, maintaining these segment–property bindings as a continuously updated state. Designers interact with this state by reusing attributes across parts through drag-and-drop, editing properties locally on selected segments, and conditioning new generations on the currently associated visual attributes, without repeatedly rewriting prompts. The VKG can be reorganized around either segments or properties and filtered by visual attributes, allowing designers to inspect and revise how abstract adjectives are concretely bound to image parts during exploration.

* Corresponding author at: Department of Interior Architecture Design, Hanyang University, Seoul, 04763, Republic of Korea.
E-mail address: hoonhello@hanyang.ac.kr (K.H. Hyun).

https://doi.org/10.1016/j.aei.2026.105165
Received 3 February 2026; Received in revised form 7 July 2026; Accepted 14 August 2026

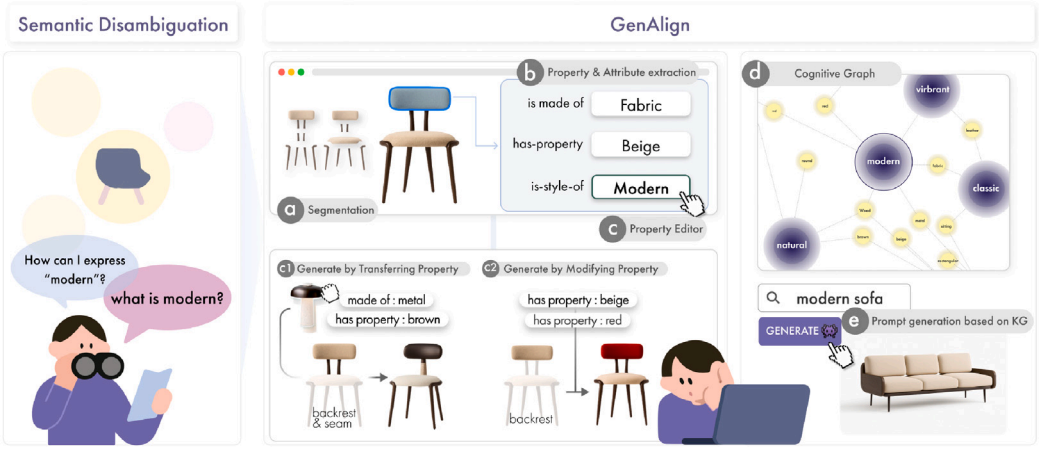


Fig. 1. GenAlign overview. An interactive design exploration system that aligns prompt-based image generation and segment-level editing according to the intent of each designer using a visual knowledge graph (VKG).

We conceptualize the VKG as an *operational boundary object* [18]: an artifact that is simultaneously interpretive (externalizes ambiguous intent for inspection), operable (directly manipulable to condition generation), and accumulative (preserves decisions across iterations and objects). Unlike traditional boundary objects in design — mood boards, sketches, material samples — that coordinate understanding but cannot execute decisions [19], VKG both represents intent and operationalizes it as generation constraints. This dual nature enables what we term semantic anchoring: once a designer establishes that "modern leg" means `thin + metal + angular`, this interpretation persists as a computational constraint rather than dissipating as ephemeral language.

To evaluate GenAlign, we ran a within-subject study with 24 professional designers, addressing three questions: whether *GenAlign* improves designers' **control** over generative outputs, how it affects **exploration and refinement**, and how designers **experience** structured versus linguistic intent representations. *GenAlign* outperformed a baseline interface without VKG support by producing more semantic triples and reducing semantic drift across iterative generations. While the number of prompt edits did not differ significantly between conditions, designers generated more design outcomes with *GenAlign*, using prompts as initial anchors followed by segment-level refinement. Participants reported that externalizing abstract adjectives as reusable segment–property bindings (e.g., leg—made-of→ metal (segment—predicate→ property)) enabled them to build and consistently apply a personalized design vocabulary, improving both exploration efficiency and perceived intent alignment.

In summary, the major contributions of this work are:

- **Segment–property binding as an operable design** state: an interaction design that externalizes abstract prompts as inspectable, transferable, and provenance-tracked bindings in a designer-specific graph.
- **Real-time VKG construction and update pipeline** that integrates segmentation, property extraction, and visual attribute-conditioned inference into a unified graph structure, updated incrementally through user interactions.
- **Empirical characterization of semantic disambiguation** from a comparative study (N = 24 professional designers) demonstrating that VKG-based editing shifts interaction from linguistic negotiation to property composition, with improvements in controllability, traceability and perceived intent alignment.

2. Related work

GenAlign externalizes abstract attributes as structured, reusable knowledge that persists across design iterations. We review three strands: (2.1) generative design support and limitations in accumulating intent, (2.2) knowledge graphs as interactive substrates for capturing designer-specific semantics, and (2.3) visual representations as interpretive yet operable boundary objects.

2.1. Generative design support

Text-to-image GenAI models struggle to align outputs with user intent when descriptors are ambiguous. A user typing "modern sofa" expects specific properties (e.g., thin metal legs, matte surfaces) but the model interprets "modern" differently each time, especially across object types or iterations. Iterative prompt refinement systems allow conversational feedback [20,21]. PromptMagician [15] suggests alternative keywords by visualizing prompt embeddings, while GenTune [22] adapts generation parameters to user goals. However, each refinement is treated as a new textual instruction with no mechanism to accumulate decisions. Switching from "modern sofa with thin metal legs" to "modern table" forces users to re-specify "thin metal legs", risking semantic contamination where prior context unpredictably interferes with new prompts [15]. Structured input approaches decompose intent into smaller units. Intent tagging [12] breaks slide authoring into micro-prompts, while GenPara [23] exposes parametric controls for color and style. These improve granularity but remain object-level—users cannot specify "metal for legs, fabric for seat" within one object. Specifications are not persistent: changing objects requires re-entering all parameters, as systems do not learn user-specific mappings between abstract attributes and concrete properties. Reference-based guidance reduces articulation burden. AIdeation [13] recombines selected image patches, while DreamCrafter [24] allows VR sketching to resynthesize 3D scenes. References are interpreted globally—selecting an image for its "metal legs" may also copy its color or proportions. Users cannot isolate which aspects matter. Furthermore, personalization is operationalized by conditioning generation on user preferences and style themes [25–28]. Recent surveys confirm that LLMs are increasingly applied to design knowledge extraction and concept generation [29], and deep learning methods can automatically identify design concepts from generative outputs [30]; however, these approaches remain output-level rather than providing persistent, part-scoped constraints

A separate line of work addresses output-level alignment through preference learning. Reinforcement learning from human feedback (RLHF) [31] and direct preference optimization (DPO) [32] train models on preference signals to shift output distributions toward user-preferred results. While effective for general stylistic consistency, these approaches optimize latent model parameters rather than exposing an inspectable representation that designers can directly manipulate. A designer cannot query *why* the model interpreted "modern" as thin

metal legs, nor override this interpretation for specific parts while preserving it for others. The alignment remains implicit—encoded in weights, not externalized as editable structure. Similarly, policy-optimized pipelines [33] automate prompt selection to improve output quality but treat the optimization as a black-box process invisible to the designer.

These four approaches — prompt refinement, structured input, reference-based guidance, and preference-aware generation — share one shortcoming: none exposes a designer-operable representation of semantic decisions. Prompt refinement treats each turn as isolated text, so "thin metal legs" defined for a sofa does not carry to a table [15,20–22]. Structured input adds granularity but resets when the object changes [12,23]. Reference-based guidance interprets selections globally, preventing users from isolating which aspect should transfer [13,24]. Preference-aware methods (RLHF, DPO, policy-optimized pipelines) encode learned interpretations in weights rather than as editable structure [31–33]. *GenAlign* closes this gap by binding abstract adjectives to concrete properties at the segment level: editing "modern sofa" records explicit triples—`leg-has-property→ thin`, `leg-made-of→ metal`—that are reusable, inspectable, and provenance-tracked rather than drifting with each rephrase.

2.2. Knowledge graphs as interactive substrates for design

Knowledge graphs structure entities and relationships for reasoning, but most applications use static, text-only graphs that do not evolve with user interactions or capture personal semantic mappings [34–39]. A critical distinction separates GenAlign's VKG from these applications. In typical deployments, knowledge graphs are prebuilt from curated datasets or extracted from document corpora, then served as retrieval backends—users query the graph but do not construct or restructure it through their interactions [34,40–42]. Even interactive KG systems such as LinkQ [39] allow natural-language queries over existing graphs without enabling users to modify structure directly. GenAlign's VKG, by contrast, is a personalized graph gradually assembled from each designer's interactions. The graph starts empty and grows incrementally through `save_seg`, `edit_attr`, and `transfer` actions, making it a direct manipulation substrate rather than a reference database. Each designer's VKG captures divergent, context-dependent meanings — one designer's "modern" may comprise `thin + metal + angular` while another's comprises `glossy + curved + plastic` — that static ontologies like VisualSem [43] cannot represent. Spinneret [44] feeds non-obvious associations from pre-built text KGs into brainstorming. VisualSem [43] provides a visual-semantic ontology (e.g., chair—has-part→ leg). These graphs are fixed datasets—when one designer interprets "modern" as thin metal and another as glossy plastic, VisualSem cannot represent these divergent, context-dependent meanings. Research on ontology evolution [45] and dynamic KGs [46,47] tracks structural changes (drift, growth) but focuses on automated updates from data streams, not interactive user construction. LLM-based KG generation [48] extracts triples from text but remains retrospective—built from completed documents, not shaped incrementally through design actions. Beyond Entities [49] extends KG content to visual attributes, but the graph remains a reference layer for retrieval, not an editing substrate. LinkQ [39] enables natural language queries over KGs, yet users cannot modify structure directly—they consume through search rather than construct through interaction. Recent work combines LLMs with knowledge graphs to construct and reason over domain knowledge from multimodal sources [40–42,50], and ontology-augmented frameworks employ structured reasoning for early-stage design exploration [51]; however, these graphs typically serve as backend resources rather than interactive, provenance-preserving structures coupled to segment-level generative editing. *GenAlign* treats VKG as a living design artifact updating with every action. Saving a segment, editing a property, or transferring an attribute immediately updates the graph. Whereas prior knowledge-graph systems serve as retrieval backends over prebuilt graphs [39,43,49] or scaffolding templates queried by users [36,44], GenAlign's VKG is a personalized, interactively constructed substrate that captures emergent, designer-specific semantics through incremental save/edit/transfer actions, and that directly conditions subsequent generation rather than merely informing it.

2.3. Visual representations for interpretive design work

Design relies on external representations — sketches, mood boards — to negotiate ambiguous meanings. These function as boundary objects [18] and epistemic objects [19] mediating interpretation, but recent generative tools treat them as retrospective displays rather than operable substrates. Mood boards [52,53] and sketches [54,55] externalize partial ideas open to reinterpretation. However, they are typically decoupled from generative tools—designers create mood boards in one app, then manually translate aesthetics into prompts for another, risking semantic loss [56]. Semantic interaction [57] lets visual manipulation (repositioning data points) implicitly update computational models. Luminate [58] maps LLM ideas along semantic axes, MetaMap [59] positions images on multi-dimensional canvases, IdeationWeb [60] graphs edit lineage. These make history visible but read-only—users inspect past decisions but cannot manipulate them to condition new generations. Repositioning an image in MetaMap reflects preference but does not extract properties ("glossy", "curved") for reuse. PromptMap [61] and Imagination Vellum [55] organize prompts spatially with version history. This improves traceability but prompts remain text strings, not structured semantics. A designer using "modern, sleek sofa" in iteration 5 cannot transfer the system's interpretation of "modern" (thin legs) to iteration 12 (a table) without re-typing and risking reinterpretation. *GenAlign* positions VKG as simultaneously interpretive (externalizes ambiguous intent) and operable (directly manipulable). This transforms the graph from passive record to active design medium, each interaction updates the VKG, which conditions subsequent operations.

These strategies reduce the burden of articulation because micro-prompts are shorter than full sentences, references are easier to pick than to describe, and sketches bypass language. However, interpretation of these signals remains global, not personal: a minimal tag, chosen reference, or rough sketch is translated in the same manner for every user. The system does not capture differences in designers' visual vocabularies.

Across all three strands — generative design support, knowledge graphs, and visual representations — existing systems primarily support search and heuristic exploration: designers browse results, accumulate examples, and hope models infer preferred patterns. GenAlign transforms this process by turning design interactions into persistent, structured, and reusable vocabularies. Design decisions do not dissipate as ephemeral prompt strings or unlinked image collections; instead, they accumulate as semantic triples in a graph that both records intent and conditions subsequent operations.

More specifically, existing approaches share a common limitation: they do not provide a persistent, designer-operable representation of semantic decisions. Prompt refinement systems [15,20,21] treat each iteration as independent text; structured input approaches [12,23] improve granularity but remain object-level and ephemeral; reference-based systems [13,24] interpret selections globally without isolating transferable properties; knowledge graphs [39,43,44] serve as retrieval backends rather than interaction substrates; visual history tools [55,59,60] make decisions visible but not manipulable; and alignment techniques (RLHF, DPO, preference learning [33]) optimize output distributions without exposing intermediate representations. *GenAlign* introduces a *mediation representation* that combines three properties absent from these approaches: *operability* (segment–property bindings are directly editable, not latent parameters), *visibility* (every semantic decision is inspectable as a graph node and edge), and *accumulability* (decisions persist and compound across iterations rather than resetting with each generation).

3. Design requirements: Disambiguating intent through segment–property decomposition

The preceding review reveals that existing systems treat design interactions as disposable queries rather than cumulative knowledge; the following requirements specify what a persistent, structured, and reusable representation must support to close this gap. Current generative design systems struggle to align outputs with user intent because property specifications are ephemeral, non-transferable, and globally interpreted. Prior work has documented how prompt-based workflows lead to semantic drift [15], require repetitive re-articulation [16], and fail to capture designer-specific meanings [5,6]. Semantic disambiguation requires decomposing ambiguous attributes into manipulable units. For example, "Modern sofa" is ambiguous at three levels: **which parts** embody "modern" (legs or seat?), through **which properties** (material or geometry?), and according to **whose interpretation** (thin metal vs. glossy plastic). Text-to-image models apply global interpretations, collapsing these distinctions, preventing both part-level control and personalized semantics.

Segment–property bindings address this by enabling three critical operations: (1) localized specification—bind properties to specific parts ('leg'—'metal') rather than entire objects, (2) explicit reuse—transfer property bundles (thin + metal + angular) across objects without reinterpretation, and (3) inspectable mappings—externalize how attributes decompose personally (modern = 'leg'—'thin', 'surface'—'matte'). This structural unit is compositional (parts combine into wholes), transferable (properties carry across contexts), and interpretable (bindings reveal decisions). We propose the following design requirements to address these ambiguities.

DR1: Persistent Properties. Studies of generative AI usage reveal that designers spend significant effort re-explaining intent across iterations [20,21]. When a designer refines "modern sofa" to have thin metal legs, then switches to "modern table", existing systems treat each prompt as an isolated instruction—prior semantic decisions evaporate. This forces designers into repetitive linguistic negotiation rather than cumulative refinement [16]. Research on design conversation [56] shows that translation work compounds when shared meanings cannot be anchored in stable representations. Bind abstract attributes to concrete properties in a persistent structure that survives across iterations and object changes, maintaining semantic consistency without re-articulation—enabling what Endert et al. [57] term "semantic interaction" where prior interpretations inform future operations.

DR2: Reusable Properties. Professional design practice relies heavily on reusing established visual languages across projects [52,53], yet generative tools lack support for systematic property transfer. A designer cannot apply a "metal leg" defined for a table to a chair without re-prompting. Inpainting tools [14] enable region-specific edits but treat each mask as an independent operation—there is no semantic memory linking "metal leg on table" to "metal leg on chair". Reference-based systems [13] allow patch recombination but interpret selections globally rather than isolating transferable properties. This contradicts established design theory: Schön [62] describes how designers build repertoires of reusable "moves", while Cross [63] emphasizes design knowledge as accumulated patterns. Enable cross-object property transfer through direct manipulation (e.g., drag-and-drop) where properties carry their semantic meaning to new contexts—what Star [18] terms a "boundary object" that maintains interpretive flexibility while supporting coordination across different design instances.

DR3: Personalized Properties. Abstract attributes have no universal visual grounding—"modern" varies across designers, domains, and contexts [7,10]. One designer's "modern" (thin, metal, angular) differs from another's (glossy, curved, plastic). Recent systems address personalization through prompt engineering [15] or preference learning [33], but these operate at output level—optimizing consistency without extracting manipulable property-level mappings. Prompt refinement systems [15,22] optimize for average interpretations but cannot capture idiosyncratic mappings. Even with user examples [13], systems lack mechanisms to extract and preserve designer-specific bindings across sessions. Capture and maintain designer-specific attribute-property bindings that evolve through interaction, enabling the system to learn how each user grounds abstract terms—creating an "epistemic object" [19] that documents and shapes emerging knowledge.

These requirements converge on a fundamental gap: existing generative systems lack an intermediate representation that is simultaneously interpretive (captures ambiguous intent), operable (directly manipulable), and cumulative (preserves decisions across iterations). This gap motivates *GenAlign*'s core mechanism: a VKG that externalizes segment–property bindings as nodes and edges, updated incrementally through user actions (save, edit, transfer). Drawing on research showing that structured representations improve interpretive work [57,64], the VKG serves as a persistent, reusable, and personalized semantic layer conditioning all subsequent generation and editing operations (Section 4).

4. Genalign

4.1. System overview

We introduce *GenAlign*, a system that leverages VKGs to continually refine property-conditioned image generation and segment-level editing in step with each designer's evolving intent (Fig. 2). *GenAlign* operationalizes the three design requirements (Section 3) through VKG's architectural design: Addressing **DR1** (*Persistent Properties*): VKG maintains property bindings as persistent state rather than ephemeral prompts. Each segment–property triple (e.g., leg—made-of→ metal) is stored with provenance metadata, enabling bindings to survive across iterations and object changes without re-articulation. Addressing **DR2** (*Reusable Properties*): The property editor (Fig. 3) and drag-and-drop transfer mechanism enable systematic cross-object reuse. Designers transfer saved segment bundles (e.g., table-leg: metal, thin, angular) directly onto new objects, maintaining semantic meaning without linguistic reinterpretation (Fig. 4). Addressing **DR3** (*Personalized Properties*): VKG accumulates designer-specific mappings through interaction (Fig. 5). Each action updates visual attribute-property associations (e.g., modern—is-style-of→ metal, thin, matte), capturing personal semantics that differ across designers yet remain computationally interpretable for generation conditioning. These requirements are realized through three coupled subsystems (Section 5): (1) a VKG construction pipeline extracting segment–property bindings from images, (2) segment–property interaction & generation supporting drag-and-drop transfer, direct editing, and VKG-conditioned generation, and (3) an interactive graph visualization enabling exploration, filtering, and provenance tracking (Table 1). A demonstration video of the system workflow is available on the project page.[1]

4.2. Core interactions and design workflow

The workflow proceeds as follows: search reference images → save segments with properties → edit or transfer properties via direct manipulation → trigger generation conditioned on the updated VKG. Each action incrementally updates the graph, which serves as both a visual design artifact and an executable semantic constraint. We demonstrate *GenAlign*'s capabilities through a user scenario that illustrates its main functionalities. Note that users do not need to follow these steps as *GenAlign* supports free-form exploration.

Initial query via CLIP-based retrieval. Phillip designs a "modern, minimal sofa". He types "modern minimal sofa" in the search field

[1] https://cosjiimos.github.io/GenAlign/.

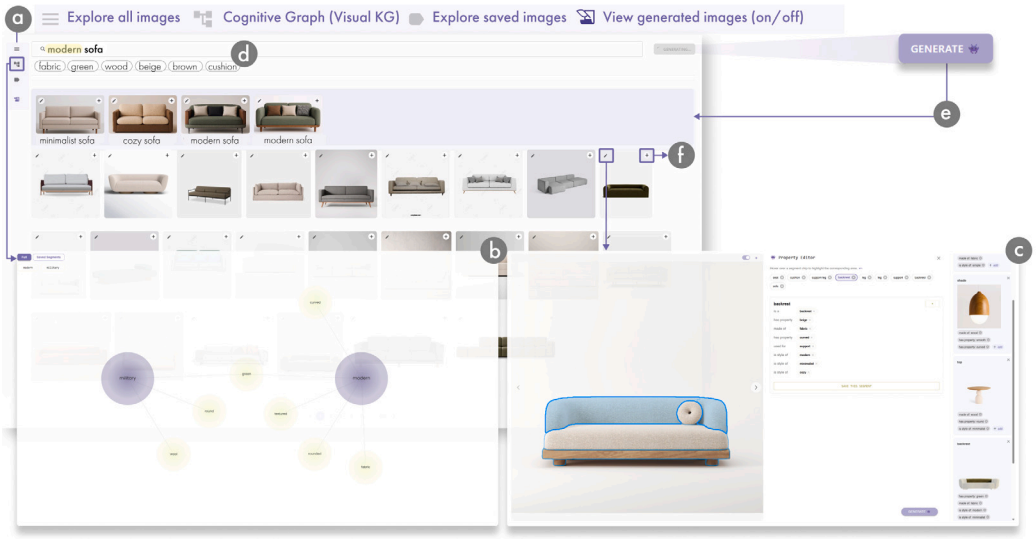


Fig. 2. System workflow UI with labeled panels: (a) navigation tabs, (b) a live cognitive graph of styles/segments/properties, (c) a part-aware property editor, (d) CLIP-based retrieval with property suggestions, (e) prompt-based generation conditioned on the user-specific VKG, and (f) save actions for selected results.

Table 1
Design requirements → System components.

DR	System component	Mechanism	UI Ref.
DR1: Persistent	VKG state + provenance metadata	Triples stored with timestamps; survive across iterations without re-articulation	Fig. 5 (VKG visualization)
DR2: Reusable	Property editor + drag-and-drop transfer	Cross-object transfer via delta computation; segment bundles carry semantic meaning	Fig. 3 (editor), Fig. 4 (transfer)
DR3: Personalized	VKG accumulation + HBI (Section 5.3.2)	Designer-specific style→property via Dirichlet-multinomial posterior; top-k selection	Fig. 2-b, e

(Fig. 2-a). *GenAlign* computes a CLIP text embedding and returns top-ranked images by cosine similarity from a database of images.[2,3,4] Phillip browses results, clicks "save" or "edit" on promising candidates, triggering segmentation and property extraction.

Segment-based property editing with drag-and-drop transfer. Phillip opens a saved sofa in the property editor (Fig. 4-a), which displays decomposed segments as chips with editable property slots (has-property, made-of, is-style-of). In the Saved Segments panel, he sees a previously saved "leg" segment with properties glossy, green, plastic, rectangular. To apply these properties to another part, Phillip drags this saved leg segment chip and drops it onto the sofa's backrest segment. *GenAlign* computes the property delta (Δ = source_props target_props), generates an imperative instruction ("For the backrest, change to: glossy surface, green color, plastic material, rectangular shape. Preserve other parts"), and triggers mask-local inpainting. The sofa regenerates with the backrest updated to match the leg properties while other parts remain intact. The VKG records this transfer (backrest—has-property→ glossy, green; backrest—made-of→ plastic) with provenance (timestamp, source=leg, action = transfer).

Direct property modification for iterative refinement. Phillip then selects the backrest segment and clicks the made-of property chip showing "plastic" (Fig. 4-c,d). He edits it directly by changing made-of: plastic to made-of: leather in the property editor. *GenAlign* computes the delta, generates an instruction, rasterizes the segment mask, and sends both to the system. The result (Fig. 4-e) shows the backrest updated to leather material while other segments preserve their original properties. The VKG immediately updates this change (backrest—made-of→ leather), maintaining a live semantic record of the design state.

Cross-object property reuse without re-prompting. Refining his concept, Phillip switches to designing an armchair. Rather than starting from scratch with a new prompt, he opens the Saved Segments panel and drags the previously saved "leg" segment from his sofa design onto the armchair's leg area. *GenAlign* transfers the entire property bundle (has-property: glossy, green; made-of: plastic; rectangular shape) without linguistic reinterpretation, maintaining exact semantic meaning. The VKG updates with provenance = transfer, enabling systematic reuse of designer-specific vocabularies across different objects. The armchair generates with consistent properties, preserving Phillip's personal interpretation established during earlier explorations.

VKG-conditioned generation for consistent exploration. Instead of manually transferring each property, Phillip opens the generation panel and selects style nodes "modern" and "minimal" from the VKG visualization. *GenAlign* serializes relevant properties by computing $P(\text{property} \mid \text{style}) \propto \text{count}(\text{style-is-style-of-segment-has-property-property})$, retrieving top-k attributes frequently bound to these styles in Phillip's VKG. The system constructs a prompt: "armchair, modern, minimal, [retrieved properties]" and generates candidates. The results align with Phillip's established design language without requiring him to re-specify each property.

Interactive graph visualization for exploration and provenance. Phillip opens the VKG visualization tab (Fig. 5) to review accumulated bindings. The graph displays style nodes (modern, minimal), segment nodes (leg, seat, backrest), and property nodes (glossy, green, plastic, leather) with connecting edges. He clicks a style chip, and the graph filters to show only nodes within 2 hops, revealing which properties consistently map to that style across multiple saved segments. This provenance enables Phillip to verify the origin of each property binding and understand how his personal design vocabulary evolved through iterative explorations.

Expanding the property space through dynamic updates. Finally, Phillip wonders if "minimal" could incorporate organic curves

[2] Multidomain Image Characteristics: https://www.kaggle.com/datasets/realtimear/multidomain-image-characteristics-dataset.
[3] Furniture-101: https://www.kaggle.com/datasets/kerrit/furniture-101.
[4] Furniture Detector: https://www.kaggle.com/datasets/akkithetechie/furniture-detector.

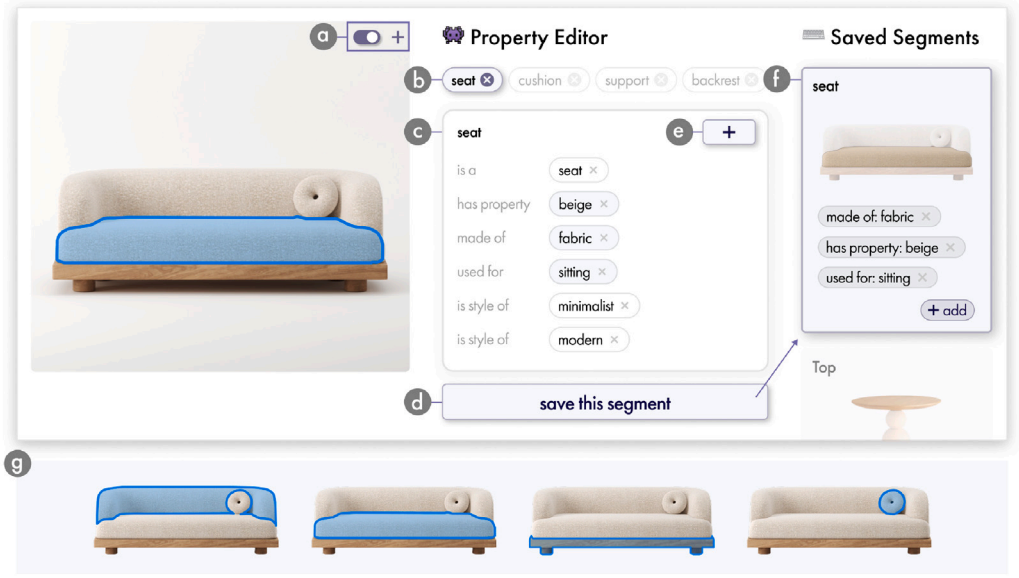


Fig. 3. Property editor and segment library: (a) segment mode/save, (b) segment preview, (c) segment properties, (d) saving the segment with user-specific properties and attributes, (e) adding the properties, (f) saving the segments, (g) sample results; enabling part-level property assignment, editing, and reuse.

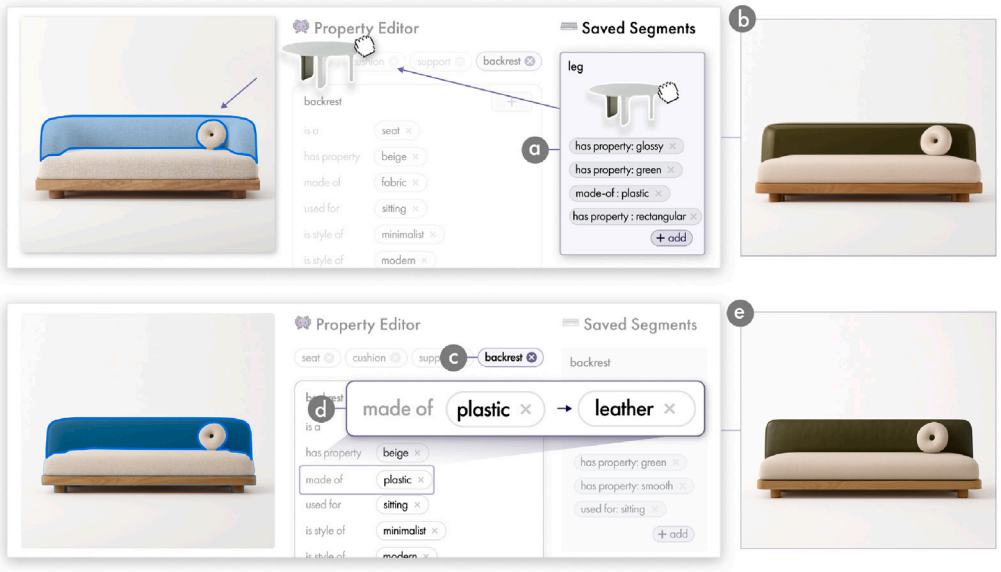


Fig. 4. Generate design by drag-and-drop transfer and modify properties: (a) drag and drop the saved segment, (b) result of sofa generation after applying the attributes `has-property: glossy`, `has-property: green`, `made-of: plastic`, and `has-property: rectangular` via drag-and-drop, (c) selecting a segment element, (d) direct editing of a specific element in the property editor, and (e) result of sofa generation after changing the attributes `made-of: plastic` to `made-of: leather` via direct editing.

rather than strict rectilinearity. He clicks the "+" button next to the shape property options and types "curved" (Fig. 3-e). *GenAlign* extracts properties for curved forms, updates the VKG with a new branch, and generates variations combining minimal + curved. This expansion allows Phillip to explore beyond his initial property space while maintaining semantic consistency with his established bindings. The VKG grows incrementally, capturing his evolving design intent without requiring him to rebuild his vocabulary from scratch.

5. Visual knowledge graph construction and generation

This section details *GenAlign*'s technical pipeline for constructing designer-specific VKGs and conditioning generation on accumulated segment–property bindings.

5.1. VKG definition and data structure

GenAlign's technical pipeline is structured around a VKG rather than relying solely on end-to-end prompting. The VKG maintains persistent, designer-specific state at the segment level, capturing how abstract visual attributes (e.g., minimal, playful) are bound to concrete properties across tasks and iterations (Fig. 6). By turning edits into explicit triples with provenance, the VKG enables part-level controllability, reuse and recombination of properties, and visually inspectable explanations that are difficult to obtain from opaque prompt histories alone. Encoding alignment as a dynamic graph also makes semantic disambiguation measurable, allowing us to analyze growth, reuse, stabilization, and drift over time. Critically, the VKG is not a prebuilt ontology; it begins as an empty graph and is constructed entirely through the designer's

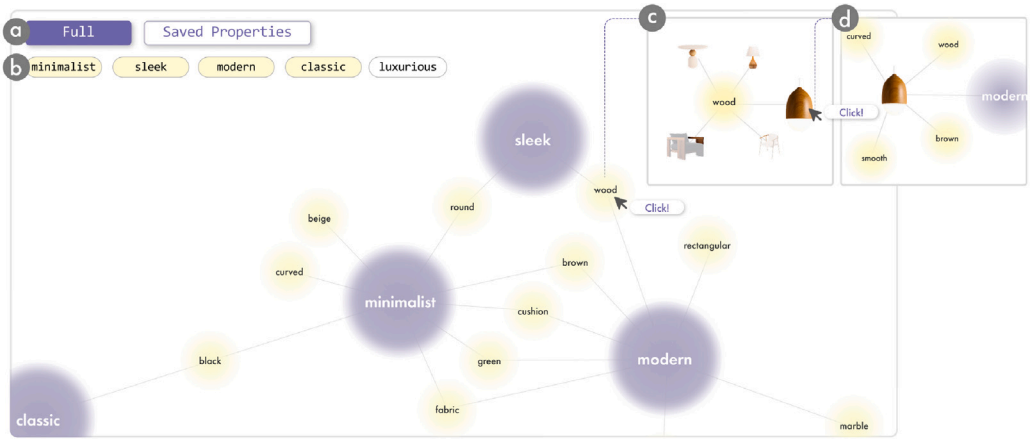


Fig. 5. Cognitive VKG visualization from saved images and segments: (a) full/saved segments, (b) attribute chips, (c) property-centered visualization, and (d) segment-centered visualization.

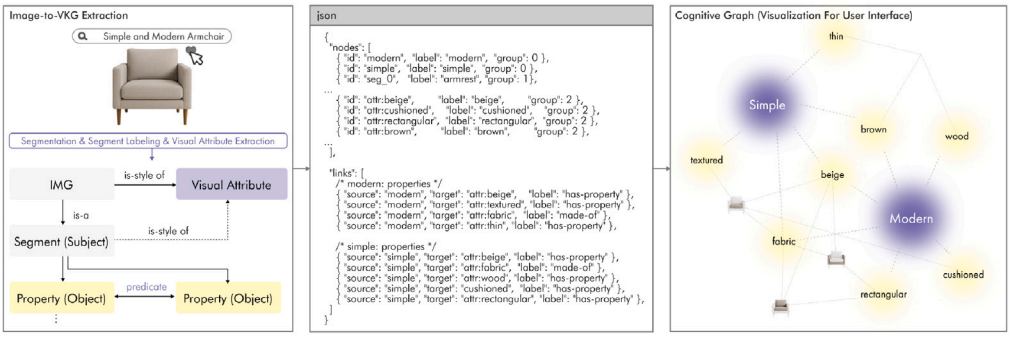


Fig. 6. VKG representation and construction pipeline. A designer-specific VKG is modeled as a multi-relational RDF-style graph with nodes for segments, visual attributes, and properties, and edges for predicates such as `has-property`. The figure shows how these triples are stored as JSON (`nodes[]` and `links[]`) and updated online with provenance as the designer interacts with the system.

interactions (`save_seg`, `edit_attr`, `transfer`, etc.), making each VKG instance a personalized semantic artifact rather than a shared reference database.

Visual knowledge graph structure. Nodes (V) include three groups:

1. *Style nodes* (group = 0) representing abstract attributes (e.g., modern, minimal, sleek);
2. *Segment nodes* (group = 1) representing part labels (e.g., sofa-leg, table-top, lamp-shade);
3. *Property nodes* (group = 2) representing concrete attributes (e.g., metal, thin, angular, matte).

Each node carries a stable `id`, a human-readable `label`, and a group identifier group $\in \{0, 1, 2\}$.

Edges (E) use 13 VisualSem [43]-aligned predicates (e.g., `is-a`, `has-part`, `made-of`, `has-property`) together with a custom `is-style-of` predicate that links segments or properties to style attributes (Table 2). Each edge represents a triple (*subject*, *predicate*, *object*), such as `leg–made-of→metal` or `sofa–is-style-of→modern`.

Provenance and traceability. Provenance metadata (P) attached to each edge includes a creation timestamp, a `source_image` indicating the origin reference, and a `user_action` describing how the binding was created (i.e., `save_seg`, `edit_attr`, or `transfer`). This provenance information enables traceability, allowing users to inspect which image and interaction produced each binding.

Hierarchy and constraints. The VKG organizes three node types in a strict subject–predicate–object hierarchy. Segment nodes (group = 1) act as editable subjects (e.g., backrest, leg); Property nodes (group = 2) serve as objects describing color, material, texture, shape, or function (e.g., wood, curved, matte); and Style nodes (group = 0) represent high-level visual attributes (e.g., modern, minimalist, sleek). The hierarchy is enforced by predicate constraints such that each segment must carry at least one `is-a` label. Segment–property relations use `has-property`, `made-of`, and `part-of/has-part`, while style attachment is expressed via `is-style-of` links from either an image or a segment to a style attribute. All edges are normalized triples (s, p, o), with de-duplicated (p, o) pairs per segment.

5.2. VKG construction pipeline

When a user selects an image, *GenAlign* constructs a VKG in four stages.

Stage 1: Foreground isolation. Given an input image I, `Rembg` [65] removes the background and produces an RGBA foreground image I_f together with a binary mask M. This preprocessing step eliminates shadows and irrelevant artifacts, yielding clean inputs for subsequent segmentation.

Stage 2: Semantic segmentation. Semantic-SAM [66] decomposes I_f into a set of candidate segments

$$S = \{s_i\}_{i=1}^{N},$$

each corresponding to an actionable object part (e.g., sofa-seat, sofa-armrest, lamp-shade). Segments are filtered by area and connectivity

Table 2
Visual knowledge graph predicates and their definitions.

Predicate	Definition	Predicate	Definition
is-a	One node is a superclass or category of another.	has-part	One node includes the other as a part.
related-to	Two nodes are generally semantically associated.	used-for	A node is employed for a specific purpose.
used-by	A node is utilized by another node.	subject-of	The node is the topic or subject of another entity.
receives-action	The node is the target of an action.	made-of	The node is composed of a particular material.
has-property	The node possesses a specific attribute or property.	gloss-related	Nodes are linked via semantic similarity in glosses.
synonym	The two nodes are synonyms.	part-of	The node forms a component of another node.
located-at	The node exists at a particular location.	is-style-of	cognitive visual attribute.

constraints such that

$$\text{area}(s_i) > 0.02 \cdot \text{area}(I_f),$$

and only valid connected components are retained. Each surviving segment is cropped and padded to form a white-backed chip s_i^{chip} for downstream property extraction.

Stage 3: Segment labeling. For each segment s_i, GPT-4o[5] assigns a human-readable kebab-case identifier ℓ_i using the prompt (Appendix E.4): *"This segment is part of [object_name]. Provide a concise label in kebab-case format (e.g., sofa-armrest, table-leg, lamp-base)"*. Each label ℓ_i is decomposed into an (object, part) pair. Segments identified as background or non-actionable elements are discarded, yielding a refined segment set:

$$S' \subseteq S$$

Stage 4: Property triple extraction. We first extract global visual attributes from the full image I and record them as style priors using `is-style-of` relations (Appendix E.5). Then, for each part-level segment $s_i \in S'$, an LLM extracts a set of VisualSem-constrained triples

$$T_i = \{(s_i, p, o)\},$$

where predicates p are drawn from a predefined set of 13 relations (e.g., `made-of`, `has-property`), augmented with a custom `is-style-of` predicate to connect attributes across design elements (Table 2). The prompt explicitly requests attributes along six design-relevant dimensions: color, material, texture, shape, function, and style (Appendix E.6).

This process yields triples such as `leg–made-of→metal`, `leg–has-property→thin`, and `leg–has-property→angular`. Redundant `is-a` triples are resolved by enforcing the object–part label, and, when applicable, injecting global style priors to avoid duplication. The result is a multi-relational, RDF-style VKG, where nodes represent segments, properties, and abstract attributes, and edges encode their relations.

The VKG is serialized as JSON using `nodes[]` and `links[]` arrays (mirrored in `triples[]`). All graph mutations are time-stamped and logged, supporting inspection, reuse, and modeling of the designer's evolving intent.

Incremental VKG update. Graph updates follow upsert and removal semantics over triple sets. Given a previous triple set T_{old} and an updated set T_{new}, the update delta is defined as

$$\Delta T = T_{\text{new}} \setminus T_{\text{old}},$$

with deprecated triples $T_{\text{old}} \setminus T_{\text{new}}$ removed and new triples in ΔT inserted with corresponding provenance metadata.

- **Save segment**: Given an extracted triple set T_s, the system upserts nodes by label (i.e., deduplicating existing nodes), inserts edges $E_s = \{(s, p, o) \mid (s, p, o) \in T_s\}$, and assigns a timestamp with `action` = save_seg.
- **Edit property**: Let T_{old} and T_{new} denote the triple sets before and after editing. The update delta is computed as

  $$\Delta T = T_{\text{new}} \setminus T_{\text{old}}.$$

  Deprecated edges in $T_{\text{old}} \setminus T_{\text{new}}$ are removed, and edges in ΔT are upserted with `action` = edit_attr and a new timestamp.
- **Transfer property**: Given a source segment s_{src} and a destination segment s_{dst}, all outgoing edges

  $$E_{\text{src}} = \{(s_{\text{src}}, p, o)\}$$

  are cloned and retargeted to form

  $$E_{\text{dst}} = \{(s_{\text{dst}}, p, o) \mid (s_{\text{src}}, p, o) \in E_{\text{src}}\}.$$

  The transferred edges are inserted with `action` = transfer and metadata `source_segment` = s_{src}.
- **Delete segment**: For a segment node s, the delete operation removes s and all edges

  $$E_s = \{(s, p, o)\}$$

  where s is the subject. Property nodes that become isolated are preserved as orphan nodes and remain reusable.

5.3. VKG-conditioned generation and editing

We treat designer-specific VKG as a graph-backed operational state and drive two complementary pathways. Text-to-image serializes relevant triples (e.g., has-property, made-of, shape) and augments them with property-conditioned priors when a request is under- or over-specified, to make abstract adjectives concrete.

5.3.1. Property edit and transfer generation

For local modifications, the system computes a triple delta between the previous and updated VKG values for the selected segment. This delta is translated into imperative instructions (e.g., "For the `back-rest`, change the `color` to `deep-green` and use `velvet` as the `material`. Do not modify other parts".). The corresponding SVG path is rasterized into a binary mask, and both the mask and edit instructions are used to condition an image inpainting request.[6] This enables nondestructive, segment-aware updates that preserve the global intent while precisely applying changes to the targeted part (Fig. 7).

Both pathways are associated with a user-specific VKG that acts as the semantic backbone of the system. This approach not only facilitates the traceability and reusability of design decisions but also supports cognitive continuity, as edits accumulate in the VKG rather than resetting outcomes through trial-and-error prompts. The VKG enables designers to externalize, refine, and preserve their evolving intent across an entire generative workflow (See more example and details in Appendix E.7).

[5] https://platform.openai.com/docs/models/GPT-4o.

[6] https://platform.openai.com/docs/guides/image-generation?image-generation-model=gpt-image-1.

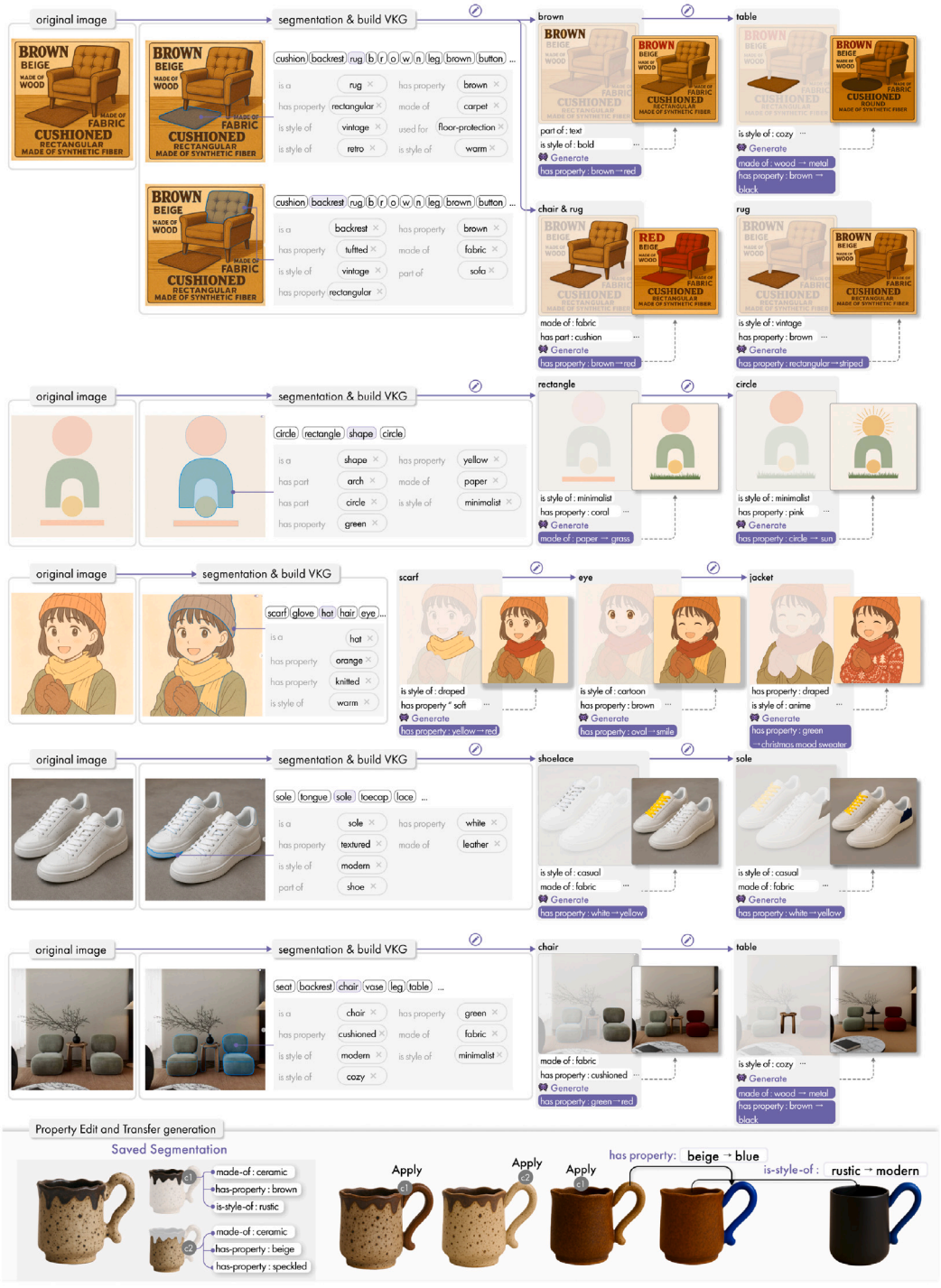


Fig. 7. VKG-guided generation, editing, and property extraction demonstrating intent-preserving part-level operations.

5.3.2. *Property-conditioned text-to-image generation*

For new design queries, the system serializes the VKG into a textual prompt by collecting all relevant triples such as `has-property`, `made-of`, and `shape`. Duplicates are removed while preserving semantic order—visual attributes first, followed by material, shape, and color. As VKG accumulates properties across interactions, injecting all properties would exceed token limits and introduce noise, while injecting none would lose accumulated knowledge. To address token limits and noise from injecting all properties, *GenAlign* employs Hierarchical Bayesian Inference (HBI) to selectively inject properties based on style-level priors. Each `is-style-of` node maintains counts over observed property–value pairs (e.g., `has-property: curved`, `made-of: brushed-metal` under `modern`). When generating with a style token, HBI computes a smoothed posterior using Dirichlet-multinomial formulation:

$$E[\theta_s(p,o)] \propto c_s(p,o) + \alpha \tag{1}$$

where $c_s(p,o)$ is the count of pair (p,o) under style s, and $\alpha = 0.1$. The system selects top-k triples, merges them into prompt construction or inpainting instructions (full formulation in Appendix E.7). If multiple styles are referenced, posteriors are averaged; if none match, the system falls back to a global prior.

To illustrate, consider a designer who has accumulated the following VKG through prior interactions (Fig. 8). Under the style node

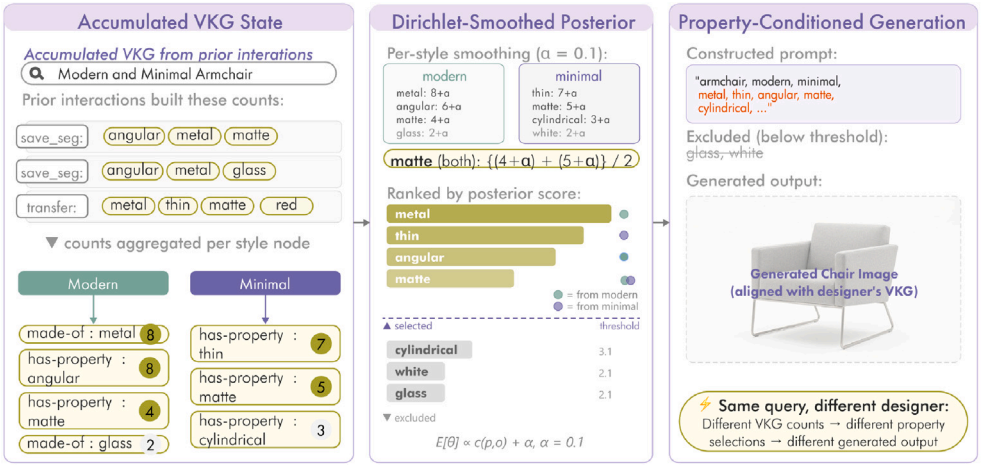


Fig. 8. HBI property selection process for a "modern, minimal armchair" query. Prior interactions accumulate property counts per style node (left); Dirichlet smoothing ranks properties and averages shared ones (center); high-ranked properties are injected into the prompt while low-ranked ones are excluded (right). The same query yields different selections for different designers.

"modern", the system has recorded properties such as `metal` (8 times), `angular` (6), `matte` (4), and `glass` (2). Under "minimal", it has recorded `thin` (7), `matte` (5), `cylindrical` (3), and `white` (2). These counts reflect the designer's accumulated interaction history: each `save_seg`, `edit_attr`, and `transfer` action increments the count for the associated style–property pair. When the designer queries "modern, minimal armchair", HBI computes a smoothed posterior for each property based on its observed frequency under the matched styles. Properties appearing under a single style use that style's count directly; properties appearing under both styles (e.g., `matte`, observed 4 times under modern and 5 times under minimal) are averaged across styles. The resulting scores are ranked, and high-ranked properties (`metal, thin, angular, matte`) are injected into the generation prompt, while low-ranked properties (`glass, white`) are excluded to prevent over-specification (Fig. 8, center and right panels). Crucially, the same query issued by a different designer with different VKG counts would yield different property selections, reflecting personalized rather than generic interpretations.

Because recommendations are backed by VKG counts, designers can inspect and override defaults, keeping prompts concise and token-efficient. This ensures the same attribute (e.g., "modern") reflects each designer's reinforced meanings rather than generic interpretations. Visual-attribute–conditioned priors are merged into the prompt to reflect designer-specific semantics. The prompt is then sent to gpt-image-1[7] to generate candidate images that align with evolving user preferences.

6. User study

To understand how *GenAlign* could help designers to resolve ambiguous design tasks, we conducted a within-subject study with 24 professional designers. We aim to answer the following research questions:

- **RQ1**: Does VKG-based editing improve designers' control and perceived intent alignment with generative outputs?
- **RQ2**: How does externalizing segment–property bindings in a VKG affect designers' exploration and refinement processes?
- **RQ3**: What are designers' experiences and strategies when using structured semantic representations versus unstructured prompt histories?

[7] https://platform.openai.com/docs/guides/image-generation?image-generation-model=gpt-image-1.

6.1. Participants and procedure

We recruited 24 professional designers from diverse design domains (Table 3, years of experience: M = 5.375, range = 3–12 years). To be eligible, designers were required to have (i) hands-on experience in designing or specifying a furniture/product; (ii) prior experience using GenAI tools such as ChatGPT, Gemini, or Midjourney as part of their practice; and (iii) at least two years of ongoing professional experience working on design projects in a relevant domain. An a priori power analysis for a within-subject contrast (two-tailed paired t-test, α = .05) targeting a medium–large effect ($|d|$ = 0.7) using G*Power 3.1 indicated that this sample size would be sufficient [67]. Participants reported GenAI usage across design stages on a 7-point scale (1 = not at all, 7 = very often): overall use (M = 4.25, SD = 1.75), idea generation (M = 4.63, SD = 1.55), concrete development—attribute exploration and layout (M = 3.33, SD = 1.63), and final production (M = 2.17, SD = 1.23). The study protocol was approved by our institutional ethics board.

Each participant used both *GenAlign* and a baseline system in a counterbalanced within-subjects design (AB/BA). The baseline represents the image-exploration workflow designers commonly use in practice (e.g., Pinterest, Behance, mood-board tools combined with text-to-image generation): keyword-based search, gallery-style browsing, prompt-based generation, and image-level saving. It lacks three VKG-enabled features: (1) property editor for segment-level inspection and modification, (2) drag-and-drop transfer of properties across objects, and (3) interactive graph visualization of accumulated bindings. Both systems used identical generation (OpenAI gpt-image-1) and retrieval (CLIP ViT-B/32) backends, ensuring VKG was the sole systematic difference. This isolates VKG's contribution while maintaining ecological validity of prompt-based workflows (Details in Appendix D.1). All sessions were conducted in an offline laboratory setting and screen-recorded with audio. Sessions lasted $\tilde{2}$ h.

Introduction (10 min). Participants signed informed consent forms and received an overview of both systems. They were informed that the study involved generating design variations using abstract visual attributes.

Tutorials (10 min per system). Participants explored each system through guided walkthroughs using a sample task (*"modern minimal lamp"*) to become familiar with the available features.

Design Tasks (30 min per system). Participants designed furniture under two attribute-pair conditions: (1) *conflicting adjectives*, defined as the intersection minimal ∩ playful [68], and (2) *semantically similar adjectives*, defined as modern ∩ sleek [69]. For each attribute pair,

Table 3
Participant demographics (N = 24). Mean professional experience was 5.375 years (SD = 2.57).

P	Age	Gender	Domain	Exp.	P	Age	Gender	Domain	Exp.
P1	25	M	Interior/Architecture Design	4	P13	42	M	Space and Architecture Design	12
P2	28	M	Interior/Architecture Design	4	P14	26	F	Automotive interior Design	4
P3	42	F	Interior/Architecture Design	12	P15	33	M	Retail Design	8
P4	26	F	Product/graphic design	4	P16	34	M	Retail Design	9
P5	27	F	Interior/Architecture Design	3	P17	25	F	Interior/Architecture Design	4
P6	26	F	Interior/Architecture Design	3	P18	26	F	Architecture Design	4
P7	27	M	Interior and residential Design	7	P19	27	F	Architectural Designer	6
P8	26	F	Exhibition and BX	4	P20	30	F	Architecture Design	5
P9	24	F	Interior/Architecture Design	4	P21	27	F	Interior/Architecture Design	4
P10	28	M	Interior/Architecture Design	4	P22	28	F	Interior/Architecture Design	5
P11	29	M	Interior/Architecture Design	7	P23	26	F	Product Design	4
P12	27	M	Interior/Architecture Design	3	P24	29	F	Architecture Design	5

Table 4
Model stack used in GenAlign across pipeline stages, combining Rembg and Semantic-SAM for preprocessing, GPT-4o for language tasks, gpt-image-1 for image generation, and CLIP for retrieval.

Pipeline stage	Model	Version	Role
Background removal	Rembg	v2.0.67	Foreground isolation
Semantic segmentation	Semantic-SAM	SwinL backbone	Part decomposition
Segment labeling	GPT-4o	2024-08-06	Kebab-case label assignment
Style extraction	GPT-4o	2024-08-06	1–3 style adjectives
Triple extraction	GPT-4o	2024-08-06	VisualSem triples
Text-to-image generation	gpt-image-1	2025 release	Property-conditioned generation
Segment inpainting	gpt-image-1	2025 release	Mask + instruction local edit
Retrieval	CLIP ViT-B/32	OpenAI 2021	Image–text matching

participants explored multiple design variations, selected a final design, and justified their choices using their own design language.

Post-Task Questionnaire (5 min per system). After completing each system condition, participants rated the system on multiple dimensions, including cognitive load (NASA-TLX) [70], creativity support (CSI) [71], user satisfaction [59], intent expression, output quality [72], and cognitive support. All subjective measures were collected using 7-point Likert scales.

Interview (20 min). A semi-structured interview probed how *GenAlign*'s core features, including the VKG, property editor, and graph visualization, influenced participants' design processes, clarified visual attributes, and supported articulation of design intent (Appendix D.2).

6.2. Implementation details

Architecture. The system used in the study was developed using Python 3.9 and operated in a high-performance client–server architecture (client: React running on Windows OS with an Intel Core i9-10980XE CPU and 64 GB RAM; server: Python Flask on Linux OS with an NVIDIA RTX 4090 GPU). VKG stored in-memory as NetworkX graph for O(1) node/edge lookups, serialized to JSON for persistence.

Models and Latency. Semantic-SAM [66] (Swin-L backbone, 0.8 s/image on RTX 4090) | GPT-4o[8] via OpenAI API (batch requests, 400 ms/segment for labeling and property extraction) | gpt-image-1[9] for generation and edit for inpainting with mask + instruction. Table 4 summarizes the models, versions, and roles in our pipeline.

Dataset. Furniture and interior images curated from public Kaggle datasets (Multidomain Image Characteristics,[10] Furniture-101,[11] Furniture Detector[12] with CLIP embeddings precomputed for retrieval.

[8] https://platform.openai.com/docs/models/GPT-4o.
[9] https://platform.openai.com/docs/models/gpt-image-1.
[10] https://www.kaggle.com/datasets/realtimear/multidomain-image-characteristics-dataset.
[11] https://www.kaggle.com/datasets/kerrit/furniture-101.
[12] https://www.kaggle.com/datasets/akkithetechie/furniture-detector.

6.3. Measures

We focus on process-level properties of designer–system interaction — control, traceability, semantic stability, and personalization — rather than artifact-level image quality. We capture these through three data sources: behavioral interaction logs (objective), VKG-evolution metrics (objective), and participant self-report (subjective).

6.3.1. System usability and creative-support metrics

After each task, participants completed a post-task questionnaire that assessed satisfaction with the generated designs and exploration process [59], CSI [71], and NASA–TLX [70]. All items used a 7-point Likert scale, except CSI, which used a 10-point scale. We first tested normality using the Shapiro–Wilk test to compare the two systems in our within-subject design. When both condition-wise distributions met normality, we applied a paired-samples t-test; otherwise, we used a Wilcoxon signed-rank test. We also conducted in-depth interviews to probe perceived differences in the design process with and without *GenAlign*'s key capabilities. Finally, we analyzed interaction behavior logs to characterize how the design process unfolded under each condition. For each system, we measured: (i) the number of images saved, (ii) the total number of generations (prompt-based only in *Baseline* and prompt-based plus property-editor–initiated in *GenAlign*), and (iii) the proportion of saved images that were generated within the system.

6.3.2. Knowledge-graph performance metrics

We further analyzed the evolution of each designer-specific VKG over time. For GenAlign, VKG snapshots were captured incrementally through save, edit, and transfer actions during interaction. For Baseline, which lacks these interactive operations, we applied the same VKG construction pipeline (Section 5.2) post hoc to each saved or generated image, treating each as an equivalent snapshot event. This ensures that `triple_count`, `reuse_rate`, `avg_semantic_drift`, and `graph_change_rate` are computed through the same extraction procedure in both conditions (full detail in Appendix A).

We computed two variants of the VKG evolution metrics: a *full-history* view based on all VKG snapshots created during exploration

Table 5
Summary of VKG evolution metrics.

Metric	What it captures	Direction
`triple_count`	How much structured knowledge the designer accumulated	↑ = richer
`property_growth_rate`	Whether designers discovered new properties or exhausted keywords early	↑ = expanding
`convergence_point`	How quickly the designer achieved semantic clarity	↓ = faster
`reuse_rate`	Whether designers built on stable meanings or introduced new ones each time	↑ = more stable
`avg_semantic_drift`	How much the semantic structure shifted per step	↓ = less drift
`graph_change_rate`	How frequently property bindings were rewired	↓ = less rewiring

(`per design step`) and a *save-only* view based only on snapshots created when participants explicitly save images (`per save step`). The former reflects how VKG changes across all explored states, whereas the latter focuses on how the graph evolves for the design states that the participants chose to maintain (See detailed metric definitions/equations in Appendix A). We measured VKG evolution across four key dimensions. To aid interpretation, Table 5 provides a quick-reference summary of all metrics; formal definitions appear in Appendix A. For all per-step metrics, values were first averaged within each participant's snapshot sequence; condition-level comparisons were then performed across N = 24 participants using paired tests.

1. ***Knowledge graph scale***: `triple_count` counts the total segment–property bindings in the final VKG snapshot—intuitively, how many concrete design decisions the designer made explicit.
2. ***Knowledge expandability***: `property_growth_rate` measures whether the designer's vocabulary grew over the session (positive) or plateaued early (negative). It is computed as the relative change in distinct predicates from the first non-empty to the final snapshot. We also report `convergence_point`, the earliest snapshot where the vocabulary stabilized—lower values mean the designer reached semantic clarity faster.
3. ***Structural consistency***: `reuse_rate` captures whether designers reused established properties or introduced new ones with each triple. A value near 1.0 means most new triples reuse existing properties rather than introducing novel ones, indicating a stable design vocabulary. Formally, for each snapshot: $1 - |P_i|/|T_i|$; we report both the final value and its session mean (`avg_reuse_rate`).
4. ***Semantic stability***: `avg_semantic_drift` measures how much the VKG changed between consecutive steps—lower values mean the designer made smaller, more coherent updates rather than large reinterpretations. It is computed as the mean Jaccard distance over nodes and edges between consecutive snapshots. `graph_change_rate` captures how frequently bindings were rewired, measured as the mean symmetric difference of edge sets between consecutive snapshots. We report both metrics for all design steps and for saved-only steps.
5. ***Embedding-Based Semantic Drift***: A potential concern with Jaccard-based drift is that it treats all property changes as equally significant—"red" → "burgundy" counts the same as "red" → "metal". To validate robustness, we computed an embedding-based drift measure mapping all property labels into CLIP text embedding space (ViT-B/32). For each pair of consecutive snapshots, we calculated a soft drift score:

$$\text{Drift}_{\text{emb}}(S_{i-1}, S_i) = 1 - \frac{1}{|S_i|} \sum_{p \in S_i} \max_{q \in S_{i-1}} \cos(\mathbf{e}_p, \mathbf{e}_q) \tag{2}$$

where $\mathbf{e}_p$ denotes the CLIP text embedding of property label p. This continuous measure captures graded semantic proximity—"red" → "burgundy" registers as a smaller change than "red" → "blue"—and serves as a robustness check for the discrete Jaccard metric (full per-participant results in Appendix B).

7. Results & findings

We evaluate how effectively VKG-based segment–property decomposition addresses semantic ambiguity through three research questions. **RQ1** examines whether VKG-based editing improves designers' control and perceived intent alignment with generative outputs by making ambiguous attributes explicit. **RQ2** investigates how externalizing these bindings affects designers' exploration and refinement processes by providing stable semantic anchors. **RQ3** explores designers' experiences and strategies when using structured semantic representations that disambiguate meaning versus unstructured prompt histories that leave meaning implicit. The survey and comparative evaluation results are presented in Table 6 and Table 7, respectively.

7.1. Control and alignment (RQ1)

In baseline text-to-image workflows, abstract adjectives like "modern" or "minimal" force models to simultaneously interpret multiple conflicting semantic dimensions, often yielding outputs that miss designers' intent. We hypothesized that decomposing these adjectives into explicit segment–property bindings would enable fine-grained control. Our findings show that semantic disambiguation transforms abstract attributes into manipulable design vocabularies, shifting interaction from guess-and-check prompting to systematic property exploration. We report this shift through four metrics: `triple_count` (how many explicit bindings accumulated), `property_growth_rate` (whether the vocabulary expanded or plateaued), `avg_semantic_drift` (how much the VKG changed per step; lower = more coherent), and `graph_change_rate` (how frequently bindings were rewired; lower = more stable). See Table 5 for a full summary.

VKG Enables Richer, Faster Semantic Externalization. *GenAlign* participants accumulated 3.74× more semantic triples (124.00 → 463.92, $p < .001$) than Baseline, demonstrating that they no longer treat abstract terms as monolithic keywords but decompose them into rich segment–property assemblies (Fig. 9-b). For instance, "modern" expanded into `leg→material→metal` + `leg→texture→brushed`, `body→geometry→angular`, `surface→finish → matte` each binding representing a concrete design decision rather than a vague descriptor. This explosion in triple count reflects what we term *semantic externalization*: designers' implicit aesthetic judgments became explicit through structured bindings, enabling computational interpretation. More importantly, property growth rate shifted from negative to positive (−0.64 → +0.18, $p = .017$), and convergence point decreased 3× (3.00 → 1.08, $p < .001$). In Baseline, participants exhausted keywords early (negative growth) with no further way to refine after listing 'futuristic, sleek, modern, minimal". In *GenAlign*, participants continuously discovered new properties by inspecting segments that revealed attributes like *chamfered edges* or *light reflectance*. Semantic clarity was achieved 3× faster; participants no longer required multiple prompt rephrasing iterations because segment–property bindings made concepts computationally interpretable from the first generation. Participants' ratings confirmed this shift. Intent expression improved dramatically (Q2, 3.29 → 6.08, $p < .001$, $r = 0.81$), and exploration effectiveness increased (Q16, 3.54 → 6.29, $p < .001$, $r = 0.86$). P14 captured the contrast succinctly: "In Baseline, I kept rephrasing prompts hoping the AI would guess my intent. With *GenAlign*, I could just adjust the specific property that was wrong".

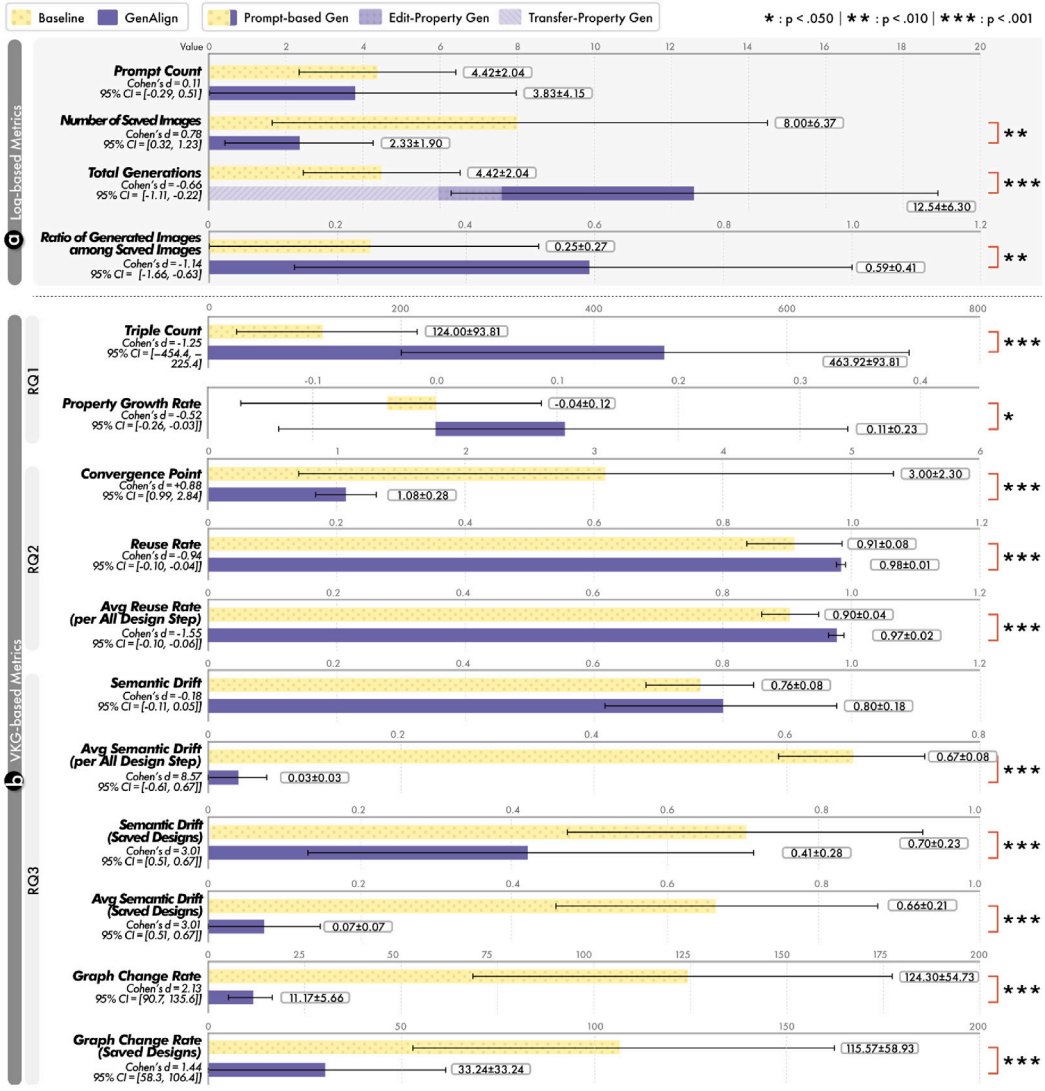


Fig. 9. Comparison of (a) log-based metrics (prompt count, saved images, ratio, and total generations); (b) VKG-based metrics (triple and property dynamics, reuse, semantic drift, and graph change rate) between Baseline and *GenAlign* conditions.

Explicit Semantic Bindings Reduce Per-Step Drift. *GenAlign* reduced avg_semantic_drift (per design step) by 22× (0.67 → 0.03, $t(23) = 42.01$, $p < .001$, $d = 8.57$), indicating that consecutive VKG snapshots changed in smaller, more coherent increments rather than through large reinterpretations at each step (Fig. 9-b). For explicitly saved segments, drift decreased by 9.4× (0.66 → 0.07, $t(23) = 14.73$, $p < .001$, $d = 3.01$). Importantly, semantic_drift (first-to-final) showed no significant difference between conditions (0.76 → 0.80, $p = .229$, *n.s.*): both conditions traversed similar overall semantic distances, ruling out the alternative that GenAlign simply explored a narrower space. What differed was the path—avg_semantic_drift (per step) decreased sharply in GenAlign because designers reached final designs through smaller, coherent updates rather than noisy reinterpretations at each step. In Baseline, each prompt risked reinterpretation; "modern" in prompt 1 could be minimalist but by prompt 5 became industrial—shifts that compounded into a semantic telephone game contradicting the initial intent. In contrast, GenAlign's property bindings externalize intent as explicit, inspectable triples: body→geometry→angular preserved its meaning across iterations because the VKG made it directly operable rather than re-inferred from language at each turn. This stability arises from active re-anchoring through designer interactions (save/edit/-transfer), as evidenced by the positive property growth rate (+0.18) and 0.98 reuse rate reported above, rather than from passive accumulation of graph nodes. Graph_change_rate decreased by 11× (124.30 → 11.17, $t(23) = 10.44$, $p < .001$, $d = 2.13$), indicating that the VKG remained stable across iterations (Fig. 9(b)). Participants were not forced to repeatedly revise properties to "correct" misinterpretations. Once `roof→material→glass` was specified, its meaning was preserved throughout the session. This stability enabled participants to build on prior decisions rather than counteracting semantic drift. Participants' ratings corroborated these findings, showing reduced frustration (Q10, 3.38 → 1.44, $p = .003$, $r = 0.78$), higher satisfaction (Q11, 3.88 → 6.42, $p < .001$, $r = 0.83$), and stronger intent matching (Q20, 3.21 → 6.31, $p < .001$, $r = 0.85$). Crucially, alignment did not reduce diversity (Q13, 4.50 → 6.44, $p < .001$). Lower drift combined with higher diversity reflects a shift from unintended variability to controlled exploration. As P9 summarized: "Baseline gave me random surprises, sometimes good, often frustrating. *GenAlign* gave me intentional variations—I knew why each output was different".

The embedding-based drift measure confirmed the Jaccard-based findings, with near-perfect correlation across all 24 participants ($r = 0.998$, $p < .001$; mean Jaccard drift = 0.130, $SD = 0.088$; mean embedding drift = 0.016, $SD = 0.011$)—that is, lexical-level (Jaccard) and semantic-level (CLIP embedding) similarity measures converged, suggesting that VKG-induced stability operates not only at the symbolic

Table 6
Survey results: Baseline vs. *GenAlign* (n = 24).

Item	Baseline	GenAlign	95% CI	*p*	Effect size
Intent Expression					
Q1. I was able to freely express my design exploration intentions in the system***	3.33 ± 1.46	6.31 ± 0.93	[2.33, 3.63]	<.001	0.85
Q2. I was able to accurately express my design exploration intentions***	3.29 ± 1.60	6.08 ± 0.93	[2.10, 3.48]	<.001	0.81
Q3. The system understood my design exploration intentions well***	3.42 ± 1.84	6.15 ± 0.93	[1.93, 3.53]	<.001	0.79
Q4. I am generally satisfied with the system used for expressing intentions***	3.83 ± 1.74	6.40 ± 0.77	[1.84, 3.30]	<.001	0.79
NASA-TLX					
Q5. Mental Demand	2.94 ± 1.84	2.25 ± 1.48	[−1.62, 0.24]	.163	0.48
Q6. Physical Demand*	2.00 ± 1.53	1.25 ± 0.53	[−1.35, −0.15]	.022	0.83
Q7. Temporal Demand	2.75 ± 1.65	2.04 ± 1.27	[−1.63, 0.21]	.163	0.48
Q8. Performance***	3.54 ± 1.67	6.00 ± 1.06	[1.72, 3.20]	<.001	0.80
Q9. Effort[a]	4.25 ± 1.59	3.65 ± 1.46	[−1.42, 0.22]	.136	0.32
Q10. Frustration**	3.38 ± 2.28	1.44 ± 0.77	[−2.94, −0.94]	.003	0.78
Satisfaction					
Q11. I am generally satisfied with the results found through the system***	3.88 ± 1.78	6.42 ± 0.65	[1.82, 3.26]	<.001	0.83
Q12. I am satisfied with the quantity of results found through the system**	4.12 ± 1.87	5.96 ± 1.16	[0.98, 2.70]	.002	0.69
Q13. I believe that the system allows me to find diverse results***	4.50 ± 1.64	6.44 ± 0.61	[1.28, 2.60]	<.001	0.81
Q14. I think the results found through the system are creative***	3.38 ± 1.84	6.29 ± 0.81	[2.13, 3.69]	<.001	0.84
Knowledge Graph Support					
Q15. The system allowed me to freely explore the design space***	3.92 ± 1.84	6.06 ± 0.81	[1.40, 2.88]	<.001	0.78
Q16. The system was effective for focused exploration based on my design goals***	3.54 ± 1.44	6.29 ± 1.04	[2.13, 3.37]	<.001	0.86
Q17. The information helped me decide which areas to explore further***	3.21 ± 1.53	6.29 ± 1.00	[2.43, 3.73]	<.001	0.87
Q18. The system helped me understand attributes and connections between options***	2.71 ± 1.52	6.46 ± 0.93	[3.09, 4.41]	<.001	0.87
Q19. I found the suggestions and information to be accurate and trustworthy***	3.29 ± 1.78	6.31 ± 0.88	[2.30, 3.74]	<.001	0.87
Q20. The suggested properties and generated image matched what I had in mind***	3.21 ± 1.44	6.31 ± 0.93	[2.48, 3.72]	<.001	0.85
Creative Support Index (CSI)					
Q21. Expressiveness***	5.29 ± 2.34	8.78 ± 1.06	[2.54, 4.44]	<.001	0.82
Q22. Exploration***	5.00 ± 2.52	8.58 ± 1.56	[2.48, 4.68]	<.001	0.76
Q23. Enjoyment***	5.56 ± 2.77	9.39 ± 0.94	[2.73, 4.93]	<.001	0.83
Q24. Results Worth Effort***	4.98 ± 2.57	9.02 ± 1.10	[3.00, 5.08]	<.001	0.82
Q25. Immersion***	4.50 ± 2.84	8.14 ± 1.68	[2.45, 4.83]	<.001	0.76

Note. Values shown as Mean ± SD. CI = 95% confidence interval for mean difference. Effect Size = Rank-biserial correlation (r_{rb}) for Wilcoxon tests, Cohen's d for t-test. Test selection based on Shapiro–Wilk normality tests ($\alpha = .05$).

* $p < .05$.

** $p < .01$.

*** $p < .001$.

(after Holm-Bonferroni correction).

[a] Paired t-test (normal distribution);

all other items used Wilcoxon signed-rank test (non-normal distribution).

label level but also in the underlying semantic space. Participants' property modifications were predominantly categorical (e.g., `metal` → `wood`, `angular` → `rounded`) rather than within-cluster refinements (e.g., "red" → "burgundy"), indicating that the drift reductions reported above are consistent across lexical (Jaccard) and embedding-based measures rather than an artifact of any single metric (equations and detailed results in Appendix B).

Case: P6's Role-Splitting Strategy (Fig. 10). P6, designing a "minimal × playful" lamp, discovered role-splitting: minimal as primitive geometry (cylinders, spheres) and playful as glossy color. Crucially, this insight emerged not from prompt experimentation but from inspecting saved segment properties. *GenAlign* made the system's interpretation visible, allowing P6 to verify and refine semantic mappings. P6 then systematically assigned roles: minimal → primitive forms ('base→ shape→ cylindrical', 'shade→ shape→ spherical'), playful → vibrant finishes ('shade→ color→ glossy-red', 'base→ color→ yellow'). P6 rapidly transferred saved segments across designs, creating new color combinations while preserving geometric minimalism. Compared to Baseline's opaque trial-and-error, *GenAlign*'s segment inspection made the AI's interpretation visible and correctable.

Case: P8's Cross-Object Property Transfer (Fig. 11). P8 defined "sleek" as slim metal legs and "modern" as chunky white-fabric body, then faced a practical problem: the prompt "metal sofa" failed to produce slim-metal examples—a common Baseline failure with abstract prompts yielding unexpected interpretations. Rather than rephrasing endlessly, P8 built a vocabulary across reference objects. P8 searched for "metal table" and "futuristic table", saved segments exhibiting desired slimness, and refined attributes by adding 'industrial', 'layered', 'thin' tags. P8 then dragged these table-leg segments directly onto the sofa design. The VKG transferred properties ('material→ metal', 'shape→ tapered', 'finish→ brushed') without linguistic reinterpretation. This cross-object workflow — impossible in Baseline's unstructured prompting — became routine with VKG's explicit bindings. P8's case demonstrates how VKG enables systematic vocabulary building: properties accumulate across explorations and transfer precisely, rather than hoping each new prompt captures the right interpretation. In summary, by decomposing abstract attributes into explicit bindings, VKG transforms control, enabling designers to systematically compose outcomes rather than repeatedly negotiate intent through prompts.

7.2. Exploration and refinement (RQ2)

Even when participants successfully expressed intent (RQ1), maintaining intent across iterations remained challenging—especially when recombining elements from different references. Our findings show that VKG transforms exploration from example accumulation to intentional vocabulary building, with participants constructing reusable, personalized property collections. We characterize this shift primarily through `reuse_rate` (the proportion of triples reusing existing properties rather than introducing new ones; higher = more stable vocabulary) alongside behavioral log metrics (saved images, total generations, generation-to-save ratio).

Semantic Stability Enables Property Reuse. The most striking evidence is the 0.98 reuse rate (Fig. 9-b)—both in final snapshots ($p < .001$) and averaged across all design steps ($p < .001$). This near-perfect reuse indicates that participants treated segment–property bindings

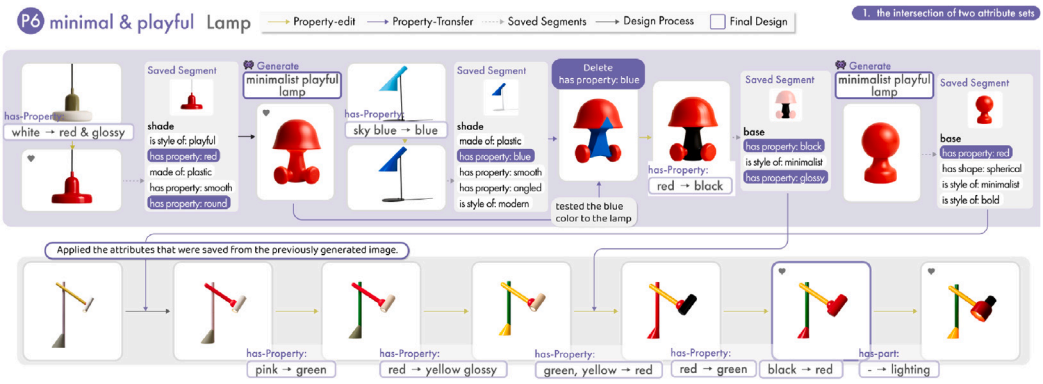


Fig. 10. Case (P6): Segment-based color–material recombination in a "minimal & playful" lamp. The designer decomposed the lamp into shade and base segments, edited color and material attributes (e.g., red, glossy, blue), and saved intermediate segments across iterations. By recombining these stored segments — along with their inherited properties — the designer refined the palette and consistently reapplied preferred attributes, ultimately producing a coherent minimalist playful lamp.

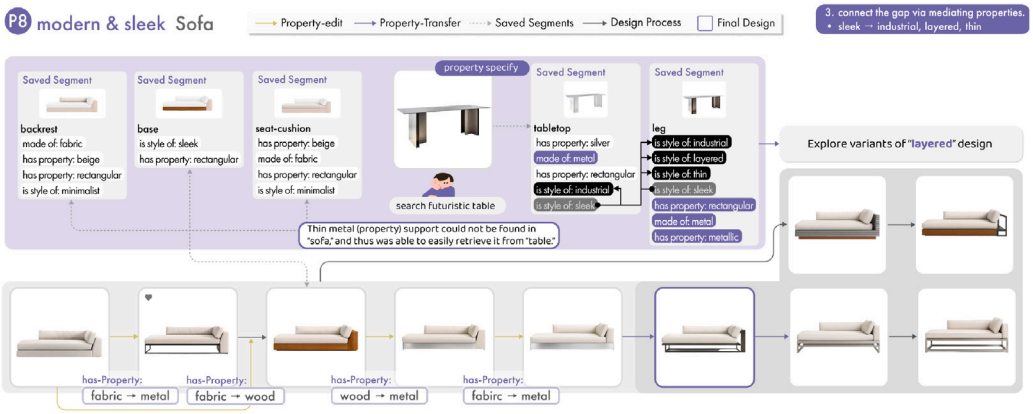


Fig. 11. Case (P8): Creative cross-object property transfer using segment-level mediation. The designer identified meaningful properties from a table (e.g., metal legs, layered structure) and saved them as transferable segments. These saved leg and tabletop segments — carrying attributes such as sleek, industrial, metallic, and layered — were repeatedly applied to a sofa, using intermediate properties to align shape and material. By leveraging cross-object segment reuse, the designer generated multiple coherent variants of a modern & sleek sofa.

as stable semantic units whose meanings persisted across operations. For instance, top→ material→ glass consistently meant "transparent glass top" throughout a session, not drifting to "glossy plastic" or "frosted glass" as in Baseline. This stability manifested behaviorally: saved images decreased 3.43× (8.00 → 2.33, p = .001) while total generations increased 2.84× (4.42 → 12.54, p < .001) (Fig. 9-a). This inversion reveals a fundamental shift. In Baseline, participants saved many outputs hoping to "show the AI" their preferences—accumulating examples like a mood board, hoping the model would infer patterns. Without explicit semantic linkage, participants could not trace which aspects of saved images would influence future outputs. In *GenAlign*, participants saved far fewer images because each save was a semantic anchor rather than an example. A single saved segment carries explicit properties (leg→ material→ wood + geometry→ tapered + finish→ matte), making further saves redundant until new concepts emerge. The generation-to-save ratio increased 2.35× (0.25 → 0.59, p = .009), participants generated more variations per saved segment because VKG maintained alignment without constant regeneration.

Participants' ratings confirmed improved traceability (Q18, 2.71 → 6.46, p < .001, r = 0.87; Q19, 3.29 → 6.31, p < .001, r = 0.87). Satisfaction also increased: outputs were rated as more creative (Q14, 3.38 → 6.29, p < .001) and more diverse (Q13, 4.50 → 6.44, p < .001), evidence that grounding intent in properties clarifies exploration rather than constraining it.

From Keyword Lists to Personal Design Languages. Multiple participants emphasized that abstract terms like warm, minimal, sleek conveyed idiosyncratic meanings. *GenAlign* addresses this through `Save_seg` and `Add/Edit_attr`, allowing designers to associate personal semantics with attributes. The VKG became designer-specific, mapping the same term to different properties across individuals.

P3 and P9 both designed "minimal × playful" sofas but arrived at radically different outcomes (Fig. 12). In Baseline, P3 enumerated 15–20 keywords and still got off-target results. In *GenAlign*, P3 inspected per-segment properties (leg material, surface finish, cushion presence) as design guidelines. Because properties were directly editable, P3 expressed fine-grained distinctions (matte vs. glossy, fabric vs. leather), regenerated from segments (`Gen_modify`), and ensured rapid convergence P3's strategy shows that playful stayed within disciplined geometry/materials (thin, metal, angular, matte). P3 defined "minimal legs" as thin + metal + angular, a binding reliably reused across all generations. P3 reported this direct editability reduced cognitive load, allowing focus on design decisions rather than keyword tracking. P9's strategy diverged entirely. P9 first anchored a minimal sofa, then incrementally composed the conflict: changed body to deep red, legs to silver metal. Searching for more playful forms, P9 anchored a rounded sofa, transferred saved attributes (red, leather, rectangular), rebalanced the palette when deep tones undermined playful (backrest → mustard, seat → orange), edited toward spherical massing, and retained silver legs. P9's playful emerged through chroma and rounded forms (mustard/orange palette, spherical geometry, silver-metal legs).

Same task, different VKG → different outcomes. This personalization demonstrates that *GenAlign* captures designer-specific semantics rather than universal interpretations. P9 explicitly saved properties "for

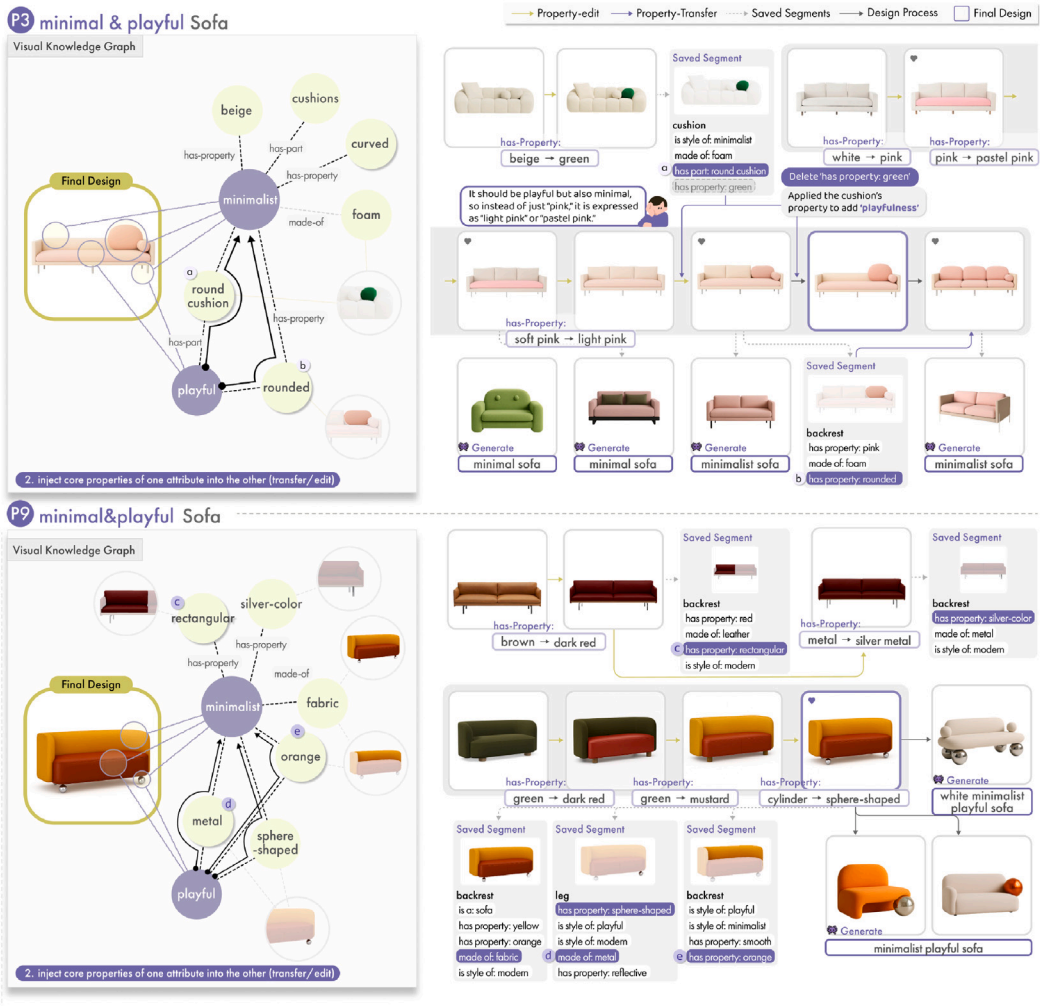


Fig. 12. Personalized "minimal × playful" sofa results based on each designer's VKG. Although P3 and P9 perform the same task, their distinct segment–attribute graphs (colors, materials, shapes) guide different edit paths and lead to different final sofas. Each generation updates and refines the designer-specific VKG, showing how personalized semantics produce diverging yet coherent outcomes.

client explanation", highlighting how VKG improves not just control but design justifiability—participants can articulate exactly which decisions produce final outcomes. Also, P10 summarized the contrast: "*In Baseline, I was dissatisfied owing to difficulty controlling words. In GenAlign, I could incrementally expand and apply properties to establish finer variations and clearer directions*". This incremental expansion reflects property_growth_rate (RQ1) and reuse_rate (RQ2) working in concert, enabling continuous vocabulary building with stable semantic anchors.

7.3. Experiences and strategies (RQ3)

Beyond RQ1 and RQ2, we investigated how designers actually worked with VKG versus unstructured prompts. Our analysis integrates behavioral logs, cognitive graphs, and interviews to reveal: (1) what design strategies emerge when intent is externalized as manipulable structure, (2) how designers experience the shift from linguistic negotiation to property-based composition, and (3) what workflows enable effective attribute synthesis (See all behavioral logs in Appendix D.3).

Prompts Become Anchors, Not Trials. Behavioral logs reveal a fundamental shift in how designers use prompts. While prompt count remained similar (4.42 vs. 3.83, $p = .112$, n.s.), total generations increased 2.84 × (4.42 → 12.54, $p < .001$) (Fig. 9-a). This pattern, similar prompts but far more generations, indicates that participants used prompts not for trial-and-error disambiguation but as initial anchors. Once abstract intent was decomposed into properties through segment inspection ('`Save_seg`' → '`Add/Edit_attr`'), participants systematically generated variations by manipulating specific attributes ('Gen_modify', 'Gen_transfer') without re-prompting. Log interaction coding confirmed a consistent workflow: explore ('`Search`'/'`Gen_query`' to gather references) → anchor ('`Save_seg`' to decompose segments) → iterate ('`Add/Edit_attr`' → '`Gen_modify`'/'`Gen_transfer`' to refine) → consolidate ('`Cognitive_Map`' to review bindings) This represents a qualitative shift from linguistic negotiation to property-based composition. Designers no longer asked "*what words will make the AI understand*?" but "*which properties should I transfer or modify?*" Three Design Strategies for Attribute Synthesis. Analyzing cognitive graphs and interviews, we traced how designers synthesized conflicting attributes (minimal × playful, modern × sleek) through three distinct strategies (Fig. 13):

Strategy 1: Intersection-First (P4, 6, 7, 12–14, 16, 18–20, 22) Designers identified and fixed overlapping properties between attributes. P14 continuously referenced the modern × sleek intersection (rectangular, black, metal) on the cognitive map, advancing designs while maintaining these core properties. P4 explained the graph "organized mental structures I unconsciously relied on", providing strong directional support. Rather than indiscriminately applying attributes, participants carefully distinguished shared vs. divergent properties and considered which segments should embody each (Fig. 14).

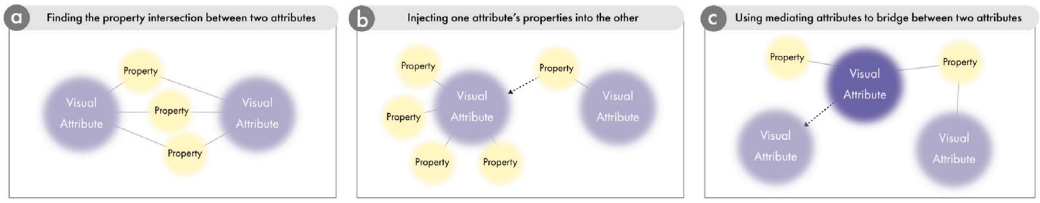


Fig. 13. Three cognitive-graph-guided strategies: (a) fix the intersection of two attribute sets, (b) inject core properties of one attribute into the other (transfer/edit), and (c) connect the gap via mediating properties.

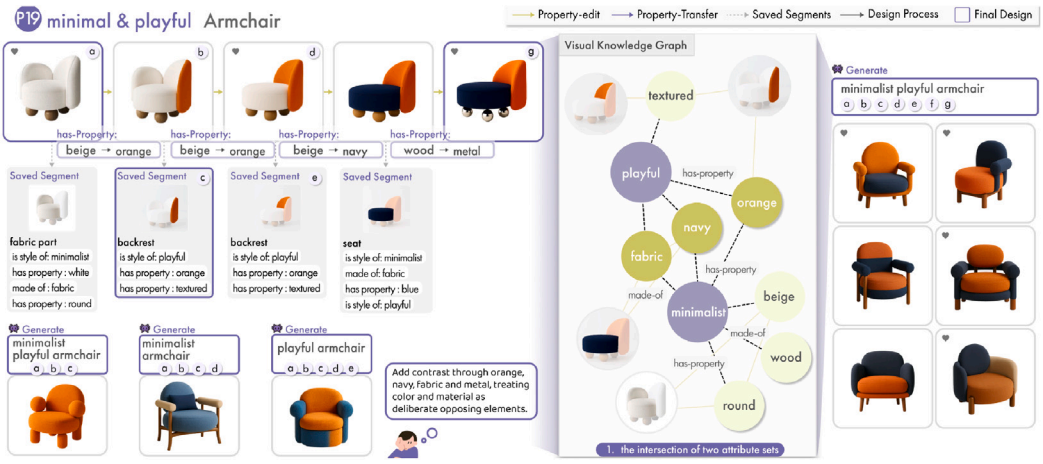


Fig. 14. Case (P19): Intent-preserving armchair generation. The designer maintains a consistent "minimal × playful" intent while repeatedly generating diverse armchair variations. Saved fabric, armrest, and seat segments — and liked images — are reused throughout the process, with the VKG guiding color, material, and shape consistency. The final set shows coherent variations that align with the designer's evolving yet stable intent.

Strategy 2: Anchor-and-Inject (P1-3, 9, 11, 15–17, 21, 23) Designers established a base design anchored in one attribute, then injected properties from the second. P1 started with a modern armchair (clean lines, neutral tones), then added sleek through orange highlights and rounded corners. P3 built a minimal sofa foundation (simple geometry, muted colors), then reinforced playful by adding cushions and bright accents, fine-tuning tones (pink → pastel pink) to balance attributes. This sequential approach allowed incremental conflict resolution rather than attempting simultaneous synthesis (Fig. 15).

Strategy 3: Mediating Properties (P5, 8, 10) Designers introduced bridging properties that connected seemingly incompatible attributes. P5 enriched minimal with playful by introducing modularity as a structural mediator—playful means reconfigurable, not decorative. This abstract insight was concretized through segment properties: joint→ property→ detachable, leg→ shape→ angular-modular. P8 used layered and thin as mediating properties to stabilize sleek, then transferred segments to converge on the intended outcome. This strategy demonstrates conceptual reframing through property decomposition (Fig. 16).

All three strategies shared a common workflow: **explore → anchor → segment-edit loop → consolidate**. However, specific paths varied with personal cognitive spaces—even identical prompts led to divergent outcomes because each participant's VKG-shaped anchor led to different property assemblies and satisfaction levels. This variability is not a limitation but evidence that the system adapts to individual design thinking rather than imposing universal workflows.

7.4. Token efficiency of HBI property selection

Beyond the behavioral and experiential effects established in RQ1–RQ3, we examine the inference layer that delivers VKG content to the generator. As VKGs accumulate during interaction, injecting all available properties exceeds practical prompt budgets; HBI addresses this through style-level priors that select a small set of properties for each query. We compare three property injection strategies — prompt only, all available properties via random sampling, and HBI's selective injection — across 113 generation actions from 24 participants. All statistical comparisons are performed at participant level (N = 24 with valid HBI-condition data) using paired tests.

7.4.1. Experimental setup

We analyzed 113 generation actions(`gen_query`, `gen_edit`, `gen_transfer`) from 24 participants in our user study, regenerating images under three property injection strategies using identical prompts to ensure fair comparison: **Prompt Only**: User's original text prompt with zero VKG properties injected, representing baseline generation without knowledge augmentation. **VKG All**: User's prompt augmented with all available VKG properties at generation time. Due to token-budget constraint of gpt-image-1's prompt API, 96.5% of actions (109/113) required random sampling to 100 properties when available property count exceeded this threshold (mean available: 1056; max: 3136). Only 3.5% (4/113) of actions with ≤100 properties could use truly all properties without sampling, meaning VKG All fundamentally represents random sampling under token constraints rather than exhaustive property inclusion. **VKG+HBI**: User's prompt augmented with properties selected by our Hierarchical Bayesian Inference algorithm. Given the same property pool (mean: 1056; range: 0–3136), HBI selects properties using style-level priors with top-k frequency selection to balance specificity and diversity (See more details in Appendix C).

All images were generated using gpt-image-1 with identical parameters (1024 × 1024, standard quality). We measured metrics across four dimensions.

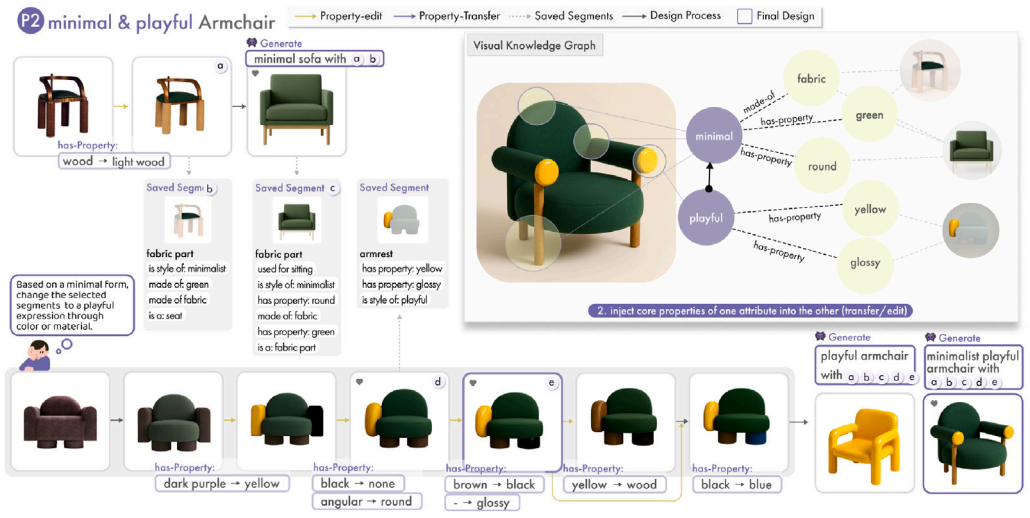


Fig. 15. Case (P2): Intent-preserving armchair edits. Starting from a minimal base chair, the designer incrementally edits and saves segment properties (fabric parts, armrest color, finish) while the VKG captures "minimal & playful" attributes such as green, round, yellow, and glossy. The final armchair converges toward the user's intended minimal yet playful style, consistent with the accumulated VKG.

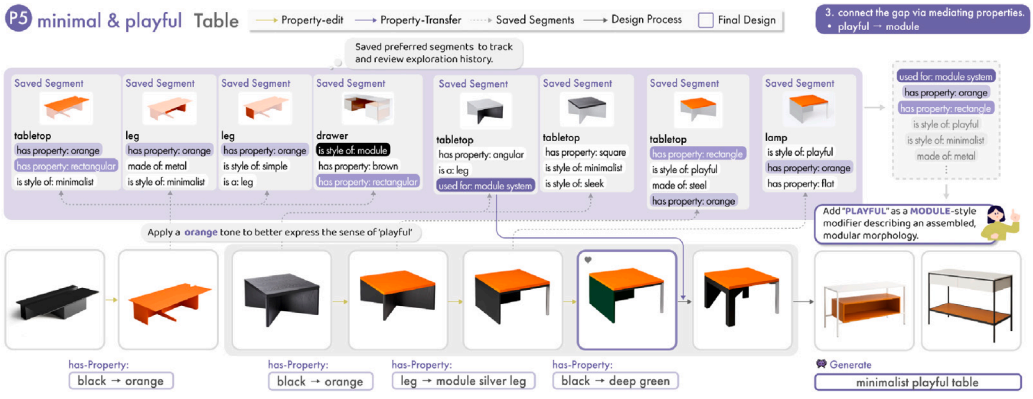


Fig. 16. Case (P5): Mediating properties through modular design. The designer decomposed "minimal & playful" into segment-level representations (tabletop, legs, drawer) and reused saved segments throughout exploration. By introducing modularity as a structural mediator — where "playful" is concretized as reconfigurable assemblage — the designer iteratively refined segment properties (color, material, style) and transferred them across generations, converging on a coherent minimal-playful table grounded in modular morphology.

1. Image–Text Alignment: CLIP similarity (ViT-B/32) between generated images and original user prompts, measuring how well outputs match textual intent.
2. Knowledge Utilization: Direct count of properties injected during generation—Prompt Only: always 0; VKG All: mean 96.7 (median 100, capped by token limits); VKG+HBI: mean 13.4 (median 14, range 0–15 via top-k selection).
3. Prompt Engineering: Expansion ratio based on token count, plus three sub-metrics for VKG conditions—Jaccard similarity between prompt and property keywords, overlap rate (percentage of prompt keywords found in properties), and expansion rate (percentage of novel properties not in original prompt).
4. Stability: Repetition consistency measured via CLIP similarity between pairs of regenerated images using identical prompts, indicating output predictability.

7.4.2. *Key findings and discussion*

At participant level ($N = 24$), all three conditions yielded comparable CLIP similarity (0.316/0.315/0.318; $\chi^2(2) = 3.74$, $p = .154$; pairwise all ns after Holm–Bonferroni correction). HBI achieves this parity with 7.2× fewer injected properties (13.4 vs. 96.7, 85.9% token savings); 98.2% of injected properties were novel relative to the user prompt (vs. 59.5% under random sampling).

This difference reflects how each strategy selects properties relative to the user prompt. For a prompt like "modern sofa", random sampling often re-injects properties that overlap with terms already articulated by the user (e.g., further "modern"-related descriptors), resulting in about 40% redundancy. HBI's style-level priors, by contrast, prioritize properties that extend the user's articulation rather than restate it.

HBI scales without over-specification across the tested property range ($r = -0.058$, $p = .79$). Repetition consistency was numerically higher under HBI (0.915 vs. 0.900, +1.5% vs. Prompt Only), and property selection adapted to knowledge availability. Participants' reports that generation "felt more predictable" and "respected saved properties" reflect the controllability and traceability of the VKG itself—segment–property bindings, reuse rate (0.98), and reduced per-step drift (22×)—delivered to the generator through HBI's property selection.

Table 7
Three-way comparison results across evaluation metrics (N = 24 participants, 113 actions total). VKG+HBI demonstrates efficiency advantages through selective property injection, achieving similar CLIP similarity with 7.2 × fewer properties than VKG All. Property pool vs. injected counts shown where applicable.

Metric	Prompt only	VKG All	VKG+HBI	Condition with advantage
1. CLIP similarity (Image–Text alignment)				
Mean ± SD	0.316 ± 0.017	0.315 ± 0.016	0.318 ± 0.017	Tie
95% CI	[0.309, 0.323]	[0.309, 0.322]	[0.311, 0.325]	
Friedman: $\chi^2(2) = 3.74$, $p = 0.154$ (ns); Post-hoc: all ns *Wilcoxon: Prompt Only vs VKG All: $W = 110.5$, $r = 0.20$, $p = 0.40$* (Holm-corrected) *Prompt Only vs VKG+HBI: $W = 99.0$, $r = -0.28$, $p = 0.47$* *VKG All vs VKG+HBI: $W = 92.0$, $r = -0.33$, $p = 0.51$*				
2. Property count utilized in image generation				
Available (pool)	0	1056 ± 628	1056 ± 628	–
Injected (actual)	0	96.7 ± 11.2†	13.4 ± 2.9‡	**VKG All**
†*96.5% (109/113) capped at 100 due to token limits (random sampling)* ‡*HBI top-k selection (median: 14, range: 0–15)*				
3. Prompt expansion (based on available pool)				
Expansion Ratio	1.0×	59.5×‡	98.2×	**VKG+HBI**
Jaccard Sim.	N/A	0.032 ± 0.032	0.030 ± 0.031	**VKG+HBI**
Overlap Rate	N/A	40.5% ± 23.0%	39.7% ± 24.0%	Tie
Expansion Rate	N/A	59.5%	98.2%	**VKG+HBI**
‡*Token-based expansion; only 100 properties injected on average*				
4. Repetition consistency (Stability)				
Mean ± SD	0.900 ± 0.015	0.908 ± 0.018	0.915 ± 0.012	**VKG+HBI**
Sample Size	(n = 8 pairs)	(n = 25 pairs)	(n = 12 pairs)	(+1.5% vs PO)
5. Property count sensitivity				
Pearson r (N = 24)	N/A	r = 0.060, p = 0.79 (ns)	r = −0.058, p = 0.79 (ns)	Tie

8. Discussion: Semantic disambiguation as continuous interpretive process

Our findings suggest disambiguation is better understood as ongoing interpretive work [5,6] distributed across multiple interactions rather than solved in a single prompt.

8.1. Ambiguity as resource, not defect

Following Gaver et al.'s [4] framing, semantic ambiguity is productive early in design—"modern" remains intentionally vague to enable divergent exploration before converging on specific interpretations. *GenAlign* does not eliminate this ambiguity but provides mechanisms to progressively stabilize meanings through interaction. The shift from negative to positive property growth rate (RQ1) exemplifies this progression. In baseline workflows, participants exhausted keywords early because each iteration forced full re-specification—once "minimal" and "modern" are listed, designers have no structured way to elaborate further without introducing semantic contamination. In *GenAlign*, participants continuously discovered properties by inspecting what the system externalized (e.g.,segment labels, material tags, shape descriptors), then refined and reused those bindings. The VKG captures this semantic accumulation, each interaction incrementally disambiguates without prematurely collapsing meanings into fixed interpretations. This aligns with interpretive perspectives that emphasize meaning-making as situated practice [5,56]. VKG makes this negotiation explicit and traceable rather than leaving it implicit in prompt histories. Critically, this externalization transforms abstract terms from subjective impressions into inspectable, manipulable structures that both designers and AI can reference, creating what Scaife and Rogers [64] term computational offloading—the cognitive work of disambiguation shifts from internal mental tracking to external structural refinement.

8.2. Personalization through property binding

The divergent outcomes for identical prompts (RQ2) demonstrate a fundamental principle: disambiguation is designer-specific, not universal. This aligns with Pan et al.'s [6] professional vision theory—visual meanings are learned through situated practice rather than defined by universal rules. *GenAlign*'s property-level personalization offers three advantages over output-level approaches. First, *finer granularity*: rather than learning "this designer prefers modern styles" (output-level), the system learns "`leg—material→metal` + `leg—shape→thin` constitutes modern for this designer" (property-level). Second, *interpretability*: designers can inspect why the system interprets "modern" as thin metal legs by querying the VKG's `is-style-of` edges, then override defaults when inappropriate. Third, *transferability*: property bindings carry semantic meaning across objects—a "modern table" vocabulary transfers directly to "modern shelving" without linguistic reinterpretation, whereas output-level style transfer risks copying irrelevant features. The 0.98 reuse rate indicates these personal mappings stabilize within sessions, enabling what we term *cognitive continuity* [57]. Once a designer establishes property bindings through repeated interactions, those bindings become reliable semantic anchors, the system no longer guesses what "modern" means but retrieves the designer's established interpretation. This differs fundamentally from example-based personalization [13,24], which accumulates outputs hoping the model infers patterns but cannot guarantee which features transfer. Participants nevertheless noted moments where the system's interpretation fell short: P15 wanted finer-grained micro-adjustments (e.g., fillet radius, material finish) that property-level editing could not fully capture, and P13 observed that background and environmental context — absent from the VKG — still shifted perceived style even when part-level properties were fixed. These residual mismatches reveal a gap between standardized property extraction and the fuller design context, which property-level personalization narrows without closing. Future work could extend this to cross-session persistence, treating VKG as an evolving semantic profile [73] where systems maintain designer-specific interpretations across projects, collaborators, and temporal gaps.

8.3. VKG as operational boundary object

Star's [18] boundary objects theory describes artifacts that coordinate work across communities without requiring consensus—they are "plastic enough to adapt to local needs yet robust enough to

maintain common identity". VKG exhibits this plasticity-robustness balance through its three-layer structure: abstract style nodes (*modern*, *minimal*) provide shared vocabulary enabling communication, concrete property nodes (*metal*, *thin*, *angular*) enable precise specification, and segment nodes (*leg*, *seat*, *backrest*) ground properties in manipulable parts. This layering allows VKG to simultaneously capture personal semantics (each designer's unique style-property mappings) and maintain computational consistency (properties remain well-defined for generation). Critically, VKG is operational rather than merely communicative. Most boundary objects in design practice — sketches [54], mood boards [52], diagrams — support interpretation and negotiation but do not execute decisions [19]. To sharpen this distinction: traditional boundary objects in design practice are primarily interpretive and retrospective. A mood board curates aesthetic references that anchor team conversation, but its semantic content — why a particular texture was chosen, which parts it should apply to, how it maps to fabrication parameters — remains implicit in the designer's head [52,53]. A sketch captures spatial intent open to reinterpretation, yet translating its meaning into generative prompts requires manual articulation that risks semantic loss [56]. Material samples communicate tactile qualities but cannot specify conditional application ("use this finish on the leg, not the seat"). These artifacts coordinate understanding but leave the gap between representation and execution unbridged.

VKG closes this gap by functioning simultaneously as a representation for sensemaking and an operational substrate for generative control. As a representation, VKG externalizes how abstract terms decompose into concrete properties—designers can query "which parts are currently bound to metal?" and trace how their interpretation of "modern" evolved across iterations, supporting the reflective practice that Schön [62] considers central to design expertise. As an operational substrate, VKG directly conditions computation: when a designer edits `leg—made-of→metal`, the system uses this binding as a generation constraint without requiring linguistic re-articulation. This dual role — simultaneously informing designer cognition and driving machine behavior — extends boundary object theory [18] into human-AI co-creation, where shared artifacts must not only coordinate understanding across stakeholders but also drive computational processes that produce design outcomes.

A designer presenting a mood board must still translate aesthetic intent into fabrication instructions; the board coordinates understanding but does not directly produce outcomes. VKG both externalizes intent for inspection (*representation*: designers query "which parts use metal?") and operationalizes it as generation constraints (*execution*: the system directly conditions image synthesis on `leg—made-of→metal` bindings). This dual role distinguishes VKG from prior visual history systems [55,59,60] that organize prompts spatially or track edit lineages but do not extract manipulable structure from outputs. The operational nature enables *semantic anchoring*: because intent is encoded as inspectable, reusable bindings rather than ephemeral language, designers can re-apply prior decisions instead of re-inferring them from language at each step (RQ3). In our study, this corresponded with lower per-step drift and high binding reuse (0.98), consistent with active re-anchoring rather than passive accumulation. Rather than eliminating drift, the VKG appears to shift exploration away from a prompt-by-prompt random walk — where each step risks losing prior progress — toward a more incremental, traceable refinement process.

9. Limitations and future work

Domain Scope and Generalizability. Our controlled user study evaluated GenAlign exclusively with furniture and interior design tasks, where part-property structure is well-defined. However, Fig. 7 demonstrates that the VKG pipeline operates across structurally diverse domains: graphic illustrations are segmented into shape primitives (circle, rectangle) with color and material properties; fashion items decompose into functional parts (scarf, hat, glove) with style attributes such as `knitted` and `draped`; and footwear segments into sole, tongue, and toecap with texture and material triples—in each case, *GenAlign* produced actionable parts and extracted domain-appropriate properties without pipeline modification. The VKG paradigm is applicable to design tasks that satisfy three structural conditions: (i) outputs admit meaningful segmentation, (ii) properties are part-scoped and transferable, and (iii) designers' interpretations vary sufficiently to benefit from personalization. Domains with highly non-compositional attributes (e.g., typographic layout, abstract branding) may require region-based rather than part-based decomposition, which remains untested. Formal cross-domain evaluation with domain-matched participants is an important direction for future work.

User Group Limitations and Future Work. Our 24 participants were professional designers with 3–12 years of experience who could articulate design intent through property vocabularies (e.g., distinguishing "matte" from "glossy", "tapered" from "cylindrical"). Non-professional users — consumers, students, or domain novices — may face higher initial cognitive load in understanding segment–property decomposition and selecting appropriate property values. The system currently assumes familiarity with design terminology; extending to broader user groups would likely require adaptive scaffolding, such as system-suggested property vocabularies, visual property palettes (showing material/color swatches rather than text labels), or template-based starting points that provide default VKG structures users can modify. Whether VKG's benefits in semantic stability and personalization extend to users without professional design training remains an open empirical question requiring novice-expert comparison studies. Future work will therefore focus on developing adaptive scaffolding mechanisms and conducting novice–expert comparative studies to systematically investigate how VKG's property-level personalization can be extended across a broader spectrum of design expertise.

Conversational, agentic assistance. Future work could integrate conversational assistance that compiles natural language goals into segment-attribute constraints, checks generations against current snapshots and prior anchors, and proposes minimal edits preserving successful parts. Because VKG records what changed, where, and why, progress remains reviewable—previously validated settings can be reinstated on demand, enabling targeted rollbacks and segment-level property reuse. When outputs drift, the agent compares them with VKG history, flags deviations, and proposes smallest corrective edits while tracing changes entering final designs.

Methodological Considerations. Because the VKG is constructed incrementally across interactions, drift metrics inherently reflect both semantic stability and structural inheritance between snapshots, requiring careful interpretation. We address this through (i) identical VKG extractive property injection across conditions, (ii) a graph-independent embedding-based drift measure, and (iii) behavioral indicators (property growth rate, reuse rate) that confirm active re-anchoring rather than passive accumulation. Still, fully disentangling these components requires drift measures independent of graph continuity, which we leave to future work.

System Boundary Conditions. Behavioral log analysis reveals three failure modes. First, *segmentation ambiguity*: when part boundaries are unclear (e.g., where a sofa seat meets its armrest), Semantic-SAM occasionally over-segments or merges distinct parts (P16: "the system sometimes split one part into two that I considered the same"). Second, *property vocabulary mismatch*: because designers use idiosyncratic terminology for visual attributes, GPT-4o's extracted labels sometimes diverged from a designer's preferred vocabulary—for instance, labeling a surface as "matte" when the designer would describe it as "brushed" or "satin". These discrepancies prompted immediate re-editing, which the system accommodated through direct property modification; however, they highlight the gap between standardized extraction and personal design language. Third, *generation deviation*: gpt-image-1 occasionally ignores part-scoped instructions despite correctly constructed

VKG-conditioned prompts—a limitation of the downstream generation model rather than the VKG pipeline itself. These failures concentrate in early interactions before designers refine their VKG; once properties are manually verified and saved, subsequent operations build on accurate foundations.

10. Conclusion

GenAlign demonstrates that semantic disambiguation through structured property bindings can fundamentally transform generative design workflows. By externalizing abstract attributes into segment-level properties and structuring them as operational knowledge graphs, the system shifts interaction from linguistic negotiation to property composition, from passive hoping to active assembly, and from retrospective mood boards to executable vocabularies. Importantly, they reveal a path toward intent-aligned generative systems that learn personal semantics through interaction rather than requiring universal prompt languages. More broadly, this work extends boundary object theory [18] into human-AI co-creation by demonstrating that boundary objects need not remain purely interpretive. By making them operational — capable of both externalizing intent for inspection and executing generation constraints — designers gain computational leverage over their evolving semantic vocabularies. The VKG paradigm offers a generalizable framework: wherever design knowledge can be decomposed into part-property structures, accumulated through interaction, and reused across contexts, operational boundary objects can transform the generative process from linguistic negotiation to structured, intent-aligned composition.

CRediT authorship contribution statement

Jiin Choi: Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Minkyung Seo:** Writing – original draft, Investigation, Data curation. **Kyung Hoon Hyun:** Writing – original draft, Supervision, Resources, Project administration, Investigation, Funding acquisition, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by the Technology Innovation Program (RS-2025-02317326, Development of AI-Driven Design Generation Technology Based on Designer Intent) funded by the Ministry of Trade, Industry & Energy (MOTIE, Korea).

Appendix A. VKG evolution metrics

To understand how semantic disambiguation unfolded in the VKG, we analyzed the evolution of each designer-specific VKG over time. For each participant and condition, we computed two variants of our VKG evolution metrics: *full-history* metrics based on all VKG snapshots created during exploration, and *save-only* metrics based only on snapshots created when participants explicitly saved or liked images. The former reflects how the VKG changed across all explored states, whereas the latter focuses on how the graph evolved for the design states that participants actually chose to keep.

Snapshot construction differed by condition in how it was triggered, not in how it was computed. In *GenAlign*, snapshots correspond to live VKG states produced by `save_seg`, `edit_attr`, and `transfer` actions. In Baseline, which provides no segment-level editing or graph-building interaction, we constructed comparable snapshots by applying the same pipeline (Section 5.2: foreground isolation, segmentation, segment labeling, and triple extraction) post hoc to each image the participant saved or generated, treating each such image as one snapshot. All downstream metrics (`triple_count`, `property_growth_rate`, `reuse_rate`, `avg_semantic_drift`, `graph_change_rate`) were then computed identically across both conditions from these snapshot sequences.

1. **Knowledge graph scale**: `triple_count`
2. **Knowledge expandability**: `property_growth_rate`, `convergence_point`
3. **Structural consistency**: `reuse_rate`, `avg_reuse_rate (per all design step)`
4. **Semantic stability**: `semantic_drift`, `avg_semantic_drift (per all design step)`, `avg_semantic_drift (saved designs)`, `graph_change_rate`, `graph_change_rate (saved designs)`

A.1. Knowledge graph scale: *triple_count*

The *triple_count* is the total number of segment–property triples in the final VKG snapshot, i.e., the size of the final edge set $|E_{\text{final}}|$. This captures how much semantic structure each designer accumulated by the end of the task.

A.2. Knowledge expandability: *property_growth_rate*

To characterize how designers' property vocabularies expanded, we compute a *property_growth_rate* for each VKG as the relative change in the number of distinct predicates (properties) from the first non-empty snapshot to the final snapshot. If P_{first} and P_{final} denote the sets of distinct predicates in the first and final snapshots, respectively, then $\texttt{property\_growth\_rate} = \left(|P_{\text{final}}| - |P_{\text{first}}|\right)/|P_{\text{first}}|$. In our implementation, we treat predicates (e.g., has-color, has-material, has-form) as coarse property categories and use their distinct count as the size of the property vocabulary. High values indicate that designers continuously introduced new properties over the session, whereas low values indicate that they quickly settled on a stable vocabulary. For interpretation, we also track a convergence point (used descriptively in the text), defined as the first snapshot at which $|P_i| = |P_{i-1}|$ and the property vocabulary plateaus.

A.3. Structural consistency: *reuse_rate and avg_reuse_rate (per all design step)*

We next examine how designers consolidate and reuse meanings within their VKGs. For a given snapshot with triple set T_i and predicate set P_i, the *reuse_rate* is defined as $\texttt{reuse\_rate}_i = 1 - |P_i|/|T_i|$, i.e., the complement of the ratio of unique predicates to total triples. This value is high when many triples reuse the same properties and low when every triple introduces a new one.

We report:

- **`reuse_rate`**: the reuse rate in the final snapshot $\texttt{reuse\_rate}_{\text{final}}$, indicating how strongly the designer's VKG has concentrated onto a compact set of properties by the end of the session;
- **`avg_reuse_rate (per all design step)`**: the reuse rate averaged across all snapshots in the full-history view, $\texttt{avg\_reuse\_rate (per all design step)} = \frac{1}{N}\sum_{i=1}^{N} \texttt{reuse\_rate}_i$, summarizing how consistently the designer relied on a stable property palette over time.

A.4. Semantic stability: Drift and graph change

Finally, we assess how stable or volatile the VKG structure is over time by comparing consecutive snapshots.

Average semantic drift (`avg_semantic_drift (per all design step)`). For the full-history view, we compute semantic drift between each pair of consecutive VKG snapshots using a Jaccard distance over both nodes and edges. If snapshot i has node set V_i and edge set E_i, the Jaccard distance between two sets A and B is $d_J(A,B) = 1 - |A \cap B|/|A \cup B|$. We define the drift between two snapshots as $\text{Drift}_i = \big(d_J(V_{i-1}, V_i) + d_J(E_{i-1}, E_i)\big)/2$, and we average this quantity over all consecutive pairs:

$$\texttt{avg\_semantic\_drift(per all design step)} = \frac{1}{N-1}\sum_{i=2}^{N} \text{Drift}_i.$$

Higher values imply more frequent shifts in which segments and properties are present; lower values imply more incremental refinement around a stable core.

Average semantic drift for saved states. (`avg_semantic_drift (per all design step)`) The *avg_semantic_drift (saved designs)* is computed in the same way, but only over the subsequence of snapshots taken when the participant explicitly saved or liked an image. This "save-only" variant focuses on how the VKG changes between design states that participants deemed worth keeping, rather than across all transient edits.

Graph rewiring. (`graph_change_rate` and `graph_change_rate (saved designs)`) While drift captures changes in which nodes and edges are present, we also measure how strongly the VKG is rewired at the edge level. For each consecutive pair of snapshots, we compute the size of the symmetric difference of the edge sets,

$|E_{i-1} \triangle E_i| = |(E_{i-1} \setminus E_i) \cup (E_i \setminus E_{i-1})|$, and average this value across the session: $\texttt{graph\_change\_rate} = \frac{1}{N-1}\sum_{i=2}^{N} |E_{i-1} \triangle E_i|$. The `graph_change_rate (saved designs)` applies the same formula to the save-only sequence of snapshots. High graph change rates indicate frequent restructuring of attribute–property bindings, whereas low rates indicate that designers mainly make local adjustments around an already-stable semantic structure.

Appendix B. Embedding-based drift analysis

To address the concern that Jaccard distance may overestimate drift for semantically proximate property changes (e.g., "red" → "burgundy"), we computed an embedding-based drift measure as a robustness check (Section 6.3.2).

B.1. Methodology

For each participant's VKG snapshots in the GenAlign condition, all property node labels were encoded using the CLIP text encoder (ViT-B/32). For each pair of consecutive snapshots (S_{i-1}, S_i), we computed:

$$\text{Drift}_{\text{emb}}(S_{i-1}, S_i) = 1 - \frac{1}{|S_i|}\sum_{p \in S_i} \max_{q \in S_{i-1}} \cos(\mathbf{e}_p, \mathbf{e}_q) \tag{B.1}$$

This measure assigns partial credit to semantically similar changes: replacing "red" with "burgundy" (cosine similarity ≈ 0.85) yields a smaller drift contribution than replacing "red" with "metal" (cosine similarity ≈ 0.20).

B.2. Results

The Jaccard-based and embedding-based drift measures showed near-perfect Pearson correlation across all 24 participants ($r = 0.998$, $p < .001$). While absolute magnitudes differed (embedding scores are lower because semantically similar properties receive partial credit), the relative ranking of participants was fully preserved (Table B.8).

This convergence provides strong evidence that Jaccard-based drift is robust to "semantic blind spots". Participants' property modifications were predominantly categorical — changing material type, shape class, or color family — rather than subtle within-cluster refinements. This behavioral pattern explains why discrete set-theoretic and continuous embedding-based measures yield concordant results, and indicates that the 22× reduction reported in Section 7.1 is consistent across both measures rather than an artifact of a single metric.

Appendix C. Hierarchical Bayesian inference evaluation

C.1. Hierarchical Bayesian inference

To further reduce semantic ambiguity, we implemented a Hierarchical Bayesian inference mechanism that infers designer-specific property triples from the accumulated interaction data in a VKG (See Algorithm 1). This mechanism considers ambiguous attributes (e.g., `modern` and `minimalist`) as higher-level priors that condition distributions over lower-level properties such as `material`, `shape`, or `texture`. Concretely, each style node in the KG is associated with a multinomial distribution over property–value pairs (p, o), estimated from observed triples during prior design sessions. Following a Dirichlet–multinomial formulation, the posterior expectation of each property is proportional to its empirical frequency with additive smoothing:

$$\mathbb{E}\big[\theta_s(p,o)\big] \;\propto\; c_s(p,o) + \alpha,$$

where $c_s(p,o)$ is the count of occurrences of property–value pair (p,o) under style s, and α is a smoothing constant. When a new query prompt includes a style token (e.g., `modern sofa`), the system first tokenizes the prompt and matches it against the set of styles in the KG. If multiple styles are matched, their posterior distributions are averaged; if none are matched, the system falls back to the global distribution across all styles in the KG.

From this aggregated distribution, the system selects the top-k most probable triples (e.g., `has-property` → `curved`, `made-of` → `brushed metal`). These inferred triples are then merged into the prompt construction or inpainting instruction stage, thereby associating abstract adjectives in concrete, user-specific property specifications.

This hierarchical mechanism provides three key benefits. First, it personalizes generation by ensuring that the same abstract attribute (e.g., `modern`) reflects meanings previously reinforced by a designer (e.g., sleek geometry + matte metal for one designer and minimal curvature + light wood for another). Second, it offers robustness by smoothing over sparse or rarely used properties, while still privileging frequently reinforced ones. Third, it enhances explainability and reusability because inferred triples are directly traceable to stored counts in the KG, allowing designers to inspect why a certain property is recommended. Ultimately, the hierarchical Bayesian inference module acts as the probabilistic core of the VKG, that is, by mapping styles to properties in a structured and adaptive manner, it enables consistent, intent-aligned generation and editing across evolving design sessions.

Table B.8
Per-participant drift comparison: Jaccard vs. embedding-based measures (GenAlign condition). Pearson $r = 0.998$, $p < .001$.

Participant	Snapshots	Jaccard drift	Embedding drift
P1	15	0.122 ± 0.130	0.016 ± 0.019
P2	14	0.105 ± 0.119	0.012 ± 0.015
P3	32	0.061 ± 0.093	0.008 ± 0.012
P4	32	0.068 ± 0.086	0.008 ± 0.012
P5	15	0.102 ± 0.101	0.012 ± 0.013
P6	12	0.129 ± 0.114	0.016 ± 0.017
P7	30	0.080 ± 0.093	0.009 ± 0.013
P8	25	0.086 ± 0.105	0.010 ± 0.013
P9	13	0.119 ± 0.112	0.015 ± 0.015
P10	11	0.124 ± 0.129	0.016 ± 0.019
P11	36	0.065 ± 0.104	0.008 ± 0.014
P12	13	0.097 ± 0.094	0.011 ± 0.011
P13	14	0.111 ± 0.181	0.014 ± 0.024
P14	13	0.080 ± 0.050	0.010 ± 0.006
P15	14	0.080 ± 0.068	0.008 ± 0.009
P16	2	0.500 ± 0.000	0.062 ± 0.000
P17	4	0.226 ± 0.171	0.028 ± 0.022
P18	7	0.158 ± 0.096	0.020 ± 0.012
P19	4	0.157 ± 0.052	0.019 ± 0.007
P20	8	0.126 ± 0.113	0.016 ± 0.016
P21	8	0.126 ± 0.136	0.015 ± 0.018
P22	5	0.075 ± 0.059	0.008 ± 0.006
P23	8	0.147 ± 0.151	0.017 ± 0.019
P24	7	0.185 ± 0.098	0.023 ± 0.012
Mean	**15.6**	**0.130 ± 0.088**	**0.016 ± 0.011**

C.2. Evaluation motivation

While our main study established the behavioral and experiential effects of VKG+HBI (RQ1–RQ3), a separate question concerns the inference layer that delivers VKG content to the generator: as a designer's VKG accumulates hundreds to thousands of properties, injecting all of them exceeds practical prompt budgets. The motivating question for this appendix is therefore not whether HBI improves output quality, but how efficiently HBI can surface a small, designer-relevant subset of properties at generation time without degrading image–text alignment. We ran a three-way comparison (Prompt Only, VKG All, VKG+HBI) at the participant level (N = 24) to characterize:

- whether HBI maintains image–text alignment (CLIP similarity) while injecting far fewer properties than exhaustive sampling;
- how much of the injected content extends, rather than restates, the user's prompt; and
- whether output stability (repetition consistency) is preserved under

selective injection.

C.3. Experimental design

C.3.1. Three comparison conditions

We compared three distinct approaches to property injection during image generation:

1. **Prompt Only**: User's original text prompt with *zero* VKG properties injected. This represents a baseline without any knowledge augmentation.
2. **VKG All**: User's prompt augmented with *all available* VKG properties at generation time. **Critical limitation**: Due to token-budget constraint of gpt-image-1's prompt API, **96.5% of actions (109/113) required random sampling to 100 properties** when the available property count exceeded this limit (mean available: 1056, max: 3136). Only 3.5% (4/113) of actions with ≤100 properties could use truly all properties without sampling. This means VKG All is fundamentally limited by token constraints and represents an underestimate of the true "all properties" condition.
3. **VKG+HBI**: User's prompt augmented with properties selected by our Hierarchical Bayesian Inference algorithm. Despite having access to the same large property pool (mean: 1056 properties, range: 0–3136), HBI selectively injects properties based on style-level priors and top-k frequency selection to balance specificity and diversity. **This is the key advantage of HBI**: intelligent selection rather than random sampling.

C.3.2. Data generation

We analyzed **113 generation actions** from 24 participants in our user study. For each action, we regenerated images under all three conditions using identical prompts to ensure fair comparison:

- **Prompt Only**: Generated 113 new images using only user prompts
- **VKG All**: Generated 113 new images with available properties (capped at 100 due to token limits; mean actually used: 96.7)
- **VKG+HBI**: Used 113 existing images from the user study (already generated with HBI)

All images were generated using gpt-image-1 with identical parameters (1024 × 1024 resolution, standard quality).

C.4. Evaluation metrics

We measured metrics across four dimensions to comprehensively evaluate HBI's effectiveness:

C.4.1. Image–text alignment: CLIP similarity

- **Purpose**: Measure how well generated images match the user's original text prompt.
- **Calculation**: We computed cosine similarity between CLIP (ViT-B/32) embeddings of the generated image and the original user prompt:

$$\text{CLIP\_Sim}(I, T) = \frac{\text{CLIP}_{\text{img}}(I) \cdot \text{CLIP}_{\text{text}}(T)}{\|\text{CLIP}_{\text{img}}(I)\| \cdot \|\text{CLIP}_{\text{text}}(T)\|} \tag{C.1}$$

Higher scores indicate better image–text alignment. This is the primary metric used in text-to-image generation research.

C.4.2. Knowledge utilization: Attribute count

- **Purpose**: Quantify how many VKG properties were actually used during generation.
- **Calculation**: Direct count of properties injected into the generation prompt for each condition:
  - Prompt Only: Always 0 properties
  - VKG All: Mean 96.7 properties (median: 100, capped due to token limits)
  - VKG+HBI: Mean 13.4 properties (median: 14, range: 0–15) selected by HBI

This metric reveals the scale of knowledge augmentation in each condition.

C.4.3. Prompt engineering: Prompt expansion ratio

- **Purpose**: Measure how much the original prompt was expanded with additional information.
- **Calculation**: We computed expansion ratio based on token count:

$$\text{Expansion\_Ratio} = \frac{\text{Token\_Count(Augmented\_Prompt)}}{\text{Token\_Count(Original\_Prompt)}} \tag{C.2}$$

For both VKG conditions, we also measured three sub-metrics:

- **Jaccard Similarity**: $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$ between prompt keywords and property keywords
- **Overlap Rate**: Percentage of prompt keywords found in VKG properties
- **Expansion Rate**: Percentage of VKG properties not in original prompt (novelty)

C.4.4. Stability: Repetition consistency

- **Purpose**: Measure whether regenerating the same prompt produces consistent results.
- **Calculation**: For participants who repeated identical prompts, we computed CLIP similarity between pairs of regenerated images:

$$\text{Consistency} = \text{CLIP\_Sim}(I_1, I_2) \tag{C.3}$$

Higher consistency indicates more stable, predictable generation behavior.

C.4.5. Prompt length effect

- **Purpose**: Test whether prompt length influences generation quality differently across conditions.
- **Calculation**: We categorized prompts into short (≤7 words) and long (8+ words), then compared mean CLIP similarity using Mann–Whitney U test.

C.4.6. Property count sensitivity

- **Purpose**: Detect over-specification (too many properties hurting quality) or under-specification (too few properties hurting quality).
- **Calculation**: We categorized actions by VKG property count into five bins:
  - Very Low: 0–100 properties
  - Low: 101–500 properties
  - Medium: 501–1000 properties
  - High: 1001–1500 properties
  - Very High: 1,501+ properties

We then applied Kruskal–Wallis H-test to detect significant differences in CLIP similarity across these categories, and computed Pearson correlation between property count and CLIP similarity.

C.5. Results

Table 7 summarizes the comparison across all metrics.

C.5.1. Key findings

Efficiency Through Selective Injection. VKG All injected 96.7 properties on average through random sampling (capped at 100 due to `gpt-image-1`'s prompt token limit), while VKG+HBI injected only 13.4 properties (median: 14, range: 0–15) through top-k selection ($k = 15$). Despite using 7.2× fewer properties, VKG+HBI achieved comparable image–text alignment at the participant level (CLIP similarity 0.318 vs. 0.315; 85.9% token savings), indicating that selective injection delivers equivalent alignment far more efficiently than exhaustive random sampling.

CLIP Similarity Invariance. All three conditions produced comparable CLIP similarity (Prompt Only 0.316, VKG All 0.315, VKG+HBI 0.318; difference < 1%). Following participant-level re-analysis ($N = 24$), which replaces the earlier action-level test to avoid pseudoreplication (Section 6.3), the Friedman test was non-significant ($\chi^2(2) = 3.74$, $p = 0.154$). All post-hoc Wilcoxon pairwise comparisons with Holm–Bonferroni correction were likewise non-significant (Prompt Only vs. VKG All: $W = 110.5$, $r = 0.20$, $p = 0.40$; Prompt Only vs. VKG+HBI: $W = 99.0$, $r = -0.28$, $p = 0.47$; VKG All vs. VKG+HBI: $W = 92.0$, $r = -0.33$, $p = 0.51$). No condition reliably outperforms the others in CLIP similarity; HBI's contribution is efficiency, not alignment gain.

Knowledge Expansion. When measured against the *available property pool*, both VKG conditions showed similar overlap with user prompts: VKG All had 40.5% overlap (Jaccard: 0.032, expansion rate: 59.5%), while VKG+HBI had 39.7% overlap (Jaccard: 0.030, expansion rate: 98.2%). VKG+HBI's higher expansion rate (98.2% vs. 59.5%) demonstrates that HBI selects more diverse, novel properties beyond what users articulate, while VKG All's random sampling captures fewer unique properties due to token constraints.

Improved Consistency. VKG+HBI showed +1.5% higher repetition consistency (0.915) compared to Prompt Only (0.900) when regenerating identical prompts. VKG All showed intermediate consistency (0.908, from 25 repeated prompts). This suggests that property-augmented prompts produce more stable outputs, with HBI's intelligent selection providing greater stability than random sampling.

Prompt Length Effect. All three conditions showed similar patterns: longer prompts (8+ words) achieved higher CLIP similarity than shorter prompts (≤7 words), with increases of +4.4% (Prompt Only), +2.8% (VKG All), and +4.4% (VKG+HBI). Mann–Whitney U tests revealed no statistically significant differences (all $p > 0.15$), indicating that prompt length effects are consistent across conditions.

Property Count Sensitivity. At the participant level ($N = 24$), neither VKG condition showed a significant relationship between available property count and CLIP similarity (VKG All: $r = 0.060$, $p = 0.79$; VKG+HBI: $r = -0.058$, $p = 0.79$), consistent with Table 7. We therefore found no evidence of over-specification: even actions drawing on pools of 1,500+ properties maintained CLIP similarity comparable to smaller pools, and HBI's selective injection preserved this alignment while injecting far fewer properties.

C.6. HBI's operational role

Despite CLIP similarity invariance, HBI plays the following operational roles:

1. Efficiency through selective injection. HBI maintains comparable image–text alignment (CLIP similarity 0.318 vs. VKG All's 0.315) while injecting only 13.4 properties versus VKG All's 96.7 (7.2× fewer, an 85.9% reduction in property-derived prompt tokens). This indicates that HBI's style-level priors and frequency-based ranking prioritize designer-relevant properties without sacrificing alignment.

2. Semantic expansion. HBI's 98.2% expansion rate (novel properties not in original prompt) shows that VKG provides rich, complementary context users would not articulate themselves. The low Jaccard similarity (0.030) confirms that VKG and user prompts capture different semantic dimensions. Compared to VKG All's 59.5% expansion rate (limited by random sampling under token constraints), this reflects a higher proportion of novel properties.

3. Stability and predictability. HBI's +1.5% consistency improvement (0.915 vs. 0.900), while modest, suggests that property-augmented prompts reduce generation variance. VKG All's intermediate consistency (0.908) suggests that style-conditioned property selection provides greater stability than random sampling.

Appendix D. User study

D.1. Baseline interface

The Baseline interface used in the user study is shown in Fig. D.1. It mirrors *GenAlign*'s gallery with keyword-based search and image browsing, providing separate tabs to explore, save, and view generated images (Fig. D.1-a). Users enter keywords in the search tab and view the results on the page (Fig. D.1-b). Unlike *GenAlign*, which generates images conditioned on an accumulated cognitive VKG, the Baseline generates images solely from the user's prompt (Fig. D.1-c). Specifically, text-to-image generation follows the pipeline in Appendix E but without any VKG state; the user's prompt is passed directly to the model, and each generation is independent (no conversational history or negotiation across steps). Similar to *GenAlign*, users can save preferred images from the list (Fig. D.1-e); instead of the property-editing tab, the Baseline offers only image enlargement (Fig. D.1-d). All VKG-related capabilities are removed: the VKG panel, segment-scoped Add/Edit/Transfer operations, saved-segment reuse, and the Cognitive Map. Only global prompt edits and image-level view/save are available. All other factors are held constant with *GenAlign* (model stack, sampler/seed policy, resolution/time cap, and logging), isolating the presence of part-scoped property bindings as the single systematic difference. In our study configuration, steerage in *GenAlign* arises from the operable VKG that is *assembled from user actions* (search/save/edit/transfer) and then used to condition subsequent generations; no additional dialogic guidance layer is introduced. By contrast, the Baseline omits this operable state entirely.

D.2. Interview questions

We conducted interviews with professional designers to understand how semantic disambiguation develops throughout the design process, addressing intent expression, VKG–related features, and satisfaction with the final outcomes. We also asked how the participants currently use GenAI in their practice and how they believe it should evolve. The interview questions were as follows.

Expressing intent and editing operations

- How did you express your design intent (e.g., text prompts and segment-level edits)? Which operations were most effective?
- When editing, which attributes did you adjust first (e.g., color, material, or shape), and why?
- When modifying a specific segment, what cues did you perceive as consistent with, or contrary to, your intent?
- Did segment-level attribute editing appear easier or more concrete than using text prompts? Please explain.
- How did segment-based actions (e.g., property copy/paste across segments) facilitate the design progress?
- What initial strategy did you use (global adjustments vs. incremental trials) and why?

Support from the knowledge graph (KG) and suggestions

- In what ways did the knowledge graph or attribute suggestions assist you, and how did they guide your exploration?
- How did you use the system-suggested keywords (attributes and relations) when selecting attributes or composing prompts?
- Did the graph aid you in understanding the relationships between the designs or fine-grained attributes? How?
- If you had edited without a structured graph or recommendations, what difficulties do you think you would have encountered?
- Were there moments you decided "I should use this" while exploring/using the graph? What triggered that decision?
- How important is it to explicitly recognize the visual attributes of your intended design? Why?

Outcome alignment and satisfaction

- To what extent did the generated results align with your intent/-expectations? In which cases were you satisfied or dissatisfied, and why?
- Among the generated images, which results appeared most consistent with your expectations? Which differed the most, and why?
- Did alignment improve over time? If so, when did you first notice this improvement?
- Which combinations of attributes tended to produce the most satisfying results?

General background

- When do you typically use GenAI in your design process?
- What role can GenAI play for designers (what can or cannot be replaced)?

D.3. Session action sequences

Fig. D.2 presents the complete action sequences for all 24 participants across the entire study session. Each row represents one participant's interaction timeline, with symbols denoting different action types: search queries (green squares), image saves (circles), segment operations (stars for save_seg, triangles for add/edit/delete attributes), generation commands (gen_query, gen_transfer, gen_modify in red/orange/yellow diamonds), and cognitive map interactions (yellow squares). Thumbnail images show key design states at generation moments, with smiley-face icons indicating final saved designs. Red annotations highlight critical semantic disambiguation moments (e.g., P2's "define 'minimal'", P5's "check 'minimal' & 'simple'", P8's "save segments for 'sleek'"). This visualization reveals diverse exploration strategies: some participants (P1, P7, P13, P14) engaged in intensive segment-level editing with frequent property transfers, while others (P10, P16, P22) relied primarily on iterative text-based generation with selective segment saves. The density and diversity of action types demonstrate how designers dynamically negotiated semantic ambiguity through mixed-initiative interaction, progressively building designer-specific VKGs that stabilized around coherent property vocabularies.

Appendix E. LLM structure prompts

We present the Visual Knowledge Graph (VKG) schema, construction pipeline, and LLM-based extraction prompts that enable our system to accumulate design knowledge across user sessions.

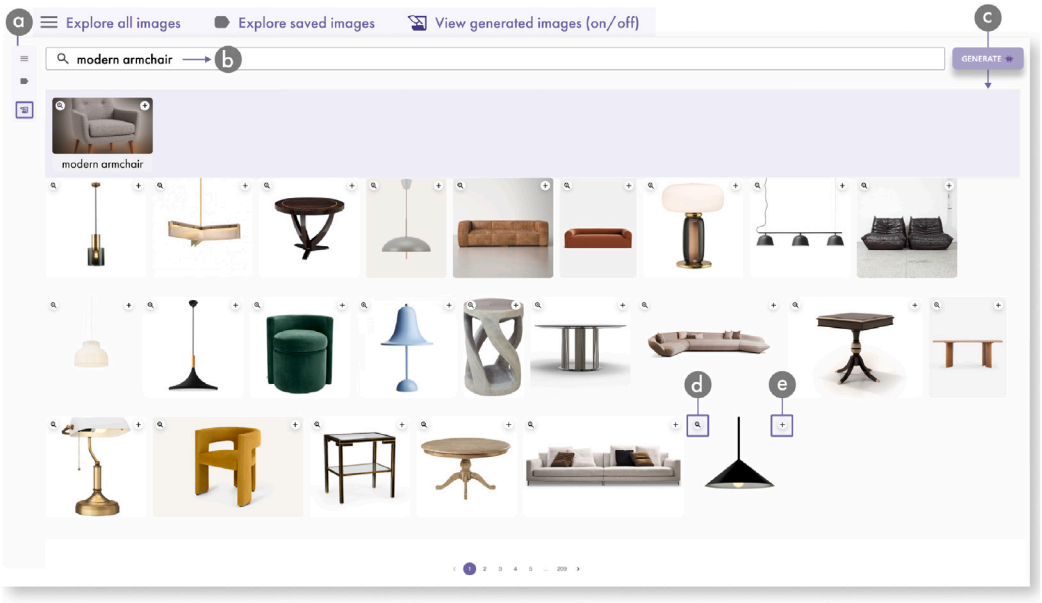


Fig. D.1. Baseline system interface: (a) navigation tabs, (b) search, (c) generate, (d) expand the image, and (e) save the image.

Table D.1
VKG Schema: Node types, edge predicates, and their purposes.

Component	Type/Predicate	Purpose
Node types (by group field)		
`group=0`	Visual Attribute (Style) nodes	Global aesthetic descriptors (e.g., "minimalist", "modern").
`group=1`	Segment nodes	Object parts with unique IDs (e.g., `seg_0`).
`group=2`	Attribute nodes	Visual properties (e.g., "brown", "fabric").
Based on edge predicates (VisualSem [43])		
	`is-a`, `part-of`, `has-part`	Object classification and composition.
	`made-of`, `has-property`	Material and appearance.
	`used-for`, `used-by`, `subject-of`, `receives-action`	Purpose and interaction.
	`related-to`, `gloss-related`, `synonym`	Conceptual associations.
	`located-at`	Positional information.
	`is-style-of`	Segment-to-Visual Attribute linking.
Storage files		
Graph	`current_graph.json`	`{nodes: [...], links: [...]}`
Triple	`current_triples.json`	`[{subject, predicate, object, . . . }]`
Combined	`current_full.json`	Graph + triples + metadata.
State	`kg_state.json`	Session-persistent graph state.

E.1. VKG schema and structure

Overview. The VKG is a directed graph $G = (V, E)$ where nodes V represent design entities (styles, segments, attributes) and edges E represent semantic relationships (VisualSem predicates). The graph is stored as JSON with two primary data structures:

- **Graph representation** (`current_graph.json`): Network topology with `nodes` (id, label, group) and `links` (source, target, label/predicate).
- **Triple representation** (`current_triples.json`): Flat list of {subject, predicate, object} triples with metadata (segId, filename).

Node Types. Nodes are categorized by `group` field:

- `group=0`: Style nodes (e.g., "minimalist", "modern", "rustic") extracted from global image analysis.
- `group=1`: Segment nodes (e.g., `seg_0`, `seg_1`) representing object parts with unique IDs.
- `group=2`: Property nodes (e.g., "brown", "fabric", "cylindrical") prefixed with `attr:` in graph representation.

Edge Types (Predicates). Edges use VisualSem [43] predicates plus one custom predicate:

- `is-a`, `part-of`, `has-part`: Taxonomic/mereological relations
- `made-of`, `has-property`: Physical attributes
- `used-for`, `used-by`, `subject-of`, `receives-action`: Functional roles
- `related-to`, `gloss-related`, `synonym`: Semantic associations
- `located-at`: Spatial relations
- `is-style-of`: Add predicate linking segments to global visual attribute nodes

Dual Representation Rationale. The system maintains both graph (`nodes/links`) and triple formats:

- Graph format enables efficient D3.js visualization in the frontend and network-based queries.
- Triple format supports HBI inference (style-level prior aggregation) and LLM prompt construction.

Table D.1 summarizes the VKG schema.

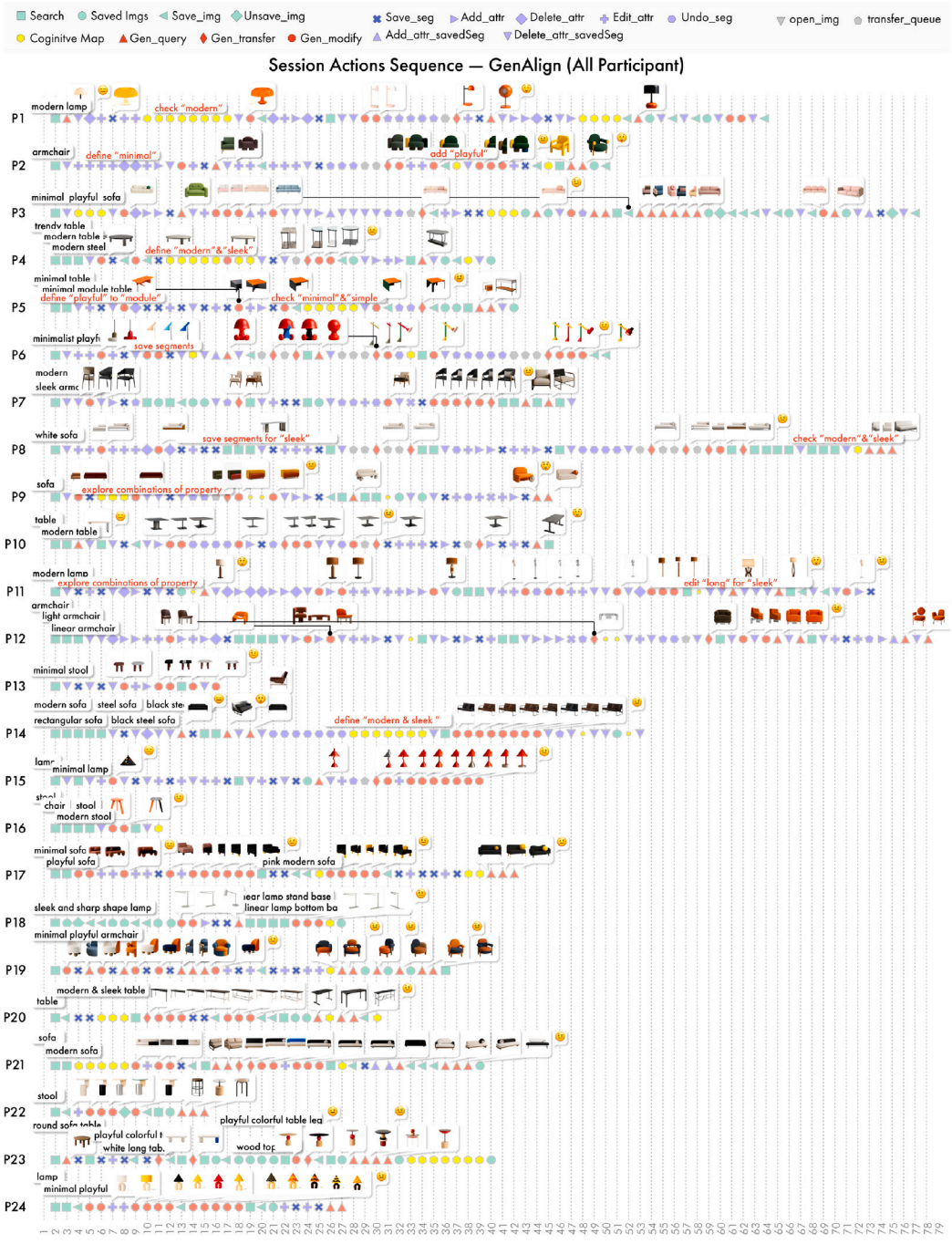


Fig. D.2. Complete action sequences for all 24 participants in the GenAlign condition, showing temporal progression of search, save, segment operations, generation commands, and cognitive map interactions. Each row represents one participant's session timeline with thumbnails indicating key design states and smiley icons marking final saved designs.

E.2. VKG construction pipeline

Overview. VKG is constructed incrementally through four model-based extraction stages executed on each user-uploaded or generated image (Fig. E.1):

1. **Image Segmentation** (Semantic-SAM): Partition image into regions using multi-granularity masks; filter foreground segments by IoU with background-removed mask (threshold = 0.5).
2. **Segment Labeling** (GPT-4o + function calling): For each segment, extract object/part label (e.g., "armrest", "table-leg") in lowercase-kebab-case.
3. **Style Extraction** (GPT-4o + function calling): Analyze full image to extract 1–3 global style adjectives (e.g., "minimalist", "modern").
4. **Visual Triple Extraction** (GPT-4o + function calling): For each segment, generate VisualSem triples capturing color, material, texture, shape, and function; automatically inject `is-style-of` triples linking segments to global styles.

Post-Processing and Storage.

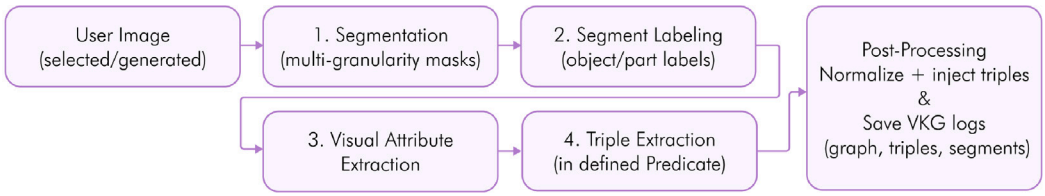


Fig. E.1. End-to-end pipeline for transforming a selected or generated user image into a structured Visual Knowledge Graph (VKG). The system performs segmentation, segment labeling, visual attribute extraction, and predicate-based triple extraction. In post-processing, extracted triples are normalized, injected into the VKG, and stored on the server as VKG logs (graph, triples, and segment images).

- **Triple normalization**: All predicates and objects converted to lowercase-kebab-case; duplicate triples removed based on (predicate, object) pairs.
- **Canonical triple injection**: System enforces `is-a` triple (matching segment label) and `is-style-of` triple (linking to global styles) for every segment.
- **Incremental storage**: Each user action (search, save_seg, gen_query, etc.) triggers delta updates saved to `kg/<timestamp>_<action>/` with:
  - `current_graph.json`: Full graph state at action time
  - `current_triples.json`: Full triple list at action time
  - `current_full.json`: Combined graph + triples + metadata
  - `segments/<seg_id>.png`: Cropped segment images
- **Session persistence**: Graph state persisted and loaded on session initialization; supports upsert/remove operations for interactive editing.

E.3. LLM instruction families

We use four instruction families with function-call schemas to guarantee machine-readable JSON output:

- Segment labeling: returns a single object/part label for a highlighted region.
- Style extraction: returns 1–3 global style adjectives for the entire image.
- Visual triple extraction: returns VisualSem-style attribute triples for a segment.
- Generative editors: three pathways driven by structured prompts derived from triples (generation by transferring property, generation by modifying property, and prompt generation based on KG).

All instructions were delivered with function-call schemas (for label, style, and triples) to guarantee machine-readable JSON, near-zero temperature for determinism, and lowercase-kebab-case normalization. Post-processing replaced/augmented certain triples (e.g., canonical `is-a` and optional `part-of`) before they were recorded in `current_triples.json` and re-used for Hierarchical Bayesian Inference(HBI).

E.4. Segment labeling

Goal. Consider a full image and an SVG path highlighting a region, output the most specific object or object-part label in the lowercase-kebab-case (e.g., `sofa-armrest`) or `background` if no valid region is selected.

Segment Labeling

```
system_msg = {
    "role": "system",
    "content": (
        "You are a vision model that identifies which OBJECT or OBJECT-PART "
        "is highlighted in a product photo. Answer in lowercase kebab-case "
        "(e.g., 'sofa-armrest', 'table-leg', 'lamp-shade', 'vase'). "
        "If nothing is selected, return 'background'."
    )
}

FUNCTION_DEF = {
    "name": "label_segment",
    "description": "Returns a label for the highlighted segment.",
    "parameters": {
        "type": "object",
        "properties": {"label": {"type": "string"}},
        "required": ["label"]
}}

def _img_to_b64(pil):
    import io, base64
    buf = io.BytesIO(); pil.save(buf, "PNG")
    return base64.b64encode(buf.getvalue()).decode("utf-8")

def gpt_label_segment(pil_crop, full_img, path_svg,
                      model="GPT-4o", temperature=0.0, top_p=0.1):
    crop_b64 = _img_to_b64(pil_crop)
    full_b64 = _img_to_b64(full_img)
    resp = openai.chat.completions.create(
        model=model, temperature=temperature, top_p=top_p,
        messages=[
            system_msg,
            {"role":"user","content":[
                {"type":"text","text":
                 "The SVG path below indicates the highlighted region. "
                 "Return the best matching object or object-part label."},
                {"type":"text","text": f"[path] {path_svg}..."},
                {"type":"image_url","image_url":{"url":f"base64,{full_b64}"}},
                {"type":"image_url","image_url":{"url":f"base64,{crop_b64}"}}
            ]}
        ],
        tools=[{"type":"function","function":FUNCTION_DEF}],
        tool_choice={"type":"function","function":{"name":"label_segment"}}
    )
    args_json = resp.choices[0].message.tool_calls[0].function.arguments
    import json; args=json.loads(args_json) if isinstance(args_json,str) else args_json
    return args["label"]
```

E.5. Visual attribute (Style) extraction

Goal Summarize the image's overall visual attribute (style) by returning 1–3 concise adjectives in lowercase-kebab-case (e.g., `modern`, `minimalist`, `luxurious`).

Overall Visual Attribute (Style) Extraction

```
STYLE_SYS = {
  "role":"system",
  "content":("You are a vision stylist. Summarise the OVERALL visual style "
             "with 1-3 short adjectives in lowercase kebab-case. "
             "Return JSON {\"styles\":[...]}." )
}
STYLE_FDEF = {
  "name":"return_styles",
  "description":"Return overall style adjectives.",
  "parameters":{"type":"object","properties":{
      "styles":{"type":"array","items":{"type":"string"}}},
    "required":["styles"]}
}

import re
def _kebab(s: str) -> str:
    return re.sub(r"[^a-z0-9]+","-", s.lower()).strip("-")

def detect_image_style(pil_img):
    import io, base64, json
    buf = io.BytesIO(); pil_img.save(buf, "PNG")
    img_b64 = base64.b64encode(buf.getvalue()).decode()
    resp = openai.chat.completions.create(
        model="GPT-4o", temperature=0,
        messages=[STYLE_SYS, {"role":"user","content":[
          {"type":"text","text":"Describe the overall style."},
          {"type":"image_url","image_url":{"url":f"{img_b64}"}}
        ]}],
        tools=[{"type":"function","function":STYLE_FDEF}],
        tool_choice={"type":"function","function":{"name":"return_styles"}}
    )
    args = resp.choices[0].message.tool_calls[0].function.arguments
    data = json.loads(args) if isinstance(args,str) else args
    styles = [_kebab(s) for s in data["styles"]]
    return list(dict.fromkeys(styles))[:3]   # keep order, max 3
```

E.6. Visual triple extraction

Goal. For the highlighted segment, return the normalized VisualSem-style triples {subject, predicate, object} that capture color, material, texture, shape, function, and style, always including an `is-a` label and at least one `is-style-of` entry, with predicates and objects in lowercase-kebab-case.

VisualSem-style Triple Extraction

```
VISUALSEM_SYSTEM_MSG = {
 "role":"system",
 "content":(
   "You are a vision+language assistant. Given a highlighted product segment, "
   "extract ONLY visually-relevant relations into JSON triples. "
   "Predicates: is-a, has-part, related-to, used-for, used-by, subject-of, "
   "receives-action, made-of, has-property, gloss-related, synonym, part-of, "
   "located-at, and custom is-style-of. "
   "Focus on color, material, texture, shape, function, style; include at least "
   "one is-a and one is-style-of when applicable. Use lowercase kebab-case."
 )}

VISUALSEM_FUNCTION_DEF = {
 "name":"return_visual_triples",
 "description":"Return the visual-attribute triples for this segment.",
 "parameters":{"type":"object","properties":{
  "triples":{"type":"array","items":{"type":"object","properties":{
    "subject":{"type":"string"},
    "predicate":{"type":"string","enum":[
      "is-a","has-part","related-to","used-for","used-by","subject-of",
      "receives-action","made-of","has-property","gloss-related","synonym",
      "part-of","located-at","is-style-of"]},
    "object":{"type":"string"}}, "required":["subject","predicate","object"]}}},
  "required":["triples"]}
}

def build_segment_triples(seg_img, seg_id, global_styles):
    import io, base64, json
    buf = io.BytesIO(); seg_img.save(buf, "PNG")
    img_b64 = base64.b64encode(buf.getvalue()).decode()
    resp = openai.chat.completions.create(
        model="GPT-4o", temperature=0,
        messages=[VISUALSEM_SYSTEM_MSG, {"role":"user","content":[
          {"type":"text","text":
            f"The subject id is '{seg_id}'. Return all applicable visual triples. "
            "The is-a triple MUST use exactly that label."},
          {"type":"image_url","image_url":{"url":f"{img_b64}"}}
        ]}],
        tools=[{"type":"function","function":VISUALSEM_FUNCTION_DEF}],
        tool_choice={"type":"function","function":{"name":"return_visual_triples"}}
    )
    call = resp.choices[0].message.tool_calls[0]
    args = call.function.arguments
    triples = (json.loads(args) if isinstance(args,str) else args)["triples"]
    exist = {(t["predicate"], t["object"]) for t in triples}
    for st in (global_styles or []):
        if ("is-style-of", st) not in exist:
            triples.append({"subject":seg_id, "predicate":"is-style-of", "object":st})
    seen, uniq = set(), []
    for t in triples:
        k = (t["predicate"], t["object"])
        if k not in seen: uniq.append(t); seen.add(k)
    return uniq
```

E.7. Generate editor

Goal. Provide mask-local image editing operators that modify only the selected region while preserving all the unmasked pixels.

Generate by transferring property.

Goal. Replace the pixels inside the selected mask by compositing the content copied from a source region/image into the target mask, while preserving all other regions.

Mask Transfer (paste-by-mask)

```python
from PIL import Image, ImageDraw
from svgpathtools import parse_path
import numpy as np, cv2

def path_to_mask(path_d: str, W: int, H: int):
    sp_path = parse_path(path_d)
    seg_len = sp_path.length(); n = max(int(seg_len // 1), 600)
    pts = [sp_path.point(t/n) for t in range(n+1)]
    xy = [(p.real, p.imag) for p in pts]
    mask = Image.new("L", (W, H), 0)
    ImageDraw.Draw(mask).polygon(xy, fill=255)
    k = np.ones((5,5), np.uint8)
    closed = cv2.morphologyEx(np.array(mask), cv2.MORPH_CLOSE, k)
    return Image.fromarray(closed)

def gen_transfer(base_img: Image.Image, src_img: Image.Image, svg_path: str):
    """Paste src_img into base_img inside svg_path mask; others untouched."""
    base = base_img.copy()
    mask = path_to_mask(svg_path, *base.size).convert("L")
    bbox = mask.getbbox()
    if not bbox: return base
    w, h = bbox[2]-bbox[0], bbox[3]-bbox[1]
    tile = src_img.resize((w, h), Image.LANCZOS).convert("RGBA")
    base = base.convert("RGBA")
    base.paste(tile, (bbox[0], bbox[1]), mask.crop(bbox))
    return base
```

Generate by modifying property.

Goal. Modify only the selected mask according to the KG triples (e.g., color, material, and shape), while preserving the remainder of the image (see Fig. E.2).

Attribute-driven Modify (mask-local)

```python
def triples_to_prompt(triples, target_label=None):
    def ph(pred, obj):
        return {
          "color": f"{obj} color",
          "material": f"made of {obj}",
          "style": f"{obj} style",
          "shape": f"{obj} shape",
          "leg-type": f"{obj} legs",
          "pattern": f"{obj} pattern",
          "texture": f"{obj} texture"
        }.get(pred, f"{pred}: {obj}")
    phrases = [ph(t["predicate"], t["object"]) for t in triples]
    part = f"the {target_label}" if target_label else "the selected part"
    return (f"Design {part} with " + ", ".join(phrases) +
            ". Do not modify the rest of the image.")

def gen_modify(base_img, mask_svg, triples, target_label=None):
    """Build an edit instruction for mask-local modification."""
    instruction = triples_to_prompt(triples, target_label)
    # call your image model with (base_img, mask=mask_svg, text=instruction)
    # result = image_edit_api(base_img, mask_svg, instruction)
    return instruction
```

Algorithm 1 Hierarchical Bayesian Inference (HBI) for Property Selection

Require: User prompt p, VKG graph G with segments and triples, top-k parameter k (default: 15)
Ensure: Top-k property triples $\{(pred, obj)\}$

$tokens \leftarrow \textsc{Tokenize}(p)$
$S_{\text{all}} \leftarrow \{s \mid (\text{seg}, \text{“is-style-of”}, s) \in G\}$
$S_{\text{matched}} \leftarrow \{t \in tokens \mid t \in S_{\text{all}}\}$
if $S_{\text{matched}} = \emptyset$ **then**
  $S_{\text{matched}} \leftarrow S_{\text{all}}$ ▷ Use all styles if no match
end if
$freq \leftarrow \{\}$ ▷ style → counts of $(pred, obj)$
for each segment seg in G **do**
  $style \leftarrow$ first $(\text{seg}, \text{“is-style-of”}, s)$
  if $style \neq$ null **then**
    for each $(\text{seg}, pred, obj)$ with $pred \notin \{\text{“is-a”}, \text{“is-style-of”}\}$ **do**
      $freq[style][(pred, obj)] \leftarrow freq[style][(pred, obj)] + 1$
    end for
  end if
end for
$total \leftarrow$ Counter()
for $style \in S_{\text{matched}}$ **do**
  for $(pred, obj), count \in freq[style]$ **do**
    $total[(pred, obj)] \leftarrow total[(pred, obj)] + \dfrac{count}{|S_{\text{matched}}|}$
  end for
end for
return $\textsc{TopK}(total, k)$

Prompt generation based on KG.

Goal. Generate or edit content by constructing a prompt from the current VKG (or KG-inferred priors) rather than from free text, ensuring that the output adheres to the attribute constraints encoded in the VKG. Algorithm 1 formalizes the HBI process for property selection.

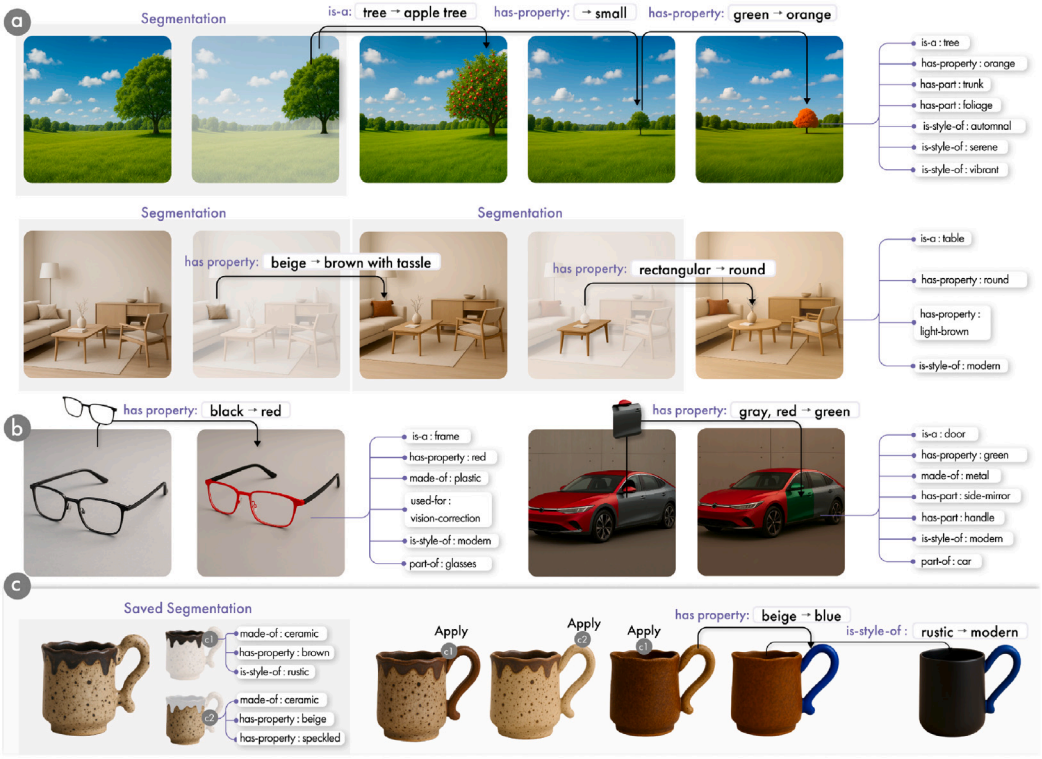


Fig. E.2. Examples illustrating the generalizability of VKG-based generation across domains. For each row, the system takes an original image (left), segments it and builds a visual knowledge graph with part-level attributes (center), then applies localized edits by modifying segment properties such as color, material, and style (right), producing intent-aligned generations for various domains.

KG-based Prompt Generation

```python
from collections import defaultdict, Counter
import re

def infer_triples_from_hierarchical_model_sampling(prompt: str, graph: list,
                                                   top_k: int = 15):
    """Style priors -> aggregate attribute frequencies -> top-k triples."""
    def tokenize(t): return re.findall(r'\w+', t.lower())
    def extract_styles(g):
        S=set()
        for e in g:
            for seg in e.get("segments", []):
                for tri in seg.get("triples", []):
                    if tri["predicate"]=="is-style-of": S.add(tri["object"].lower())
        return S
    def build_style_priors(g):
        freq=defaultdict(Counter)
        for e in g:
            for seg in e.get("segments", []):
                style = next((t["object"].lower()
                              for t in seg.get("triples", [])
                              if t["predicate"]=="is-style-of"), None)
                if not style: continue
                for t in seg.get("triples", []):
                    if t["predicate"] not in {"is-style-of","is-a"}:
                        freq[style][(t["predicate"], t["object"])] += 1
        return freq
    toks = tokenize(prompt); styles = extract_styles(graph)
    matched = [t for t in toks if t in styles] or list(styles)
    priors = build_style_priors(graph); total = Counter()
    for st in matched:
        for attr, cnt in priors[st].items():
            total[attr] += cnt / max(len(matched),1)
    return [{"predicate":p, "object":o} for (p,o),_ in total.most_common(top_k)]

def generate_image_from_attributes(prompt, triples=None, graph=None):
    """Build final prompt from KG/priors; call your generator (excerpt)."""
    triples = triples or (
        infer_triples_from_hierarchical_model_sampling(prompt, graph)
                          if graph else [])
    attr_triples = [t for t in triples
                    if t.get("predicate") not in {"is-a","label","name","has-part"}]
    attr_prompt = ", ".join([f"{t['object']}" for t in attr_triples
                             if t.get("predicate","").startswith("has-")])  # optional
    prompt_text = f"{prompt}, {attr_prompt}" if attr_prompt else prompt
    # invoke image generator here (omitted)
    return prompt_text
```

Appendix F. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.aei.2026.105165.

Data availability

The system implementation, study data, and analysis scripts are publicly available at https://cosjiimos.github.io/GenAlign/.

References

[1] M. Bernal, J.R. Haymaker, C. Eastman, On the role of computational support for designers in action, Des. Stud. 41 (2015) 163–182, http://dx.doi.org/10.1016/j.destud.2015.08.001.

[2] S. Tao, I. Liang, C. Peng, Z. Wang, S. Palani, S.P. Dow, DesignWeaver: Dimensional scaffolding for text-to-image product design, in: Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems, 2025, pp. 1–26, http://dx.doi.org/10.1145/3706598.3714211.

[3] S. Park, Y. Song, S. Lee, J. Kim, J. Seo, Leveraging multimodal LLM for inspirational user interface search, in: Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems, 2025, pp. 1–22, http://dx.doi.org/10.1145/3706598.3714213.

[4] W.W. Gaver, J. Beaver, S. Benford, Ambiguity as a resource for design, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 2003, pp. 233–240, http://dx.doi.org/10.1145/642611.642653.

[5] P. Dourish, Implications for design, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 2006, pp. 541–550, http://dx.doi.org/10.1145/1124772.1124855.

[6] Y. Pan, Reflexivity of account, professional vision, and computer-supported cooperative work: Working in the maritime domain, Proc. ACM Hum.-Comput. Interact. 5 (CSCW2) (2021) 1–32, http://dx.doi.org/10.1145/3479514.

[7] Z. Sevener, A semantic differential study of the influence of aesthetic properties on product pleasure, in: Proceedings of the 2003 International Conference on Designing Pleasurable Products and Interfaces, 2003, pp. 150–151, http://dx.doi.org/10.1145/782896.782938.

[8] E. Foong, J.O. Kim, M. Dontcheva, E.M. Gerber, CrowdFolio: Understanding how holistic and decomposed workflows influence feedback on online portfolios, Proc. ACM Hum.-Comput. Interact. 5 (CSCW1) (2021) 1–31, http://dx.doi.org/10.1145/3449096.

[9] T. Zhou, C. Liu, J.K. Yang, J. Huang, filtered. ink: Creating dynamic illustrations with SVG filters, in: Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, 2023, pp. 1–15, http://dx.doi.org/10.1145/3544548.3581051.

[10] M. Hassenzahl, The interplay of beauty, goodness, and usability in interactive products, Hum.–Comput. Interact. 19 (4) (2004) 319–349, http://dx.doi.org/10.1207/s15327051hci1904_2.

[11] Y. Jeon, S. Jin, P.C. Shih, K. Han, Fashionq: an ai-driven creativity support tool for facilitating ideation in fashion design, in: Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, 2021, pp. 1–18, http://dx.doi.org/10.1145/3411764.3445093.

[12] F. Gmeiner, N. Marquardt, M. Bentley, H. Romat, M. Pahud, D. Brown, A. Roseway, N. Martelaro, K. Holstein, K. Hinckley, et al., Intent tagging: Exploring micro-prompting interactions for supporting granular human-GenAI co-creation workflows, in: Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems, 2025, pp. 1–31, http://dx.doi.org/10.1145/3706598.3713861.

[13] W.-F. Wang, C.-T. Lu, N. Ponsa i Campanyà, B.-Y. Chen, M.Y. Chen, AIdeation: Designing a human-AI collaborative ideation system for concept designers, in: Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems, 2025, pp. 1–28, http://dx.doi.org/10.1145/3706598.3714148.

[14] Z. Liu, Y. Yu, H. Ouyang, Q. Wang, K.L. Cheng, W. Wang, Z. Liu, Q. Chen, Y. Shen, Magicquill: An intelligent interactive image editing system, in: Proceedings of the Computer Vision and Pattern Recognition Conference, 2025, pp. 13072–13082, http://dx.doi.org/10.1109/cvpr52734.2025.01220.

[15] Y. Feng, X. Wang, K.K. Wong, S. Wang, Y. Lu, M. Zhu, B. Wang, W. Chen, Promptmagician: Interactive prompt engineering for text-to-image creation, IEEE Trans. Vis. Comput. Graphics 30 (1) (2023) 295–305, http://dx.doi.org/10.1109/TVCG.2023.3327168.

[16] A. Mahdavi Goloujeh, A. Sullivan, B. Magerko, Is it AI or is it me? Understanding users' prompt journey with text-to-image generative AI tools, in: Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems, 2024, pp. 1–13, http://dx.doi.org/10.1145/3613904.3642861.

[17] K. Sha, Y. Li, Y. Dong, N. Zhang, Modelling the dynamics of customer requirements considering their lability and sensitivity in product development, Adv. Eng. Inform. 59 (2024) 102296, http://dx.doi.org/10.1016/j.aei.2023.102296.

[18] S.L. Star, The structure of ill-structured solutions: Boundary objects and heterogeneous distributed problem solving, in: Distributed Artificial Intelligence, Elsevier, 1989, pp. 37–54, http://dx.doi.org/10.1016/B978-1-55860-092-8.50006-X.

[19] B. Ewenstein, J. Whyte, Knowledge practices in design: the role of visual representations asepistemic objects', Organ. Stud. 30 (1) (2009) 07–30, http://dx.doi.org/10.1177/0170840608083014.

[20] S. Brade, B. Wang, M. Sousa, S. Oore, T. Grossman, Promptify: Text-to-image generation through interactive prompt exploration with large language models, in: Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology, 2023, pp. 1–14, http://dx.doi.org/10.1145/3586183.3606725.

[21] Z. Wang, Y. Huang, D. Song, L. Ma, T. Zhang, Promptcharm: Text-to-image generation through multi-modal prompting and refinement, in: Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems, 2024, pp. 1–21, http://dx.doi.org/10.1145/3613904.3642803.

[22] W.-F. Wang, T.-Y. Lee, C.-T. Lu, C.-W. Hsu, N. Ponsa i Campanyà, Y. Chen, M.Y. Chen, B.-Y. Chen, GenTune: Toward traceable prompts to improve controllability of image refinement in environment design, in: Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology, 2025, pp. 1–21, http://dx.doi.org/10.1145/3746059.3747774.

[23] J. Choi, S.W. Lee, K.H. Hyun, GenPara: Enhancing the 3D design editing process by inferring users' regions of interest with text-conditional shape parameters, in: Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems, 2025, pp. 1–21, http://dx.doi.org/10.1145/3706598.3713502.

[24] C. Vachha, Y. Kang, Z. Dive, A. Chidambaram, A. Gupta, E. Jun, B. Hartmann, Dreamcrafter: Immersive editing of 3D radiance fields through flexible, generative inputs and outputs, in: Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems, 2025, pp. 1–13, http://dx.doi.org/10.1145/3706598.3714312.

[25] J.-K. Lee, H. Jeong, Y. Kim, S.H. Cha, Creating spatial visualizations using fine-tuned interior design style models informed by user preferences, Adv. Eng. Inform. 62 (2024) 102686, http://dx.doi.org/10.1016/j.aei.2024.102686.

[26] Y. Wang, J. Zhang, C. Shen, H. Yu, S. Luo, Generative AI aids personalized product aesthetic generation and evaluation based on style themes, Adv. Eng. Inform. 68 (2025) 103756, http://dx.doi.org/10.1016/j.aei.2025.103756.

[27] L. Zhang, Z. Li, Y. Zheng, An interactive generative design technology for appearance diversity–Taking mouse design as an example, Adv. Eng. Inform. 59 (2024) 102263, http://dx.doi.org/10.1016/j.aei.2023.102263.

[28] X. Liu, Z. Yang, L. Gong, M. Liu, X. Xiang, Z. Mo, Intelligent color scheme generation for web interface color design based on knowledge- data fusion method, Adv. Eng. Inform. 65 (2025) 103105, http://dx.doi.org/10.1016/j.aei.2024.103105.

[29] K. Mustapha, A survey of emerging applications of large language models for problems in mechanics, product design, and manufacturing, Adv. Eng. Inform. 64 (2025) 103066, http://dx.doi.org/10.1016/j.aei.2024.103066.

[30] R. Tsumoto, K. Yaji, Y. Nomaguchi, K. Fujita, Deep concept identification for generative design, Adv. Eng. Inform. 65 (2025) 103354, http://dx.doi.org/10.1016/j.aei.2025.103354.

[31] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al., Training language models to follow instructions with human feedback, Adv. Neural Inf. Process. Syst. 35 (2022) 27730–27744.

[32] R. Rafailov, A. Sharma, E. Mitchell, C.D. Manning, S. Ermon, C. Finn, Direct preference optimization: Your language model is secretly a reward model, Adv. Neural Inf. Process. Syst. 36 (2023) 53728–53741.

[33] U. Gadot, R. Gal, Y. Ziser, G. Chechik, S. Mannor, Policy optimized text-to-image pipeline design, 2025, http://dx.doi.org/10.48550/arXiv.2505.21478, arXiv preprint arXiv:2505.21478.

[34] H. Li, X. Cheng, X. Zhang, Accurate insights, trustworthy interactions: Designing a collaborative AI-human multi-agent system with knowledge graph for diagnosis prediction, in: Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems, 2025, pp. 1–15, http://dx.doi.org/10.1145/3706598.3713526.

[35] M. Jiang, J. Gao, Z. Pan, Y. Wu, Z. Wang, NexaNota: An AI-powered smart linked lecture note-taking system leveraging large language models, in: Proceedings of the 2025 International Conference on Big Data and Informatization Education, 2025, pp. 242–248, http://dx.doi.org/10.1145/3729605.3729648.

[36] Y. Lee, J.J.Y. Chung, T.S. Kim, J.Y. Song, J. Kim, Promptiverse: Scalable generation of scaffolding prompts through human-AI hybrid knowledge graph annotation, in: Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems, 2022, pp. 1–18, http://dx.doi.org/10.1145/3491102.3502087.

[37] Q. Zheng, M. Chen, P. Sharma, Y. Tang, M. Oswal, Y. Liu, Y. Huang, EvAlignUX: Advancing UX research through LLM-supported exploration of evaluation metrics, 2024, http://dx.doi.org/10.48550/arXiv.2409.15471, arXiv preprint arXiv:2409.15471.

[38] H. Kaur, D. Downey, A. Singh, E.Y.-Y. Cheng, D. Weld, J. Bragg, FeedLens: Polymorphic lenses for personalizing exploratory search over knowledge graphs, in: Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology, 2022, pp. 1–15, http://dx.doi.org/10.1145/3526113.3545631.

[39] H. Li, G. Appleby, A. Suh, LinkQ: An LLM-assisted visual interface for knowledge graph question-answering, in: 2024 IEEE Visualization and Visual Analytics, VIS, IEEE, 2024, pp. 116–120, http://dx.doi.org/10.1109/VIS55277.2024.00031.

[40] T. Shan, F. Zhang, A.P. Chan, S. Zhu, K. Li, Large language models-empowered automatic knowledge graph development based on multi-modal data for building health resilience, Adv. Eng. Inform. 68 (2025) 103655, http://dx.doi.org/10.1016/j.aei.2025.103655.

[41] Z. Zhang, J. Yu, B. Yang, K. Du, S. Wang, X. Qi, A knowledge graphs construction method enhanced by multimodal large language model for industrial equipment operation and maintenance, Adv. Eng. Inform. 68 (2025) 103705, http://dx.doi.org/10.1016/j.aei.2025.103705.

[42] J. Ling, X. Li, H. Li, Y. An, Y. Rui, Y. Shen, H. Zhu, Hybrid NLP-based extraction method to develop a knowledge graph for rock tunnel support design, Adv. Eng. Inform. 62 (2024) 102725, http://dx.doi.org/10.1016/j.aei.2024.102725.

[43] H. Alberts, T. Huang, Y. Deshpande, Y. Liu, K. Cho, C. Vania, I. Calixto, VisualSem: a high-quality knowledge graph for vision and language, 2020, http://dx.doi.org/10.18653/v1/2021.mrl-1.13, arXiv preprint arXiv:2008.09150.

[44] S.S. Bae, O.-H. Kwon, S. Chandrasegaran, K.-L. Ma, Spinneret: Aiding creative ideation through non-obvious concept associations, in: Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, 2020, pp. 1–13, http://dx.doi.org/10.1145/3313831.3376746.

[45] T.G. Stavropoulos, S. Andreadis, M. Riga, E. Kontopoulos, P. Mitzias, I. Kompatsiaris, A framework for measuring semantic drift in ontologies, in: SEMANTiCS (Posters, Demos, SuCCESS), 2016, URL https://ceur-ws.org/Vol-1695/paper42.pdf.

[46] A. Polleres, R. Pernisch, A. Bonifati, D. Dell'Aglio, D. Dobriy, S. Dumbrava, L. Etcheverry, N. Ferranti, K. Hose, E. Jiménez-Ruiz, et al., How does knowledge evolve in open knowledge graphs? Trans. Graph Data Knowl. 1 (1) (2023) 11:1–11:59, http://dx.doi.org/10.4230/TGDK.1.1.11.

[47] R.M. Bakker, M.H. De Boer, Dynamic knowledge graph evaluation, Authorea Prepr. (2024) http://dx.doi.org/10.36227/techrxiv.171779320.04772689/v2.

[48] Q. Li, Z. Chen, C. Ji, S. Jiang, J. Li, Llm-based multi-level knowledge generation for few-shot knowledge graph completion., in: IJCAI, 2024, pp. 2135–2143, http://dx.doi.org/10.24963/ijcai.2024/236.

[49] J. Liu, M. Zhang, W. Li, C. Wang, S. Li, H. Jiang, S. Jiang, Y. Xiao, Y. Chen, Beyond entities: a large-scale multi-modal knowledge graph with triplet fact grounding, in: Proceedings of the AAAI conference on artificial intelligence, 38, (17) 2024, pp. 18653–18661, http://dx.doi.org/10.1609/aaai.v38i17.29828.

[50] Y. Cheng, Y. Li, N. Zhang, L. Chen, J. Cao, A knowledge graph-enabled multi-domain mapping approach supporting product rapid design: A case study of new energy vehicles, Adv. Eng. Inform. 62 (2024) 102779, http://dx.doi.org/10.1016/j.aei.2024.102779.

[51] V. Sahadevan, R. Joshi, K. Borg, V. Singh, A.R. Singh, B. Muhammed, S.B. Beemaraj, A. Joshi, Knowledge augmented generalizer specializer: A framework for early stage design exploration, Adv. Eng. Inform. 65 (2025) 103141, http://dx.doi.org/10.1016/j.aei.2025.103141.

[52] A. Lucero, Framing, aligning, paradoxing, abstracting, and directing: how design mood boards work, in: Proceedings of the Designing Interactive Systems Conference, 2012, pp. 438–447, http://dx.doi.org/10.1145/2317956.2318021.

[53] D. Mcdonagh, I. Storer, Mood boards as a design catalyst and resource: Researching an under-researched area, Des. J. 7 (3) (2004) 16–31, http://dx.doi.org/10.2752/146069204789338424.

[54] Z. Huang, C. Gao, Y. Shan, H. Hu, Q. Li, X. Deng, C. Ma, Y.-K. Lai, Y.-J. Liu, F. Tian, et al., SketchGPT: A sketch-based multimodal interface for application-agnostic LLM interaction, in: Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology, 2025, pp. 1–18, http://dx.doi.org/10.1145/3746059.3747598.

[55] N. Marquardt, A. Roseway, H. Romat, P. Panda, M. Pahud, G. Ramos, S.M. Drucker, A.D. Wilson, K. Hinckley, N. Riche, ImaginationVellum: Generative-AI ideation canvas with spatial prompts, generative strokes, and ideation history, in: Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology, 2025, pp. 1–19, http://dx.doi.org/10.1145/3746059.3747631.

[56] L. Colusso, M. Densmore, Translation in conversation, Interactions 27 (5) (2020) 26–33, http://dx.doi.org/10.1145/3414221.

[57] A. Endert, P. Fiaux, C. North, Semantic interaction for visual text analytics, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 2012, pp. 473–482, http://dx.doi.org/10.1145/2207676.2207741.

[58] S. Suh, M. Chen, B. Min, T.J.-J. Li, H. Xia, Luminate: Structured generation and exploration of design space with large language models for human-ai co-creation, in: Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems, 2024, pp. 1–26, http://dx.doi.org/10.1145/3613904.3642400.

[59] Y. Kang, Z. Sun, S. Wang, Z. Huang, Z. Wu, X. Ma, MetaMap: Supporting visual metaphor ideation through multi-dimensional example-based exploration, in: Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, 2021, pp. 1–15, http://dx.doi.org/10.1145/3411764.3445325.

[60] H. Shen, L. Shen, W. Wu, K. Zhang, Ideationweb: Tracking the evolution of design ideas in human-ai co-creation, in: Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems, 2025, pp. 1–19, http://dx.doi.org/10.1145/3706598.3713375.

[61] K. Adamkiewicz, P.W. Woźniak, J. Dominiak, A. Romanowski, J. Karolus, S. Frolov, PromptMap: An alternative interaction style for AI-based image generation, in: Proceedings of the 30th International Conference on Intelligent User Interfaces, 2025, pp. 1162–1176, http://dx.doi.org/10.1145/3708359.3712150.

[62] W. Visser, Schön: Design as a reflective practice, Collection (2) (2010) 21–25.

[63] N. Cross, Designerly Ways of Knowing, Springer, 2006.

[64] M. Scaife, Y. Rogers, External cognition: how do graphical representations work? Int. J. Hum.-Comput. Stud. 45 (2) (1996) 185–213, http://dx.doi.org/10.1006/ijhc.1996.0048.

[65] D. Gatis, contributors, rembg: Remove image backgrounds, 2025, https://github.com/danielgatis/rembg. Version 2.0.67, MIT License. (Accessed 01 September 2025).

[66] F. Li, H. Zhang, P. Sun, X. Zou, S. Liu, C. Li, J. Yang, L. Zhang, J. Gao, Segment and recognize anything at any granularity, in: European Conference on Computer Vision, Springer, 2024, pp. 467–484, http://dx.doi.org/10.1007/978-3-031-73195-2_27.

[67] K. Hornbæk, et al., Some whys and hows of experiments in human–computer interaction, Found. Trends® Hum.–Comput. Interact. 5 (4) (2013) 299–373, http://dx.doi.org/10.1561/1100000043.

[68] J. Blijlevens, M.E. Creusen, J.P. Schoormans, et al., How consumers perceive product appearance: The identification of three product appearance attributes, Int. J. Des. 3 (3) (2009) 27–35.

[69] S.B. Sutono, Selection of representative Kansei adjectives using cluster analysis: a case study on car design, Int. J. Adv. Eng. Manag. Sci. 2 (11) (2016) 239691.
[70] S.G. Hart, L.E. Staveland, Development of NASA-TLX (task load index): Results of empirical and theoretical research, in: Advances in Psychology, vol. 52, Elsevier, 1988, pp. 139–183, http://dx.doi.org/10.1016/S0166-4115(08)62386-9.
[71] E. Cherry, C. Latulipe, Quantifying the creativity support of digital tools through the creativity support index, ACM Trans. Comput.-Hum. Interact. (TOCHI) 21 (4) (2014) 1–25, http://dx.doi.org/10.1145/2617588.
[72] V. Venkatesh, H. Bala, Technology acceptance model 3 and a research agenda on interventions, Decis. Sci. 39 (2) (2008) 273–315, http://dx.doi.org/10.1111/j.1540-5915.2008.00192.x.
[73] O. Shaikh, S. Sapkota, S. Rizvi, E. Horvitz, J.S. Park, D. Yang, M.S. Bernstein, Creating general user models from computer use, in: Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology, 2025, pp. 1–23, http://dx.doi.org/10.1145/3746059.3747722.