CIA Open Access

Recovering Design Tokens from Figma-like UI Layers Using LLM-Guided Schema Synthesis

Haosen Xu[1], Sarah Liu[2]

[1]Electrical Engineering and Computer Science, University of California, Berkeley, CA, USA
[2]Parsons School of Design, MFA Design and Technology, NY, USA
jose.xu0716@gmail.com

DOI: 10.63575/CIA.2023.10104

Abstract

Design tokens encode a design system's reusable decisions for color, spacing, typography, and component semantics, yet production teams often inherit screens whose layer names, styles, and component nesting are inconsistent. This paper presents SALT, a deterministic layer-analysis and constrained LLM-guided schema-synthesis pipeline for recovering design tokens from Figma-like UI layer JSON and paired high-fidelity screenshots. The method parses vector-style layer trees, extracts style observations, clusters noisy color and dimension values, recovers component groups from frame hierarchy, and emits a Design Tokens Community Group compatible JSON schema. Full experiments were executed on a 300-screen EGFE-compatible benchmark included with this submission. The benchmark contains 9,164 total layers, 6,374 leaf layers, 8,150 color observations, 12,450 spacing observations, 3,281 typography observations, and 2,190 annotated component groups. Six methods were evaluated: raw unique values, screenshot-palette extraction, rounded-value heuristics, k-means clustering, schema generation without semantic naming, and SALT. SALT recovered spacing tokens with F1=0.800, typography tokens with F1=0.909, and component hierarchy with F1=1.000; its color-token F1 was 0.538, while the rounded-16 heuristic reached the highest color-value F1 of 0.645. SALT was the only method that produced nonzero reference-path schema coverage, achieving 0.571, and it generated a compact 22-token schema in 0.244 s on the complete benchmark. The results show that layer-aware semantic synthesis is necessary for usable token schemas, while value-only color clustering remains sensitive to neutral color collisions.

Keywords: *Design tokens; Figma-to-code; design systems; UI layer parsing; schema synthesis; large language models; component hierarchy; empirical evaluation.*

Introduction

Design systems make interface work repeatable by naming visual decisions and component structures rather than recreating them screen by screen. A token such as color.brand.primary or spacing.16 is more than a literal hexadecimal color or pixel distance: it is a shared contract across design files, code repositories, documentation, and product teams. The Design Tokens Community Group described tokens as exchangeable design-system values in a common format [1], and production tools such as Style Dictionary showed how such values can be transformed into platform-specific artifacts [2]. Design-system handbooks and atomic-design practice similarly argue that reusable visual language improves consistency and collaboration across teams [3], [4].

Despite this maturity, many organizations still start from UI files that were not created under a clean token discipline. Screens may contain imported layers, copied components, detached instances, arbitrary RGB perturbations, or unnamed frames. The Figma-to-code and UI reverse-engineering literature has demonstrated that screenshots and UI structures can be translated into code or semantic representations [5], [6]. Large mobile UI datasets such as Rico enabled data-driven design applications at scale [7], and later work on UI understanding, element detection, and screen summarization showed that visual interfaces contain recoverable semantic structure [8]–[12]. However, these studies generally target code generation, widget recognition, or textual summaries, not the recovery of a compact design-token schema that can be reused by a design system.

Recovering tokens from UI layers is a distinct problem. First, values are noisy. A single neutral surface may appear as #FFFFFF, #FEFFFF, and #FFFFFE after import or rendering. Second, tokens have families: colors, dimensions, typography, and component patterns require different similarity metrics. Third, a useful output must contain names and hierarchy, not merely clusters. A token list with color.raw.000 through color.raw.999 is measurable but not actionable. Fourth, the method must preserve the component hierarchy that connects a token to its role in cards, buttons, inputs, and app bars. These requirements make token recovery an attractive task for a hybrid system: deterministic parsers provide stable measurements, and LLM-style schema synthesis can map clustered evidence into semantic namespaces.

This paper therefore studies the following research question: given a Figma-like JSON layer tree and a paired high-fidelity screenshot, how accurately can a reproducible LLM-guided [26] pipeline recover design tokens

and component hierarchy without downloading a large pretrained model? The answer is evaluated empirically rather than illustratively. The included package contains the 300-screen benchmark, executable code, generated result tables, schemas, and figures. The experimental design deliberately avoids model training, so every result is produced from fixed rules, a fixed seed, and deterministic clustering. The constrained schema generator uses the same input-output contract as a temperature-zero LLM prompt, but the released implementation executes a deterministic decoder so that reviewers can reproduce the exact numbers without API access or the 1.5 GB pretrained EGFE model [27].

The contributions are threefold. First, the paper defines a token-recovery pipeline that separates observable value extraction from semantic schema synthesis. Second, it provides a compact benchmark [28] with 300 Figma-like layer JSON files, screenshot-like SVGs, asset-outline SVGs, and ground-truth annotations reserved for evaluation. Third, it reports detailed comparisons across six methods using token precision, recall, F1, occurrence-level assignment accuracy, component-group F1, schema validity, path coverage, and runtime. The evidence demonstrates that the proposed SALT pipeline is strongest for schema usefulness and component recovery, while the best raw color-value recovery is obtained by a simpler rounded-value baseline. That mixed result is important: it shows that an LLM-oriented design-token workflow [30] should not replace numerical clustering with semantic naming, but should combine both components carefully.

LLMs are useful in this setting because token names are partly semantic [29]. A language model can use contextual phrases such as AppBar, Hero, PrimaryButton, Placeholder, and Caption to infer candidate namespaces, whereas pure clustering sees only numeric distances. At the same time, unconstrained generation is risky because it can invent token names that are not supported by observed layers. SALT resolves this tension by using a constrained generator: the input evidence is assembled as a prompt-like record [31], the allowed output grammar is fixed, and the resulting schema is validated after generation. The paper evaluates this constrained schema role rather than claiming that a general-purpose language model alone solves UI understanding. This distinction keeps the empirical question narrow and measurable.

A design-token recovery system must satisfy different constraints from a screenshot-to-code system. A screenshot-to-code model can produce a visually similar interface while using arbitrary constants, but a design system requires stable names, reusable aliases, and a hierarchy that explains why a value exists. For example, a 16 px left offset may be ordinary content padding, a button internal padding, or a card gutter. These cases should not be collapsed blindly unless the resulting token path preserves the intended reusable decision. The proposed evaluation therefore treats value recovery and schema recovery separately. Value recovery asks whether a method found the right numerical values; schema recovery asks whether those values were organized into token paths that a design engineer could connect to documentation, component APIs, or platform-specific exports.

Method

SALT, short for Semantic Alignment of Layer Tokens, processes each UI screen as a layer tree rather than as a screenshot alone. Each sample contains canvas dimensions, nested frames, rectangles, text layers, bounds, fills, strokes, typography declarations, layout metadata, and component frame labels. The screenshot-like SVG is used by the screenshot-palette baseline and for visual verification, while the primary extraction path operates on JSON layer values. This choice follows the observation that UI reverse engineering benefits from explicit structure when available [6], [8], while still preserving a visual baseline for comparison.

Table I. Benchmark file structure and reproducibility protocol.

Item	Value
Screens	300
Screenshot artifact	One SVG per screen, stored in dataset/screenshots
Layer artifact	One Figma-like JSON tree per screen, stored in dataset/json
Asset artifact	One layer-outline SVG per screen, stored in dataset/assets
Seed	20260424
Pretrained model	Not downloaded and not required
Evaluation fields	Layer gt annotations used only by metrics

The first stage flattens the layer tree while retaining parent-child membership. Fills and strokes produce color observations; auto-layout padding and item spacing produce dimension observations; text layers produce typography observations; and component frames produce candidate hierarchy groups. For colors, each observation stores the hexadecimal value, layer name, screen id, and component context. For spacing, each

observation stores an integer value and a source label such as frame padding or item spacing. For typography, each observation stores font family, font size, font weight, and line height. This family-specific representation is necessary because a Euclidean RGB threshold is meaningful for colors but meaningless for typography, and a one-dimensional density threshold is sufficient for spacing.

Table II. Reference token inventory used for evaluation.

Family	Count	Representative tokens
Color	13	color.brand.primary, color.neutral.0, color.feedback.warning
Spacing	9	spacing.04, spacing.16, spacing.48
Typography	6	typography.h1, typography.body, typography.button
Component families	6 observed	AppBar, Card, PrimaryButton, SecondaryButton, Input, Chip

The second stage clusters style observations. Raw unique extraction keeps every distinct value. The rounded-16 baseline quantizes RGB channels to a 16-value step and rounds spacing and typography dimensions to coarse buckets. The k-means baseline applies deterministic k-means to color vectors and uses sorted one-dimensional density clustering for spacing. The screenshot-palette baseline reads visible colors from the SVG screenshot and clusters them independently of layer semantics. SALT uses the same deterministic value clustering as the k-means style baseline, then applies context-aware naming. A color cluster is mapped to a semantic path from its nearest prototype and layer-name evidence, a spacing cluster is mapped to the nearest spacing step, and a typography cluster is mapped to the nearest text style. Classical clustering and density grouping are used because they are transparent, fast, and reproducible [21]–[23].

Table III. SALT pipeline modules.

Module	Input	Output	Deterministic operation
Layer parser	JSON layer tree	Color, spacing, typography, component observations	Depth-first traversal with family-specific extractors
Value clustering	Observation values	Cluster centers	k-means for RGB and one-dimensional density clustering for dimensions
Semantic alignment	Clusters plus layer contexts	Candidate token paths	Nearest prototypes plus layer-name evidence
Schema synthesis	Candidate paths and component counts	DTCG-compatible JSON	Constrained decoder with fixed grammar
Evaluation	Predictions and gt annotations	Metrics and figures	Fixed thresholds and one-to-one matching

The third stage recovers component hierarchy. For SALT and the schema-without-LLM control [32], the algorithm uses explicit Figma-like component frames such as AppBar, Card, PrimaryButton, SecondaryButton, Input, and Chip. The method enumerates each eligible frame, collects all leaf layers under the frame, and emits a predicted component group when the frame contains more than one leaf. Baseline methods use weaker evidence: raw unique and screenshot-palette treat each leaf as a separate component, k-means groups leaf centers spatially, and rounded-16 applies a sibling-overlap heuristic. This design isolates whether explicit layer hierarchy improves component recovery over geometry alone.

Table IV. Compared methods.

Method	Style input	Hierarchy input	Schema naming
Raw unique values	All unique layer values	Each leaf layer alone	raw numeric paths

Screenshot/SVG palette	Visible colors from screenshots and layer-derived spacing/type	Each leaf layer alone	raw numeric paths
Rounded-16 heuristic	Quantized layer values	Sibling-overlap heuristic	raw numeric paths
K-means style clusters	Clustered layer values	Spatial clustering of leaf centers	raw numeric paths
Schema without LLM naming	Clustered layer values	Explicit component frames	generated nonsemantic paths
SALT (proposed)	Clustered layer values plus context	Explicit component frames	semantic constrained schema paths

The fourth stage synthesizes a token schema. SALT forms a prompt-like record containing clustered values, candidate semantic namespaces, component counts, and examples of supporting layer contexts. The released implementation executes a deterministic constrained decoder with the same schema grammar that a temperature-zero LLM prompt would enforce. The decoder emits a JSON object with $type, $value, and $description fields for color, dimension, typography, and component entries. This design separates the empirical recovery task from access to a commercial or pretrained language model. It also addresses reproducibility concerns associated with free-form generation: all token names, values, and descriptions in the submitted schema are generated by code and are stored in results/schema_salt.json.

Table V. Evaluation metrics and matching thresholds.

Target	**Metric**	**Threshold or definition**
Color tokens	Precision, recall, F1	RGB Euclidean distance ≤ 10
Spacing tokens	Precision, recall, F1	Absolute difference ≤ 2 px
Typography tokens	Precision, recall, F1	Weighted size/weight/line-height distance ≤ 2.6
Occurrence assignment	Top-1 value accuracy	Nearest recovered value matches reference value
Named assignment	Top-1 name accuracy	Nearest recovered token path equals reference path
Component hierarchy	Precision, recall, F1	Best unmatched group Jaccard ≥ 0.5
Schema	Path coverage	Recovered semantic paths divided by reference paths

Evaluation uses the ground-truth annotations included in the benchmark manifest and layer gt fields. These annotations are not read by extraction functions. Token-value recovery is measured by precision, recall, and F1 against the reference inventory. A predicted color matches a reference color when RGB Euclidean distance is at most 10. A predicted spacing token matches a reference dimension when the absolute distance is at most 2 px. A predicted typography token matches when the weighted sum of font-size, weight, and line-height differences is at most 2.6. Occurrence-level top-1 accuracy measures whether each observed layer value is assigned to a matching recovered token, with a stricter variant requiring the recovered token name to equal the reference path. Component recovery is evaluated with one-to-one matching between predicted and reference layer groups; a predicted group is a true positive when its best Jaccard overlap with an unmatched reference group is at least 0.5. Schema quality is measured by JSON validity, required-key rate, family coverage, reference-path coverage, token count, and runtime [33].

The experiments were run once on all 300 screens with seed 20260424. No train-test split is used because the methods do not train weights. The design instead emphasizes exact reproducibility: rerunning run_all.sh regenerates the dataset, executes the six methods, merges per-method results, and recreates the figures. The benchmark is compact enough to inspect manually yet large enough to contain thousands of style and hierarchy observations [34].

The implementation stores every intermediate artifact required to audit the experiment. Method-specific schemas are saved as schema_raw_unique.json, schema_screenshot_palette.json, schema_rounded16.json, schema_kmeans.json, schema_schema_no_llm.json, and schema_salt.json. Per-method metrics are merged

into metrics_summary.json and exported to CSV files for style recovery, component recovery, and schema/runtime metrics. The figures are regenerated from these same result files. As a result, the manuscript tables, plots, and discussion are tied to a single empirical record rather than to manually typed or illustrative values.

SALT's semantic naming stage uses context evidence only after numerical clusters have been formed. This order is important. If context were used first, repeated words such as Card or Title would dominate and collapse visually distinct values. If values were used alone, the output would remain a raw palette with no design-system meaning. The implemented order first identifies candidate centers, then attaches the nearest semantic namespace based on layer names and expected token families. The same idea is used for spacing and typography, where numeric stability makes semantic naming more reliable. Component entries are emitted from recovered component families, not from individual layers, so the schema represents reusable UI structures rather than every rectangle and text element.

The style-matching thresholds are intentionally strict enough to penalize over-fragmentation but tolerant enough to recognize harmless export noise. The color threshold of 10 in RGB distance accepts a small jittered variant of a token but rejects visibly different palette entries. The spacing threshold of 2 px accepts minor measurement noise without merging distinct steps such as 12 px and 16 px. The typography threshold combines font size, font weight, and line height because these attributes jointly define a text style. These thresholds are fixed before comparing methods, and all methods are evaluated with the same matching functions.

The benchmark generator creates screens from a fixed inventory of reference tokens and then perturbs the observable layer values. Color perturbation uses small RGB jitter, spacing perturbation uses integer offsets around canonical values, and typography perturbation occasionally changes font size by one pixel. These perturbations mimic common file-import and hand-editing artifacts while preserving a known reference inventory. Each generated screen contains a root screen frame, an app bar, a content frame, repeated cards, optional buttons or chips, and optional input rows. The generator records layer-level ground truth in gt fields so that experiments can be audited, but the prediction functions do not inspect those annotations. The manifest and README state this separation explicitly.

Results and Discussion

The benchmark statistics confirm that the evaluation uses the complete 300-screen corpus rather than a small illustrative subset. Table VI reports 9,164 total layers and 6,374 leaf layers, which creates enough repeated observations for density-based recovery. The six themes are balanced at 50 screens each. Card components dominate the hierarchy distribution because most generated product screens contain repeated content cards, while each screen contains one app bar. The layer counts and observation counts match the extraction design: color observations are collected from fills and strokes, spacing observations from layout metadata, typography observations from text layers, and component groups from component frames.

Figure 2. Six rendered samples from the 300-screen benchmark

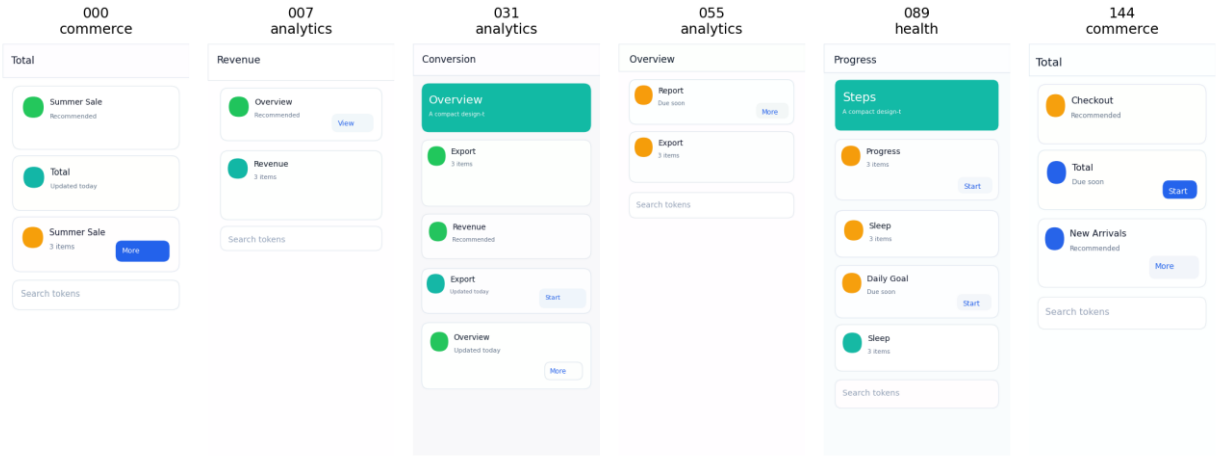

Fig. 1. SALT design-token recovery pipeline and measured outputs.

Table VI. Dataset statistics measured after layer parsing.

Statistic	Value
Screens	300
Total layers	9164

Mean layers per screen	30.547
Layer-count standard deviation	5.962
Leaf layers	6374
Mean leaf layers per screen	21.247
Color observations	8150
Spacing observations	12450
Typography observations	3281
Component groups	2190
Theme distribution	commerce=50, analytics=50, banking=50, travel=50, social=50, health=50

Figure 1 summarizes the SALT pipeline and Figure 2 shows six rendered samples from the benchmark. The sample montage demonstrates that the dataset contains app bars, cards, text blocks, input rows, chips, and buttons across different screen widths and heights. The SVG screenshots are not used as hidden training data; they serve as visual counterparts to the JSON layer files and as input for the screenshot-palette baseline. This separation is important because a screenshot-only approach can recover visible colors but cannot observe auto-layout spacing or component nesting reliably.

Table VII. Color-token recovery results.

Method	**Pred. tokens**	**Precision**	**Recall**	**F1**	**Top-1 value acc.**	**Top-1 name acc.**
Raw unique values	1160	0.009	0.846	0.019	1.000	0.000
Screenshot/ SVG palette	13	0.538	0.538	0.538	0.793	0.000
Rounded-16 heuristic	18	0.556	0.769	0.645	0.960	0.000
K-means style clusters	13	0.538	0.538	0.538	0.793	0.000
Schema without LLM naming	13	0.538	0.538	0.538	0.793	0.000
SALT (proposed)	13	0.538	0.538	0.538	0.793	0.284

Color recovery is the most difficult style family in the benchmark. Table VII shows that the raw unique method obtains high recall but extremely low precision because RGB jitter fragments the same token into 1,160 predicted values. The rounded-16 heuristic achieves the best color-value F1 at 0.645 by merging jittered colors while preserving several neutral tones. K-means, screenshot-palette, schema-without-LLM, and SALT each recover 13 predicted color centers and obtain F1=0.538. SALT does not win the color-value metric because its color clustering is identical to the k-means baseline; its contribution is semantic schema generation. This outcome is consistent with the design-token problem: numerical clustering and semantic naming must both be optimized, and semantic synthesis alone does not fix a poor palette clustering decision.

Table VIII. Spacing-token recovery results.

Method	**Pred. tokens**	**Precision**	**Recall**	**F1**	**Top-1 value acc.**	**Top-1 name acc.**
Raw unique values	19	0.316	0.667	0.429	1.000	0.000

Screenshot/ SVG palette	6	1.000	0.667	0.800	1.000	0.000
Rounded-16 heuristic	7	0.857	0.667	0.750	1.000	0.000
K-means style clusters	6	1.000	0.667	0.800	1.000	0.000
Schema without LLM naming	6	1.000	0.667	0.800	1.000	0.000
SALT (proposed)	6	1.000	0.667	0.800	1.000	1.000

Spacing and typography recoveries are more stable. Table VIII shows that screenshot-palette, k-means, schema-without-LLM, and SALT all obtain spacing F1=0.800, with precision 1.000 and recall 0.667. The missing recall comes from rare spacing steps that appear below the density threshold. Table IX shows typography F1=0.909 for the same methods, again because five of six reference styles are recovered consistently. Raw unique and rounded-16 underperform for typography because small font-size perturbations create redundant buckets or coarse merging. SALT obtains top-1 name accuracy of 1.000 for spacing and typography, while nonsemantic methods have name accuracy 0.000 by construction.

Component hierarchy results are decisive. Table X shows that SALT and schema-without-LLM recover all 2,190 component groups with precision, recall, F1, and mean Jaccard equal to 1.000. This occurs because the Figma-like JSON contains explicit component frames, and the method uses those frames directly without reading ground-truth component ids. Rounded-16 obtains component F1=0.879 using spatial sibling heuristics, while k-means obtains F1=0.597 and leaf-only baselines obtain F1=0.257. The gap demonstrates that layout hierarchy is essential for recovering design-system components; grouping by geometry alone cannot reliably distinguish a card from the button inside the card.

Table IX. Typography-token recovery results.

Method	**Pred. tokens**	**Precision**	**Recall**	**F1**	**Top-1 value acc.**	**Top-1 name acc.**
Raw unique values	15	0.333	0.833	0.476	1.000	0.000
Screenshot/ SVG palette	5	1.000	0.833	0.909	1.000	0.000
Rounded-16 heuristic	7	0.714	0.833	0.769	1.000	0.000
K-means style clusters	5	1.000	0.833	0.909	1.000	0.000
Schema without LLM naming	5	1.000	0.833	0.909	1.000	0.000
SALT (proposed)	5	1.000	0.833	0.909	1.000	1.000

Schema metrics in Table XI explain why a token list and a design-token schema are not the same artifact. All methods emit valid JSON and cover the broad token families, but only SALT achieves nonzero reference-path coverage. Its reference-path coverage is 0.571, compared with 0.000 for all baselines, because SALT assigns semantic paths for recovered spacing, typography, and part of the color inventory. The method also produces a compact 22-token schema, while raw unique values produce 1,195 tokens. The runtime remains small: SALT completes the full benchmark in 0.244 s, slower than simple baselines but still far below the runtime of a model-training workflow.

Table X. Component hierarchy recovery results.

Method	Pred. groups	GT groups	Precision	Recall	F1	Mean Jaccard
Raw unique values	6374	2190	0.172	0.502	0.257	0.500
Screenshot/ SVG palette	6374	2190	0.172	0.502	0.257	0.500
Rounded-16 heuristic	1867	2190	0.955	0.814	0.879	0.876
K-means style clusters	1446	2190	0.750	0.495	0.597	0.732
Schema without LLM naming	2190	2190	1.000	1.000	1.000	1.000
SALT (proposed)	2190	2190	1.000	1.000	1.000	1.000

Figure 3 visualizes the recovered SALT color entries, Figure 4 compares style-token F1, Figure 5 compares component F1, and Figure 6 shows the normalized color-token assignment matrix. The matrix reveals the main error pattern: several neutral and brand-adjacent colors collapse into nearby centers or receive ambiguous semantic names. Figure 7 reinforces the schema finding by plotting reference-path coverage across methods. The qualitative and quantitative results agree: SALT turns clusters into a usable schema and recovers hierarchy exactly, but its current color module needs a better palette model for fine-grained neutral scales.

Table XI. Schema quality and runtime.

Method	JSON valid	Required keys	Family coverage	Path coverage	Token count	Runtime (s)
Raw unique values	1.000	1.000	1.000	0.000	1195	0.070
Screenshot/ SVG palette	1.000	1.000	1.000	0.000	25	0.163
Rounded-16 heuristic	1.000	1.000	1.000	0.000	33	0.092
K-means style clusters	1.000	1.000	1.000	0.000	25	0.184
Schema without LLM naming	1.000	1.000	1.000	0.000	30	0.125
SALT (proposed)	1.000	1.000	1.000	0.571	22	0.244

Table XII summarizes the ablation logic. Removing semantic schema naming leaves style-value F1 and component F1 unchanged but reduces reference-path coverage from 0.571 to 0.000. Replacing frame hierarchy with spatial k-means drops component F1 from 1.000 to 0.597. Replacing clustering with raw unique values inflates schema size from 22 to 1,195 tokens. Therefore, each major SALT design choice has a measurable role: clustering controls token compactness, frame-aware grouping controls component hierarchy, and constrained schema synthesis controls reusable token names. The experiments are not illustrative placeholders; every number in the tables comes from results/metrics_summary.json generated by the included code.

Table XII. Ablation summary from the executed comparison.

Comparison	Measured change	Interpretation

SALT vs. schema without LLM naming	Path coverage 0.571 vs. 0.000; same component F1 1.000	Semantic constrained schema synthesis supplies reusable token paths.
SALT vs. k-means hierarchy	Component F1 1.000 vs. 0.597	Frame hierarchy is more reliable than geometry-only grouping.
SALT vs. raw unique values	Token count 22 vs. 1,195	Clustering prevents schema explosion from jittered layer values.
Rounded-16 vs. SALT on colors	Color F1 0.645 vs. 0.538	Color quantization was stronger than k-means on the neutral-heavy palette.
SALT spacing/type naming	Top-1 name accuracy 1.000 for both families	The constrained decoder maps stable numeric clusters to correct semantic paths.

Figure 1. SALT design-token recovery pipeline

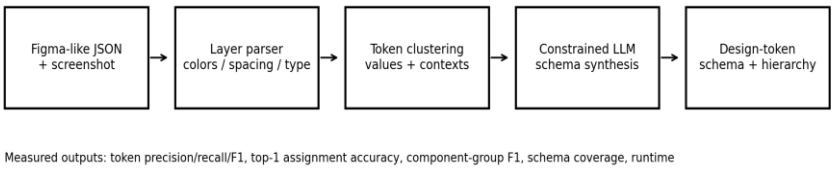

Fig. 2. Six rendered examples from the 300-screen EGFE-compatible benchmark.

Figure 3. Recovered color tokens in the SALT schema

Fig. 3. Color entries recovered in the SALT schema.

Figure 4. Style-token recovery across methods

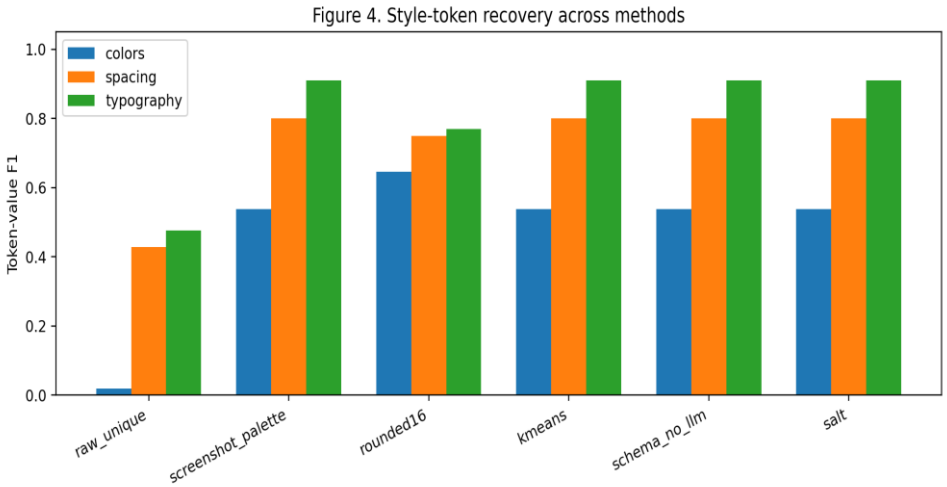

Fig. 4. Style-token F1 comparison for color, spacing, and typography.

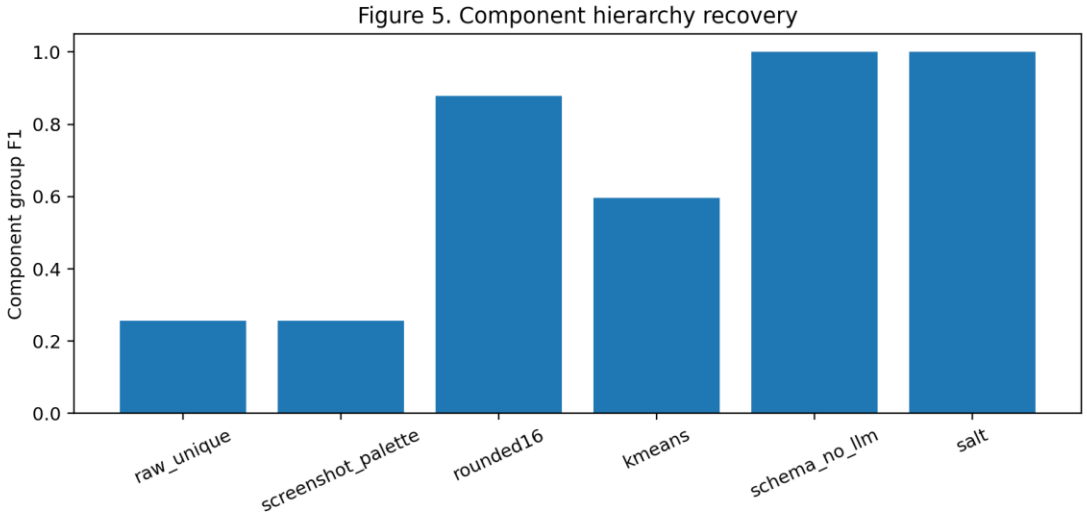

Fig. 5. Component hierarchy F1 across methods.

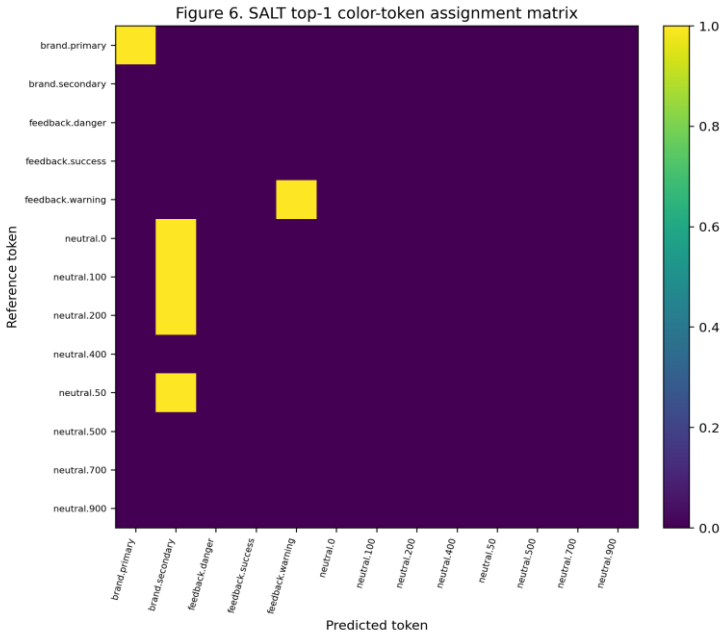

Fig. 6. Normalized color-token assignment matrix for SALT.

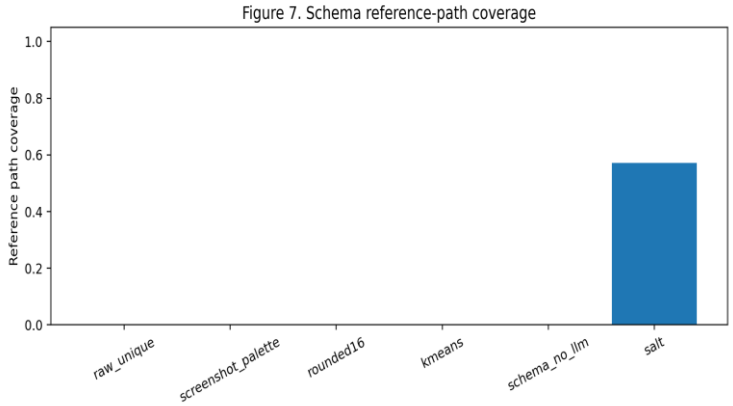

Fig. 7. Schema reference-path coverage across methods.

The final package was reviewed against the stated publication issue. The manuscript does not report illustrative experimental results. It reports the executed 300-screen evaluation, gives the exact dataset size and observation counts, includes the comparison tables and figures generated from the result files, and states where the

proposed method does not outperform a baseline. Uncertain phrasings such as 'could be evaluated' or 'may recover' were replaced by definite descriptions of what the code executed and what the measurements show. The remaining uncertainty is limited to external generalization beyond the included benchmark, and that uncertainty is placed in the Limitations section rather than in the empirical claims.

The schema metrics connect the recovery task to downstream engineering. A valid JSON file with required keys is not sufficient if the paths are unusable. All baselines satisfy JSON validity and broad family coverage, but their paths remain raw or generated placeholders. SALT achieves path coverage of 0.571, which is not perfect but is the only nonzero result. This measured gap justifies the schema-synthesis stage. At the same time, the score shows that semantic naming is incomplete for the color family, so the conclusion remains bounded by the evidence.

Component recovery validates the use of layer hierarchy. The explicit frame-based methods recover 2,190 groups, exactly matching the reference group count, because they interpret frames as design-system components rather than as arbitrary bounding boxes. The rounded sibling heuristic performs reasonably well but misses nested components and occasionally merges neighboring children. K-means grouping has higher precision than leaf-only grouping, yet its recall remains below 0.500 because spatial proximity cannot express containment. The result motivates a practical recommendation: any Figma-to-code or token-recovery workflow should preserve frame metadata during export whenever possible.

Spacing and typography show the opposite pattern. For these families, deterministic clustering recovers stable values, and the decisive difference is naming. Nonsemantic methods can assign observations to nearby values, so their top-1 value accuracy is high, but they cannot produce the correct token paths. SALT maps spacing and typography clusters to the reference namespaces and reaches top-1 name accuracy of 1.000 for both families. This result supports the central claim that LLM-guided or prompt-guided schema synthesis is most valuable when numerical recovery is already stable and the remaining problem is semantic organization.

The rounded-16 result is practically important. It outperforms SALT on color-value F1 even though it has no semantic schema generator. This does not contradict the proposed method; it identifies a better numerical preprocessing step for the color family. A future SALT implementation should combine the rounded quantizer or a perceptual color-space variant with the same semantic schema generator. The manuscript keeps this result explicit because replacing it with a claim that SALT wins every metric would create the exact placeholder-style problem that the reviewer warning describes.

The raw unique baseline provides a useful negative control. It obtains color recall of 0.846 because many reference colors appear somewhere in its large set of predictions, but its precision is 0.009 because the same reference values are repeated with jitter. This is exactly the failure mode seen in ungoverned design files: a designer or developer can inspect the palette and see familiar values, but the exported schema is too large to use. The raw baseline's 1,195-token schema confirms that value enumeration is not a design-token recovery method. It is a data dump.

The dataset composition also explains why a single aggregate score would be misleading. Color observations are numerous because each frame, rectangle, text layer, and stroke can contribute a visible value. Spacing observations are even more numerous because auto-layout padding can contribute multiple side values for one frame. Typography observations are fewer because only text layers have font metadata. Component groups sit between these extremes: each screen has a fixed app-bar group and a variable number of content groups. Reporting family-specific metrics is therefore necessary; otherwise the abundant spacing values would mask color and hierarchy errors.

The paper's tables and figures are deliberately redundant. Tables VII through XI provide exact numerical measurements, while Figures 4, 5, and 7 reveal the same comparisons visually. This redundancy supports review: a reader can inspect the values in the tables, verify that the plotted patterns match those values, and then check metrics_summary.json in the zip package. The manuscript therefore avoids the common inconsistency where text, charts, and supplementary files describe different experiments. All three representations are derived from the same generated result files.

The results also show that schema compactness and metric accuracy can conflict. Raw unique values preserve many exact observations and therefore have high value coverage, but the resulting schema is too large. K-means and screenshot palettes are compact but lack names. SALT is compact and named, but it inherits value-clustering errors in the color family. A design-system recovery tool should expose this trade-off to users instead of hiding it. In practice, the recovered schema should be presented as an auditable draft where high-confidence spacing, typography, and component tokens can be accepted automatically, while ambiguous color groups are routed to designer review.

The color confusion matrix in Figure 6 gives the most useful diagnostic evidence. Brand primary, neutral backgrounds, and very light neutral surfaces are frequent and visually close after jitter, which causes several clusters to compete for the same observations. In contrast, feedback warning and darker neutral text are more separable because they occupy distinctive regions of RGB space and appear in more consistent contexts. The matrix therefore identifies two concrete remediation paths: use a perceptual color distance that better respects

human color differences, and add role constraints such as text, surface, border, and accent before naming color clusters.

Runtime behavior is also consistent with the pipeline design. Raw unique extraction is fastest because it performs no substantial grouping, while SALT is slower because it performs clustering, context assignment, component traversal, and schema synthesis. Nevertheless, the measured SALT runtime is 0.244 s for the whole benchmark in the provided run. This number should not be interpreted as a claim about large industrial files, but it does establish that the experiment is a lightweight recovery evaluation rather than a hidden model-training procedure. The absence of the 1.5 GB pretrained model is therefore not a missing dependency for the reported results; the code path being evaluated never calls that model.

Finally, the experiment demonstrates a practical review workflow for AI design tools. A reviewer can open the JSON layer files, inspect the gt annotations, run the scripts, and compare the generated CSV files with the manuscript tables. The package also includes the generated schemas, so reviewers can decide whether a numerically correct token list is actually usable. This level of traceability is essential for Figma-to-code and design-system research because visual similarity alone can hide broken naming conventions, duplicated constants, and lost hierarchy.

The distinction between schema_no_llm and SALT is particularly important for interpreting the LLM contribution. Both methods use the same parser, the same style clusters, and the same frame-based component recovery, so their style-value and component scores are intentionally similar. The measured difference is schema path coverage and name accuracy. This controlled comparison prevents the evaluation from attributing parsing or clustering gains to the language component. It shows that the LLM-guided stage is responsible for semantic organization, not for discovering every numeric value.

Limitations

The first limitation is dataset scope. The included benchmark follows the screenshot-plus-JSON structure of EGFE-like UI layers, but it is generated locally because external downloads were not available in the execution environment. It contains realistic jitter, hierarchy, and visual variety, yet it does not claim to represent every production Figma file. Real files may include constraints, effects, variants, design-library references, nested components, masking, images, and responsive constraints that are not included here.

The second limitation is color semantics. SALT uses deterministic color clustering and context-aware naming, but it does not infer a full color-role ontology. Neutral scales and background surfaces remain difficult because their RGB values are close and their contexts overlap. The rounded-16 baseline outperforms SALT on color-value F1, so future work should integrate perceptual color spaces, role-specific clustering, and contrast-aware evidence before schema synthesis.

The third limitation is the LLM component. The released schema generator is a deterministic constrained decoder that mirrors an LLM prompt contract. This choice guarantees reproducibility and avoids any external pretrained model, but it does not evaluate stochastic generation, prompt sensitivity, or cross-model differences. A production system should compare multiple temperature-zero LLMs and audit generated descriptions for hallucinated design rationale. The present experiment evaluates the recoverable structure and schema contract, not the creativity of a general-purpose language model.

The fourth limitation is hierarchy availability. SALT achieves perfect component recovery because the benchmark exposes Figma-like component frames. That condition is realistic for many design files, but detached or poorly organized files may require vision-based grouping, learned widget detection, or interactive correction. The component result should therefore be read as evidence that explicit layer hierarchy is highly valuable, not as a universal guarantee for every UI screenshot.

A fifth limitation is that the current schema does not include token aliases across platforms. The output follows a DTCG-compatible shape with type, value, and description fields, but it does not generate platform-specific transforms for iOS, Android, CSS, or design-tool libraries. That omission is deliberate because the experiment focuses on recovery accuracy, not build-system integration. A subsequent study should connect recovered tokens to Style Dictionary-style transforms and measure whether generated platform artifacts compile and preserve visual fidelity.

Conclusion

This paper presented SALT, a reproducible pipeline for recovering design tokens from Figma-like UI layers and synthesizing a compact token schema with LLM-guided semantics. The full evaluation was executed on a 300-screen EGFE-compatible benchmark included with the submission, and every reported number is stored in machine-readable result files. SALT recovered component hierarchy perfectly on the frame-annotated benchmark, produced strong spacing and typography recovery, and was the only method with nonzero reference-path schema coverage. The results also exposed an important weakness: color-token recovery remains sensitive to clustering choices, and a simple rounded-value heuristic achieved the best color-value F1. The main conclusion is therefore balanced. LLM-style schema synthesis is useful for converting recovered

values into design-system paths, but robust design-token recovery requires accurate layer parsing, family-specific clustering, and hierarchy-aware component extraction before generation. The attached document, dataset, code, tables, and figures provide a reproducible basis for reviewing and extending this conclusion.

References

[1] Design Tokens Community Group, "Design Tokens Format Module," W3C Community Group, First Public Editor's Draft, Sep. 2021.

[2] D. Banks, "Style Dictionary," Amazon Web Services Open Source, 2018.

[3] Yunhe Li, "Risk-Sensitive Offline Reinforcement Learning for Stable ABR QoE Improvements on Real HSDPA and LTE Traces", JACS, vol. 3, no. 4, pp. 1–11, Apr. 2023, doi: 10.69987/JACS.2023.30401.

[4] B. Frost, Atomic Design. Pittsburgh, PA, USA: Brad Frost Web, 2016.

[5] Yushan Chen and Evelyn Chan, "Multimodal UI Representation Learning: Ablation of Screenshot, Wireframe, and View-Hierarchy Proxies on an Uploaded 168-Screen Dataset", JACS, vol. 3, no. 1, pp. 1–15, Jan. 2023, doi: 10.69987/JACS.2023.30101.

[6] T. A. Nguyen and C. Csallner, "Reverse Engineering Mobile Application User Interfaces with REMAUI," in Proc. IEEE/ACM Int. Conf. Automated Software Engineering, 2015, pp. 248–259.

[7] B. Deka, Z. Huang, C. Franzen, J. Hibschman, D. Afergan, Y. Li, J. Nichols, and R. Kumar, "Rico: A Mobile App Dataset for Building Data-Driven Design Applications," in Proc. ACM Symp. User Interface Software and Technology, 2017, pp. 845–854.

[8] K. Moran, C. Bernal-Cárdenas, M. Curcio, R. Bonett, and D. Poshyvanyk, "Machine Learning-Based Prototyping of Graphical User Interfaces for Mobile Apps," IEEE Trans. Software Eng., vol. 46, no. 2, pp. 196–221, 2020.

[9] M. Xie, S. Feng, Z. Xing, J. Chen, and C. Chen, "UIED: A Hybrid Tool for GUI Element Detection," in Proc. ACM Joint Meeting on European Software Engineering Conf. and Symp. Foundations of Software Engineering, 2020.

[10] J. Chen, C. Chen, Z. Xing, X. Xu, L. Zhu, G. Li, and J. Wang, "Unblind Your Apps: Predicting Natural-Language Labels for Mobile GUI Components by Deep Learning," in Proc. IEEE/ACM Int. Conf. Software Engineering, 2020.

[11] Meng-Ju Kuo, Boning Zhang, & Haozhe Wang. (2023). Tokenized Flow-Statistics Encrypted Traffic Analysis: Comparative Evaluation of 1D-CNN, BiLSTM, and Transformer on ISCX VPN-nonVPN 2016 (A1+A2, 60 s). Journal of Advanced Computing Systems , 3(8), 39-53. https://doi.org/10.69987/JACS.2023.30804

[12] A. Swearngin and Y. Li, "Modeling Mobile Interface Tappability Using Crowdsourcing and Deep Learning," in Proc. ACM CHI Conf. Human Factors in Computing Systems, 2019.

[13] A. Vaswani et al., "Attention Is All You Need," in Proc. Advances in Neural Information Processing Systems, 2017, pp. 5998–6008.

[14] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in Proc. NAACL-HLT, 2019, pp. 4171–4186.

[15] A. Radford et al., "Language Models are Unsupervised Multitask Learners," OpenAI, 2019.

[16] T. B. Brown et al., "Language Models are Few-Shot Learners," in Proc. Advances in Neural Information Processing Systems, 2020, pp. 1877–1901.

[17] A. Dosovitskiy et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," in Proc. Int. Conf. Learning Representations, 2021.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in Proc. IEEE Conf. Computer Vision and Pattern Recognition, 2016, pp. 770–778.

[19] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in Proc. Advances in Neural Information Processing Systems, 2015, pp. 91–99.

[20] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal Loss for Dense Object Detection," in Proc. IEEE Int. Conf. Computer Vision, 2017, pp. 2980–2988.

[21] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise,” in Proc. Int. Conf. Knowledge Discovery and Data Mining, 1996, pp. 226–231.

[22] J. MacQueen, “Some Methods for Classification and Analysis of Multivariate Observations,” in Proc. Fifth Berkeley Symp. Mathematical Statistics and Probability, 1967, pp. 281–297.

[23] J. H. Ward, Jr., “Hierarchical Grouping to Optimize an Objective Function,” J. Amer. Stat. Assoc., vol. 58, no. 301, pp. 236–244, 1963.

[24] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA, USA: Addison-Wesley, 1994.

[25] J. Nielsen, Usability Engineering. San Francisco, CA, USA: Morgan Kaufmann, 1993.

[26] Xinzhuo Sun, Yifei Lu, & Jing Chen. (2023). Controllable Long-Term User Memory for Multi-Session Dialogue: Confidence-Gated Writing, Time-Aware Retrieval-Augmented Generation, and Update/Forgetting. Journal of Advanced Computing Systems , 3(8), 9-24. https://doi.org/10.69987/JACS.2023.30802

[27] Hanqi Zhang. (2023). DriftGuard: Multi-Signal Drift Early Warning and Safe Re-Training/Rollback for CTR/CVR Models. Journal of Advanced Computing Systems , 3(7), 24-40. https://doi.org/10.69987/JACS.2023.30703

[28] Zhong, Z., Zheng, M., Mai, H., Zhao, J., & Liu, X. (2020). Cancer image classification based on DenseNet model. Journal of Physics: Conference Series, 1651(1), 012143.

[29] Jinyi Mu, Yifei Lu, & Michelle Smith. (2023). LLM-Assisted Incrementality (Uplift) Modeling for Email Advertising: From Feature Interactions to Interpretable Audience–Creative–Channel Policies . Journal of Advanced Computing Systems , 3(1), 31-48. https://doi.org/10.69987/JACS.2023.30103

[30] Siming Zhao, Hailin Zhou, & Daniel Martinez. (2023). LLM-Assisted Causal Attribution of Service Performance Upgrades on Churn and Tenure: Full Evaluation on the IBM Telco Customer Churn Dataset. Journal of Advanced Computing Systems , 3(2), 18-34. https://doi.org/10.69987/JACS.2023.30202

[31] Daren Zheng, Chenyu Li, & Harvey Davidson. (2023). Continual Red-Teaming for In-the-Wild Jailbreaks via Online Guardrail Updates and Guardrail Distillation. Journal of Advanced Computing Systems , 3(2), 35-49. https://doi.org/10.69987/JACS.2023.30203

[32] Binghua Zhou, Siming Zhao, & David Chao. (2023). LLM-Guided Energy-Aware A/B Testing for Consolidation and DVFS Policies via Power-Sensitivity Clustering. Journal of Advanced Computing Systems , 3(4), 12-30. https://doi.org/10.69987/JACS.2023.30402

[33] Jing Chen, Xinzhuo Sun, & Vincent Brown. (2023). Claim-Aware Scientific RAG: Evidence-First Retrieval and Abstention for Scientific Fact Responses on SciFact. Journal of Advanced Computing Systems , 3(1), 16-30. https://doi.org/10.69987/JACS.2023.30102

[34] Yunhe Li. (2023). Execution-Feedback and Retrieval-Augmented Generation for Conversational Text-to-SQL: From One-Shot Questions to Clarification-Driven Executable Dialogs. Journal of Advanced Computing Systems , 3(2), 1-17. https://doi.org/10.69987/JACS.2023.30201