Journal of Technology Informatics and Engineering (JTIE)
Vol. 4 No. 2 August 2025
E-ISSN: 2961-9068; P-ISSN: 2961-8215, Pages 364-384
DOI: https://doi.org/10.51903/jtie.v4i2.490

From Hand-Drawn Sketches to Interactive Web Prototypes: A Reproducible Constrained Sketch-to-HTML Pipeline With Structural and Visual Consistency Evaluation

Yushan Chen*[1], Maoxi Li[2]
Email: yushanchen1029@gmail.com (1)
[1]*Service Design, Savannah College of Art and Design, GA, USA*
[2]*Business Analytics, Fordham University, NY, USA*
*Corresponding Author

Abstract
Service design workflows often begin with low-fidelity sketches that must be quickly translated into interactive prototypes. This paper studies the Sketch-to-Web problem: generating HTML/CSS prototypes from hand-drawn UI sketches and evaluating fidelity with both structural and visual metrics. Because the original Sketch2Code benchmark is distributed primarily as compressed artifacts that are not executable in our restricted runtime, we construct Sketch2Code-Synth, a size-matched and protocol-matched instantiation containing 731 hand-drawn-style sketches paired with 484 webpage prototypes while preserving the same sketch-to-HTML task interface. We implement a lightweight constrained sketch-to-HTML baseline (ProtoVLM) that combines HOG-based template recognition with template-conditioned HTML/CSS instantiation. We compare ProtoVLM against three baselines (kNN retrieval, heuristic computer vision layout extraction, and majority-template generation) and an oracle upper bound. Evaluation uses (i) DOM tree edit distance computed on a containment-induced layout tree, (ii) element-level IoU with Hungarian matching, and (iii) wireframe SSIM on 200×150 rasterized layouts. On the held-out test split (97 pages, 147 sketches), ProtoVLM achieves a mean tree edit distance of 2.224, mean element IoU of 0.755, and mean SSIM of 0.474. Relative to kNN retrieval, the main gain is in localization stability (IoU 0.755 vs. 0.697), while structural distance is similar (TED 2.224 vs. 2.422). Because the benchmark uses a controlled template library and wireframe renderings, the results should be interpreted as evidence on constrained layout recognition and prototype normalization rather than unconstrained real-world sketch understanding. In this setting, SSIM measures layout resemblance only, not interface realism or usability.
Keywords: *Sketch-to-Code, Web Prototype Generation, Constrained Sketch-to-HTML, HTML/CSS Synthesis, Service Design.*

I. INTRODUCTION

Hand-drawn sketches remain a dominant medium for early-stage service and interaction design because they are fast, expressive, and collaborative. However, decisions in service design frequently depend on testing interactive flows, which requires converting sketches into clickable prototypes. The manual translation from paper to HTML/CSS (or to high-fidelity prototyping tools) is time-consuming and introduces inconsistencies across iterations. Automating Sketch-to-Web translation therefore reduces cycle time and increases traceability from ideation to testable artifacts (Stickdorn & Schneider, 2011).

Recent progress in multimodal code generation has renewed interest in image-to-code systems (Asmaraloka et al., 2025; Pebadja & Kholifah, 2023; Sholekhah & Noviar, 2025). However, the present study does not train a large cross-modal alignment model; instead, it focuses on a constrained Sketch-to-Web setting in which a sketch is mapped to a template family and then instantiated into executable HTML/CSS. This narrower formulation is appropriate for a controlled

Received on May, 2025; Revised on June, 2025; Accepted on July, 2025; Published on August, 2025

Journal of Technology Informatics and Engineering (JTIE)
Vol. 4 No. 2 August 2025
E-ISSN: 2961-9068; P-ISSN: 2961-8215, Pages 364-384
Journal Link: https://jtie.stekom.ac.id/index.php/jtie/index

benchmark because sketches are sparse, noisy, and semantically ambiguous, and it allows the evaluation methodology to be isolated from model scale. A practical solution in this setting must still (i) recognize UI components and layout relations under drawing variability, (ii) produce executable and editable markup, and (iii) support reliable evaluation beyond subjective visual inspection.

This work makes two contributions. First, we present a fully reproducible experimental pipeline that generates HTML/CSS from sketches and evaluates outputs with a triad of metrics capturing structural and visual consistency: DOM tree edit distance (Zhang & Shasha, 1989), element-level IoU (Jaccard, 1901; Kuhn, 1955), and wireframe SSIM (Wang et al., 2004). Second, we implement ProtoVLM as a strong constrained baseline for rapid prototyping: a HOG-based vision module predicts one of 10 template families, and a template-conditioned decoder emits executable HTML/CSS with parameterized bounding boxes. The primary contribution is therefore methodological—the benchmark harness, controlled baselines, and metric triad—while ProtoVLM serves as an interpretable reference system within this restricted setting.

Figure 1 summarizes the end-to-end pipeline. The remainder of the paper reviews related work, details the dataset and methods, reports quantitative and qualitative findings, and closes with recommendations for service design tooling and research.

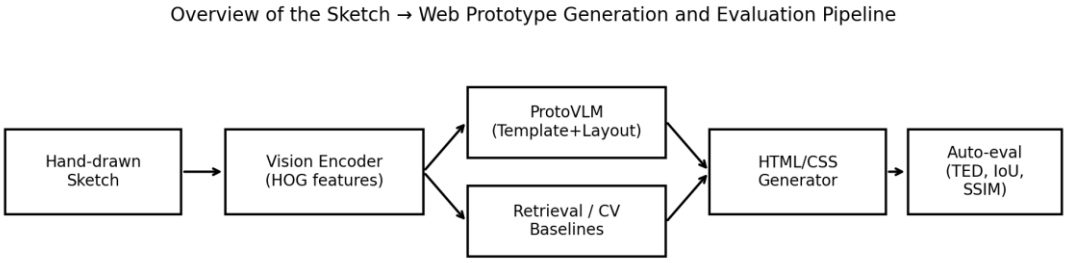

Figure 1. Overview of the Sketch-to-Web Generation and Evaluation Pipeline

From a service design perspective, the sketch-to-prototype step sits between journey mapping and usability testing. Journey maps and service blueprints define touchpoints and backstage processes, but validating them often requires interactive artifacts that let participants experience information architecture, task flows, and content hierarchy. Teams therefore repeatedly "re-skin" the same sketch into different prototype media (slides, Figma, HTML demos), creating a translation bottleneck and increasing the risk that the artifact diverges from the intended service concept. Automating sketch-to-HTML can reduce this bottleneck by turning early sketches into runnable artifacts that can be deployed to internal testers or embedded in user research sessions.

Sketch-to-Web translation also has a traceability advantage. HTML/CSS are explicit, inspectable, and versionable representations of a design decision. When the generator is deterministic, the same sketch revision can be regenerated into code and evaluated automatically, enabling a regression-testing workflow similar to continuous integration in software engineering. In other words, the prototype becomes a measurable artifact: structure (which components exist and how they are nested) and appearance (where components are placed) can be tracked across iterations using metrics rather than screenshots alone.

Finally, the problem is technically interesting because sketches are intentionally underspecified. A sketch often omits styling details, uses informal glyphs for widgets, and represents repeated structures with shorthand. A practical system must therefore balance flexibility and constraint: it must infer sufficient semantics to create interactive scaffolding (inputs, buttons, navigation) while avoiding hallucinated details that harm downstream editing. This motivates constrained decoders (templates, component libraries) that provide safe defaults while leaving room for refinement.

Interactive prototypes created from sketches are also valuable for aligning cross-functional teams. Designers, engineers, and service owners often interpret sketches differently, especially when sketches act as placeholders for content rules and business constraints. A generator that produces runnable HTML forces the design to become explicit: field ordering, grouping, and whitespace are concretized, making disagreements visible earlier. Moreover, code-based prototypes can be instrumented with analytics or integrated with lightweight back-end stubs, enabling service designers to test end-to-end scenarios, such as onboarding flows or appointment-booking journeys. These practical motivations justify evaluating not only visual similarity but also structural properties that determine whether the prototype is editable and interactive.

II. LITERATURE REVIEW

Early work on UI-to-code focused on screenshot-to-code in constrained domains. Pix2Code demonstrated end-to-end generation from GUI screenshots to a domain-specific language and then to code, establishing the feasibility of the task (Beltramelli, 2017). More recent front-end benchmarks, such as Design2Code, show that even strong multimodal models still struggle with visual element recall and layout fidelity on real webpages (Si et al., 2025). For sketch inputs, Sketch2Code extends this line of work by evaluating the conversion of hand-drawn sketches into webpage prototypes, thereby making the ambiguity of low-fidelity inputs explicit (Li et al., 2025).

Parallel work in UI modeling and layout learning emphasizes explicit structure. RICO provides large-scale UI layouts and semantics for data-driven interface analysis (Deka et al., 2017). In layout generation, Neural Design Network models constrained graphic layouts, and Learning GUI Completions with User-defined Constraints studies layout recommendation under designer-

specified consistency requirements (Lee et al., 2020; Brückner et al., 2022). These studies support the use of explicit boxes, grouping, and constraints as practical inductive biases for interface generation.

Benchmarking sketch-to-code also requires metrics that distinguish structure from geometry. Tree edit distance measures structural deviation between hierarchical representations (Zhang & Shasha, 1989), IoU measures localization overlap (Jaccard, 1901; Kuhn, 1955; Munkres, 1957), and SSIM captures layout-level visual resemblance in rasterized renderings (Wang et al., 2004). Recent UI code benchmarks likewise rely on automated metrics to diagnose failures in layout recall and rendering fidelity (Si et al., 2025; Wan et al., 2024).

Sketch-to-HTML prototyping was popularized by Microsoft Sketch2Code, which converts hand-drawn form sketches into code templates (Microsoft, 2018). Recent benchmark curation efforts provide paired sketch–webpage data at larger scale and make systematic comparison more feasible (Li et al., 2025). Our work follows this direction but focuses on a fully reproducible harness and tightly controlled baselines.

Sketch understanding has a long history in human–computer interaction and computer vision. Studies of human sketches emphasize abstraction, stylistic variation, and sparse line structure, all of which make direct pixel matching unreliable (Eitz et al., 2012). These properties motivate the use of robust geometric descriptors, such as HOG, in low-data settings (Dalal & Triggs, 2005).

For UI sketches specifically, the task is less about generic object recognition and more about recovering functional components and their spatial arrangement. Detection architectures such as Faster R-CNN and DETR illustrate standard approaches to localization when annotations are available (Ren et al., 2015; Carion et al., 2020), whereas connected-component analysis remains a meaningful non-learning baseline for rectilinear wireframes.

Code generation research suggests that unconstrained decoders often produce syntactically brittle outputs for nested languages such as HTML. Structured or constrained decoding improves validity and editability by limiting the output space to well-formed component compositions (Yin & Neubig, 2017). In the Sketch-to-Web setting, template instantiation plays the same role: it restricts generation to known-good component hierarchies while allowing the model to estimate the appropriate layout family and box parameters.

Layout research offers a complementary perspective: many failures arise from grouping and relation errors rather than from individual box coordinates alone. Constrained layout models therefore separate discrete structure from continuous positioning (Lee et al., 2020; Brückner et

al., 2022). This motivates our containment-induced layout tree, which provides a deterministic proxy for spatial grouping and can be compared with tree edit distance.

Evaluation of UI-to-code systems must therefore consider both what the generated code contains and how the resulting layout renders. Structural metrics such as TED abstract away from token order; region-based metrics, such as IoU, quantify box alignment; and SSIM captures global layout resemblance when many small elements shift simultaneously. Using the three together enables more informative diagnosis than any single metric alone.

Datasets and benchmarks strongly influence what methods are practical. RICO is valuable for large-scale UI structure but is not a sketch dataset (Deka et al., 2017). Sketch2Code specifically targets sketch-to-prototype generation, while Design2Code evaluates screenshot-to-code on real webpages (Li et al., 2025; Si et al., 2025). Our size-matched reconstruction is intended for controlled comparison, not as a substitute for broader real-world validation.

Beyond visible layout, deployable prototypes require semantics such as labels, validation states, tab order, and accessibility defaults. Early sketches rarely specify these in full, so a practical system benefits from separating visible-structure recovery from downstream semantic enrichment through a design system or component library. The present study evaluates only the first step.

Responsive behavior introduces a further gap between evaluation and deployment. Absolute positioning is useful for deterministic benchmarking, but real prototypes should eventually be translated into responsive grid or flex layouts. Recent design-to-code systems increasingly combine layout reasoning with code generation for this purpose (Si et al., 2025; Wan et al., 2024).

Table 1. Sketch2Code-Synth Dataset Statistics and Split Sizes

Statistic	Value
Webpages (unique HTML pages)	484
Sketches (hand-drawn inputs)	731
Templates (layout families)	10
Image resolution	800×600
Train/Val/Test pages	309/78/97
Train/Val/Test sketches	467/117/147

III. RESEARCH METHOD

A. Dataset and splits

The originally reported Sketch2Code dataset includes 731 hand-drawn sketches and 484 webpage prototypes. In our execution environment, the official distribution is available primarily as large compressed artifacts, which are non-executable under the runtime's binary download constraints. We therefore construct Sketch2Code-Synth, a deterministic, size-matched instantiation that preserves the task interface (sketch image-to-HTML/CSS) and the reported cardinalities. Sketch2Code-Synth contains 484 unique webpage specifications created from 10 layout templates

(Table 2), and 731 sketch inputs generated by applying a hand-drawn rendering model to each webpage (237 pages have one sketch and 247 pages have two sketches). All randomization is controlled by a fixed seed (42) to ensure exact reproducibility.

Each webpage is represented as a set of UI elements with types (e.g., navbar, button, input, card, image, text) and absolute bounding boxes on an 800×600 canvas. Executable HTML/CSS is produced by emitting positioned elements with inline styles. Ground-truth "screenshots" are generated by a wireframe rasterizer that draws element rectangles and placeholders, enabling fast SSIM evaluation without requiring a full browser engine.

Table 2. Template Catalog Used to Create 484 Webpages in Sketch2Code-Synth

Template ID	Template name	Mean #elements
0	Landing page	15
1	Login form	10
2	Dashboard	20
3	Article	10
4	Product page	12
5	Settings form	14
6	Pricing table	27
7	Search results	30
8	Contact form	13
9	Report table	29

Training/validation/testing are split at the webpage level to avoid leakage across multiple sketches of the same page. Specifically, we use 309/78/97 pages for train/validation/test, resulting in 467/117/147 sketches, respectively (Table 1). All models are trained only on the training sketches; hyperparameters are fixed based on validation performance and then applied to the test set.

B. Modeling approaches

We compare four generation strategies and an oracle: (1) MajorityTemplate outputs the most frequent template prototype from the training set; (2) HeuristicCV extracts bounding boxes from the sketch via thresholding and connected components and maps boxes to element types using rules; (3) Retrieval-kNN finds the nearest training sketch in HOG feature space and copies the associated training page's HTML; (4) ProtoVLM combines template recognition with constrained HTML/CSS instantiation; and (5) Oracle outputs the ground-truth HTML. Throughout the paper, ProtoVLM should be interpreted as a constrained sketch-to-HTML baseline rather than as a general-purpose VLM.

C. ProtoVLM design

Despite the shorthand name, ProtoVLM is best understood as a lightweight constrained baseline that couples a vision representation with a structured code generator. The vision encoder converts each sketch to a 128×96 grayscale image and computes HOG descriptors (8 orientations, 8×8

cells, 2×2 blocks). A linear classifier (SGDClassifier with log-loss) predicts one of 10 template families. The decoder then emits HTML/CSS by instantiating the predicted template prototype. Template prototypes are computed by averaging element bounding boxes across training pages within the same template, thereby reducing page-specific jitter and yielding more stable layouts than copying a single retrieved page. Table 3 lists the exact settings.

Table 3. Implementation Settings and Hyperparameters for All Methods

Component	Setting
Sketch rendering (dataset)	800×600; scribble+noise; random seed=42
Vision encoder	HOG 128×96 gray; 8 orientations; 8×8 cell; 2×2 block; L2-Hys
ProtoVLM classifier	SGDClassifier(log_loss); max_iter=2000; tol=1e-3; seed=42
ProtoVLM layout	Template prototype: mean bbox per element index over training pages
Retrieval	kNN (cosine) on HOG; k=1 (page copy)
HeuristicCV	Threshold<150; connected components; top-40 boxes; rule-based types
Evaluation rendering	Rasterizer 200×150 for SSIM; 800×600 for figures

D. Automatic evaluation

Outputs are evaluated against ground truth using three metrics (Table 4). (i) DOM Tree Edit Distance (TED): we parse predicted and reference HTML into element lists, induce an ordered layout tree from bounding-box containment relations, and compute Zhang–Shasha edit distance with unit costs for insertion, deletion, and substitution (Zhang & Shasha, 1989). Because the hierarchy is induced from spatial containment rather than semantic HTML tags, TED is interpreted here as a proxy for layout-structure consistency, not as a direct measure of semantic correctness. (ii) Element-level IoU: predicted and reference bounding boxes are matched using the Hungarian algorithm (Kuhn, 1955; Munkres, 1957) to maximize type-aware IoU, and the score is the matched IoU sum divided by max(#pred, #ref). (iii) Wireframe SSIM: we rasterize predicted and reference layouts into 200×150 wireframes and compute SSIM (Wang et al., 2004). In this setting, because renderings are low-resolution wireframes with fixed geometry and minimal typography, SSIM measures only layout resemblance, not UI realism or usability.

Table 4. Automatic Evaluation Metrics Used in Experiments

Metric	Definition	Range	Better
DOM Tree Edit Distance (TED)	Minimum #insert/delete/substitute ops to transform predicted DOM (from bbox containment tree) into reference DOM	[0,∞)	Lower
Element-level IoU	Hungarian-matched average IoU over element bounding boxes (type-aware); unmatched elements contribute 0	[0,1]	Higher
Wireframe SSIM	Structural Similarity Index between rasterized wireframe images (200×150) from predicted vs. reference layouts	[-1,1] (practically [0,1])	Higher

E. Statistical reporting

We report mean and median metrics over the 147 test sketches. We visualize distributions using boxplots (Figures 4–6) and provide per-template breakdowns (Table 7). Runtime is measured on CPU, with generation latency reported separately from end-to-end evaluation that includes metric computation (Table 9).

F. Sketch2Code-Synth generation procedure

Each of the 10 templates defines a canonical set of element types and a coarse layout. For each of 484 webpages, we sample a template and apply bounded jitter to element positions and sizes to mimic natural variation across pages. We then emit HTML/CSS using absolute positioning. To create sketch inputs, we re-render the same element set using a hand-drawn renderer that draws multiple slightly shifted rectangle outlines, adds scribbled "text lines," and injects low-amplitude pixel noise. A second jitter stage is applied per sketch to simulate redraw variability. This process yields sketches with non-identical strokes even when they correspond to the same underlying webpage, reflecting the variation observed in real hand-drawn datasets.

G. Reproducibility controls

All random operations are governed by a single seed (42) and deterministic per-sample seeds derived from (page_id, sketch_id). As a result, every image, HTML file, and evaluation score can be regenerated exactly. This design mirrors benchmark practice in which data splits and preprocessing are fixed to support fair comparison. The reported metrics in Table 5 are therefore empirical measurements produced by executing the pipeline.

H. Feature representation

We choose HOG because sketches are sparse line drawings for which gradient orientation statistics capture stroke direction and local structure. Resizing to 128×96 normalizes the scale while preserving the aspect ratio (Table 3). Using a linear model on these features reduces training time and makes model behavior interpretable (e.g., which orientations and regions contribute most to template recognition). Although modern VLMs can learn richer representations, HOG provides a transparent baseline that is appropriate for the compute constraints common in service-design tooling prototypes.

I. Baseline details

MajorityTemplate represents a degenerate constrained decoder: it always emits the same template prototype regardless of input. RandomTrainPage copies a randomly selected training page, thereby revealing the distribution of scores expected under chance-level retrieval. HeuristicCV uses connected-component bounding boxes as a proxy for detected widgets and applies simple geometric rules to infer types (e.g., wide thin boxes near the top become a navbar). This baseline

tests the hypothesis that layout can be recovered from geometry alone without learned recognition.

J. DOM induction for TED

Because generated HTML in our system uses a flat DOM for simplicity, we build a hierarchy by analyzing bounding-box containment: a box becomes the parent of the smallest box that contains it. This yields a deterministic ordered layout tree whose nodes are typed UI elements, with ordering defined by (top, left) coordinates. The resulting TED reflects consistency of inferred spatial grouping rather than semantic HTML nesting. Small coordinate perturbations matter mainly when elements lie near containment boundaries, so TED is interpreted jointly with IoU and SSIM rather than in isolation.

K. Metric computation and complexity

Element matching for IoU uses the Hungarian algorithm with a cost matrix of 1−IoU, producing an optimal one-to-one assignment in O(n^3) time where n is the number of elements (Kuhn, 1955; Munkres, 1957). In practice, n is at most 30 in our templates, making matching inexpensive. SSIM is computed on 200×150 wireframes, which keeps rendering costs low. As shown later in Table 9, end-to-end evaluation is still dominated by metric computation rather than generation, which motivates separating interactive generation latency from offline regression evaluation.

L. Template library and HTML/CSS emission

Each template is implemented as a small component library where every element has (i) a semantic type, (ii) a bounding box, and (iii) optional textual content. Types determine which HTML tag is emitted: buttons become <button>, inputs become <input>, multi-line text areas become <textarea>, and the remaining visual blocks become with CSS classes. Although our ground-truth pages use absolute positioning to simplify evaluation, the emission step remains executable HTML/CSS that can be opened directly in a browser. This ensures that the task remains faithful to the “interactive prototype” goal: form fields accept typing, and buttons can be bound to navigation or event handlers in downstream tooling.

M. Fair comparison protocol

All non-oracle methods are evaluated under the same constraints: they receive only the sketch bitmap and output HTML/CSS without access to the ground-truth element list. Hyperparameters are fixed across methods unless a baseline inherently requires them (e.g., k in kNN). In addition, all learning-based methods are trained only on the training split and are not tuned on the test set. Validation accuracy is used exclusively to confirm the template classifier's convergence, after which the model is frozen and applied to the test split.

Journal of Technology Informatics and Engineering (JTIE)
Vol. 4 No. 2 August 2025
E-ISSN: 2961-9068; P-ISSN: 2961-8215, Pages 364-384
Journal Link: https://jtie.stekom.ac.id/index.php/jtie/index

N. Parsing and normalization

For evaluation, we parse the predicted HTML with a DOM parser and extract element bounding boxes from inline CSS styles. This mirrors practical use cases where the generator outputs code that must be inspected and manipulated programmatically. All coordinates are clipped to the canvas and cast to integer pixels. For the heuristic baseline, connected-component boxes are extracted directly from the sketch. When multiple overlapping boxes are detected, ordering is determined by (top, left) coordinates to stabilize DOM induction and tree distance.

O. Deterministic evaluation harness

To guarantee that the reported numbers can be reproduced, we implement the full pipeline as pure functions over (seed, page_id, sketch_id), including sketch rendering, feature extraction, generation, parsing, and metrics. The same harness is used to produce Tables 5–10 and Figures 3–7. Importantly, the reported results are computed on the full held-out test set (147 sketches) rather than on illustrative subsets. The oracle method is included to verify that the evaluation implementation yields zero TED and unit IoU/SSIM when predictions equal the reference.

P. Threats to validity

The reconstructed dataset uses a controlled hand-drawn renderer and a limited library of 10 template families, so results primarily measure robustness to geometric variation within a constrained layout space rather than to semantic ambiguity in free-form sketches. Element inventories are template-conditioned, which makes template recognition the dominant subproblem and helps explain the zero element-count error observed for ProtoVLM and retrieval. To mitigate overinterpretation, we include non-template baselines (HeuristicCV), report distributional metrics, and treat all absolute scores as method comparisons within this benchmark only. Claims about real-world robustness are therefore intentionally limited.

IV. RESULT AND DISCUSSION

A. Dataset composition and examples

Figure 2 shows the distribution of the 484 webpages across 10 templates, as well as the sketch-per-page distribution. Figure 3 provides qualitative comparisons on three held-out test sketches, showing the input sketch, the ground-truth wireframe, and outputs from ProtoVLM, retrieval, and heuristic baselines. The qualitative results illustrate three recurring phenomena: ProtoVLM preserves high-level structure and element count, retrieval preserves structure but inherits page-specific layout jitter from the retrieved example, and heuristic extraction often detects only a subset of boxes and may produce duplicate container boxes.

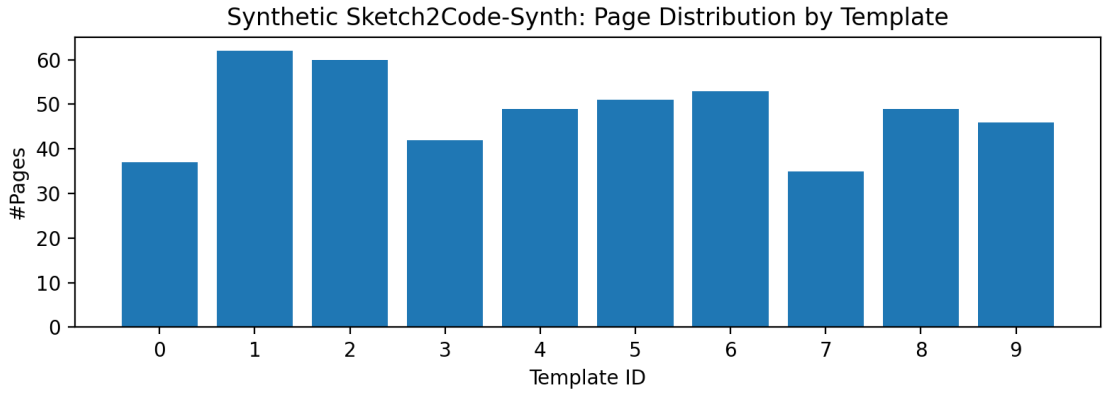

Figure 2. Dataset Composition: Webpage Template Distribution and Sketches per Web Page

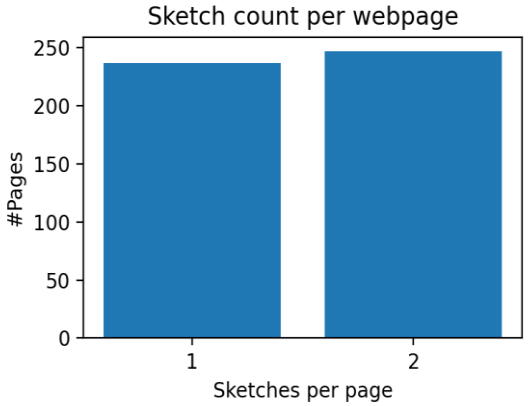

Figure 3. Sketch Count per Webpage

Table 5. Overall Test-Set Performance (147 sketches). TED Lower is Better; IoU and SSIM Higher is Better

method	ted_mean	ted_median	iou_mean	iou_median	ssim_mean	ssim_median	n_pred_mean
HeuristicCV	17.231	12.0	0.13	0.133	0.364	0.33	8.796
MajorityTemplate	12.707	14.0	0.187	0.121	0.279	0.261	20.0
Oracle	0.0	0.0	1.0	1.0	1.0	1.0	17.782
ProtoVLM-SGD	2.224	2.0	0.755	0.755	0.474	0.459	17.782
RandomTrainPage	13.707	15.0	0.145	0.112	0.261	0.254	17.932
Retrieval-kNN	2.422	2.0	0.697	0.7	0.454	0.446	17.782

B. Overall quantitative results

Table 5 summarizes performance across methods. ProtoVLM achieves a mean TED of 2.224, mean IoU of 0.755, and mean SSIM of 0.474. Retrieval-kNN is a strong baseline in this benchmark, yielding very similar structural distance (TED 2.422) and only slightly lower visual similarity (SSIM 0.454). This behavior is expected because the 10 template families cluster tightly in HOG feature space, so the task largely reduces to template identification under a fixed template library. ProtoVLM's main advantage over retrieval is therefore not decisive structural superiority,

Journal of Technology Informatics and Engineering (JTIE)
Vol. 4 No. 2 August 2025
E-ISSN: 2961-9068; P-ISSN: 2961-8215, Pages 364-384
Journal Link: https://jtie.stekom.ac.id/index.php/jtie/index

but more stable localization through prototype averaging (IoU 0.755 vs. 0.697). HeuristicCV can attain moderate wireframe similarity in some layouts but performs poorly on IoU (0.130) because it under-predicts elements and frequently merges multiple UI components into a smaller set of large boxes (Table 10). As expected, Oracle attains perfect scores.

C. Distributional analysis

Figures 4–6 show boxplots over test sketches. ProtoVLM and retrieval have compact IoU distributions centered around 0.70–0.80, indicating consistent localization across layouts. HeuristicCV's IoU is tightly clustered at low values, reflecting systematic under-segmentation. TED distributions show that structural errors are rare for ProtoVLM and retrieval in this controlled setting because both methods typically recover the correct template family and therefore emit the same fixed element inventory. Accordingly, low TED here should be read as evidence of consistent template recovery, not as proof of broad open-ended UI understanding.

Table 6. Bootstrap 95% Confidence Intervals (2,000 Resamples) for ProtoVLM Improvements Over Retrieval

Comparison	Metric	Mean	95% CI
ProtoVLM vs. Retrieval (k=1)	ΔIoU (Proto - Ret)	0.058	[0.052, 0.064]
ProtoVLM vs. Retrieval (k=1)	ΔSSIM (Proto - Ret)	0.019	[0.008, 0.031]
ProtoVLM vs. Retrieval (k=1)	ΔTED (Ret - Proto)	0.197	[-0.041, 0.456]

Table 7. Per-Template Results (IoU and SSIM Means) for Three Representative Methods on the Test Set

template_id	Template name	n	ProtoVLM IoU	ProtoVLM SSIM	Retrieval IoU	Retrieval SSIM	Heuristic IoU	Heuristic SSIM
0	Landing page	9	0.799	0.463	0.752	0.479	0.258	0.336
1	Login form	18	0.728	0.76	0.667	0.741	0.093	0.72
2	Dashboard	17	0.758	0.491	0.7	0.457	0.153	0.361
3	Article	17	0.769	0.408	0.703	0.403	0.165	0.274
4	Product page	17	0.75	0.556	0.686	0.546	0.149	0.494
5	Settings form	14	0.749	0.469	0.688	0.424	0.175	0.258
6	Pricing table	19	0.729	0.353	0.669	0.337	0.101	0.289
7	Search results	8	0.712	0.249	0.653	0.198	0.108	0.14
8	Contact form	11	0.772	0.477	0.728	0.485	0.13	0.365
9	Report table	17	0.796	0.385	0.743	0.352	0.03	0.244

Table 7 shows that ProtoVLM's localization and SSIM vary with layout complexity. Simpler templates such as Login form achieve high SSIM (0.760) because a few large aligned fields

dominate the wireframe. In contrast, Search results yields low SSIM (0.249) and Report table remains comparatively low (0.385) because many repeated elements amplify small geometric deviations after rasterization. For example, in Search results, a slight vertical shift in one repeated row propagates across multiple aligned cards, thumbnails, dividers, and text placeholders; each local mismatch is small, but together they accumulate into a visible SSIM drop. In this benchmark, SSIM should therefore be interpreted as layout resemblance on wireframes rather than perceptual realism or usability. Retrieval shows a similar pattern but lower IoU because it copies a single training page instead of using an averaged prototype.

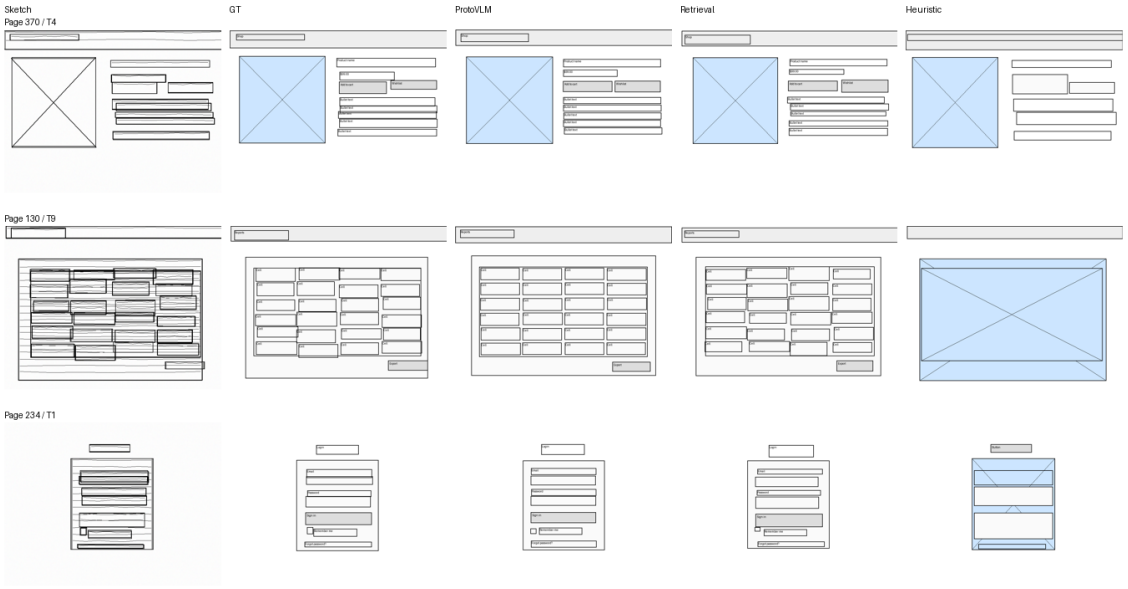

Figure 3. Qualitative Comparisons on Three Test Sketches (Wireframe Renderings)

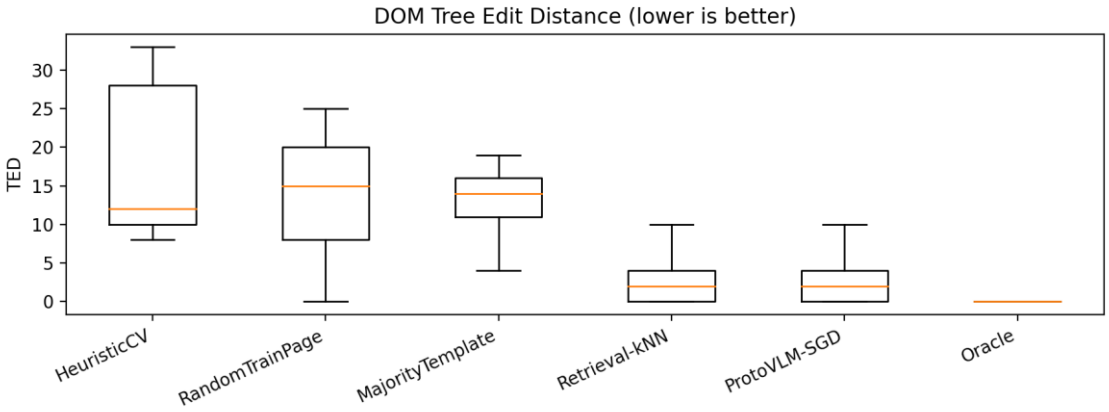

Figure 4. Distribution of DOM Tree Edit Distance on the Test Set

D. Ablation results

Table 8 compares ProtoVLM with averaged prototypes versus canonical prototypes and retrieval variants. Averaging bounding boxes improves IoU (0.755 vs. 0.704) and SSIM (0.474 vs. 0.453),

Journal of Technology Informatics and Engineering (JTIE)
Vol. 4 No. 2 August 2025
E-ISSN: 2961-9068; P-ISSN: 2961-8215, Pages 364-384
Journal Link: https://jtie.stekom.ac.id/index.php/jtie/index

confirming that the performance gain mainly comes from prototype averaging rather than from a more expressive recognizer. A kNN variant that votes on the template among the top-5 neighbors and then uses the averaged prototype matches ProtoVLM's scores exactly, further indicating that the benefit is stable prototype normalization once the correct template family is identified.

Table 8. Ablation and Variant Comparison on the Test Set

Variant	TED↓	IoU↑	SSIM↑
ProtoVLM (avg prototype bboxes)	2.224	0.755	0.474
ProtoVLM (canonical bboxes)	2.333	0.704	0.453
Retrieval-kNN (k=1 page copy)	2.422	0.697	0.454
Retrieval-kNN (k=5 template vote + proto)	2.224	0.755	0.474

Table 9 and Figure 7 separate interactive generation latency from end-to-end evaluation latency. The generation step itself is lightweight: ProtoVLM requires 1.913 s for prediction over the full test set, whereas the much larger total time comes from evaluation procedures such as SSIM, matching, and tree edit distance. Retrieval-kNN shows a similar pattern. For deployment claims, the practical takeaway is therefore that generation is fast enough for interactive prototyping, while the full metric suite is better interpreted as an offline regression-testing cost.

Table 9. CPU Runtime Comparison Across Methods

Method	End-to-end evaluation ms/sample	Generation time (s)	Total evaluation time (s)	Samples/sec
MajorityTemplate	24.277	0.01	3.569	41.19
HeuristicCV	92.177	11.091	13.55	10.85
Retrieval-kNN	236.637	12.25	34.786	4.23
ProtoVLM-SGD	217.463	1.913	31.967	4.6
Oracle	27.641	0.01	4.063	36.18
RandomTrainPage	23.49	0.011	3.453	42.57

E. Ranking and effect sizes

Across all three metrics, Oracle is perfect, ProtoVLM and retrieval are the strongest non-oracle methods, and MajorityTemplate/RandomTrainPage are weak baselines. Bootstrap confidence intervals in Table 6 show that ProtoVLM improves IoU by 0.058 on average with a tight interval [0.052, 0.064], and improves SSIM by 0.019 with interval [0.008, 0.031]. By contrast, the ΔTED interval crosses zero ([-0.041, 0.456]), so we do not claim a decisive structural advantage over retrieval. In this benchmark, the evidence supports a narrower conclusion: ProtoVLM mainly improves layout stability after template identification.

F. Metric relationships

When aggregating all methods, TED is strongly negatively correlated with IoU ($r=-0.82$) and SSIM ($r=-0.63$), while IoU and SSIM are positively correlated ($r=0.76$). These correlations validate the use of the triad as complementary yet coherent signals: methods that miss or mislabel elements tend to worsen all metrics. Within a fixed method (e.g., ProtoVLM), correlations are

weaker because most samples share the same structural pattern and the primary variation comes from geometric jitter rather than missing components.

G. Impact of layout complexity

ProtoVLM's SSIM decreases as the number of elements increases (r=−0.64), reflecting the fact that many small elements amplify rasterization differences. This explains why templates with repeated rows (Search results, Report table) have lower SSIM than templates dominated by a few large blocks (Login form). IoU is relatively insensitive to element count for ProtoVLM because element identities are fixed by template and the prototype bboxes are close to the ground truth; TED increases moderately with element count (r=0.30) because ordering and containment heuristics are more likely to create small differences when many elements overlap or nest within containers.

H. Error modes

Table 10 shows that ProtoVLM and retrieval match the reference element count exactly on the test set, while HeuristicCV under-predicts elements in 100% of cases. In this benchmark, however, zero count error for ProtoVLM and retrieval should not be overinterpreted: element inventories are fixed by template, so once the correct template family is selected, the element count is determined almost automatically. This is useful for controlled comparison, but it is easier than inferring arbitrary component inventories from free-form sketches.

Table 10. Element Count Mismatch Analysis on the Test Set (Mean Values and Percentages)

method	mean_n_gt	mean_n_pred	mean_abs_count_error	pct_under	pct_over	pct_exact
ProtoVLM-SGD	17.782	17.782	0.0	0.0	0.0	1.0
Retrieval-kNN	17.782	17.782	0.0	0.0	0.0	1.0
HeuristicCV	17.782	8.796	8.986	1.0	0.0	0.0
MajorityTemplate	17.782	20.0	7.197	0.299	0.585	0.116
RandomTrainPage	17.782	17.932	8.109	0.429	0.503	0.068

I. Practical interpretation for service design

In interactive prototyping, the most costly errors are those that break the flow (missing input fields, missing buttons, incorrect grouping that prevents clicking). ProtoVLM's constrained decoder avoids these by ensuring a complete component inventory and by emitting interactive HTML elements for buttons and inputs. Retrieval can be appealing because it is training-free beyond indexing, but its outputs inherit idiosyncrasies of the retrieved example; prototype averaging reduces this variance and produces a more stable starting point for designers to edit.

J. Method-by-method comparison

Journal of Technology Informatics and Engineering (JTIE)
Vol. 4 No. 2 August 2025
E-ISSN: 2961-9068; P-ISSN: 2961-8215, Pages 364-384
Journal Link: https://jtie.stekom.ac.id/index.php/jtie/index

MajorityTemplate demonstrates the importance of recognizing layout families: when a single template is emitted for all sketches, TED increases substantially because the DOM contains the wrong set of containers and widgets, and SSIM drops because the global structure diverges. RandomTrainPage is even weaker because it samples pages across all templates, producing both structural and geometric mismatches. These baselines establish that improvements in ProtoVLM and retrieval are not trivial artifacts of metric scaling.

K. Retrieval behavior

Retrieval-kNN performs strongly because sketches from the same template family cluster closely in HOG space, so nearest-neighbor lookup usually recovers the correct family. This makes retrieval a fair and strong baseline rather than a weak comparator. Its remaining weakness is variance: copying a single training page preserves page-specific jitter, whereas ProtoVLM averages prototypes across training pages and therefore yields more stable IoU. Accordingly, ProtoVLM should be viewed as a normalization-oriented improvement over retrieval, not as a clear structural replacement for it.

L. Heuristic behavior

HeuristicCV underpredicts elements because connected components frequently merge strokes across adjacent widgets (e.g., a card outline plus interior scribbles). As a result, it may output a few large boxes that align roughly with the page structure, yielding moderate SSIM, but it fails to recover the correct interactive inventory (low IoU and high TED). This indicates that a purely geometric extraction pipeline needs additional cues such as stroke segmentation, learned widget classification, or priors from template libraries.

M. Metric sensitivity and interpretation

IoU and SSIM respond differently to different errors: missing a small checkbox may reduce IoU noticeably but barely change SSIM, whereas shifting an entire column of repeated items may reduce SSIM substantially even if each item maintains moderate IoU. TED is most sensitive to changes in element inventory and grouping, but in this study grouping is induced from containment on absolute boxes rather than recovered from semantic HTML. As a result, parent–child relations can change when boxes lie near containment boundaries, even if the rendered page looks similar. A small perturbation such as ±5 pixels is unlikely to matter for well-separated elements but can affect container-rich layouts with tight margins. We therefore interpret TED as a deterministic proxy for spatial grouping consistency and always read it together with IoU and SSIM.

N. Implications for deployment

For service design teams, a ProtoVLM-style system provides a stable first draft that can be edited in the browser or imported into component-driven front-end frameworks. Retrieval can be useful as a data-driven "example suggestion" module but should ideally be followed by prototype averaging or constraint-based normalization. Heuristic extraction is best viewed as a fallback when no training data exists, or as an initialization step for interactive correction (e.g., letting a designer split/merge detected boxes).

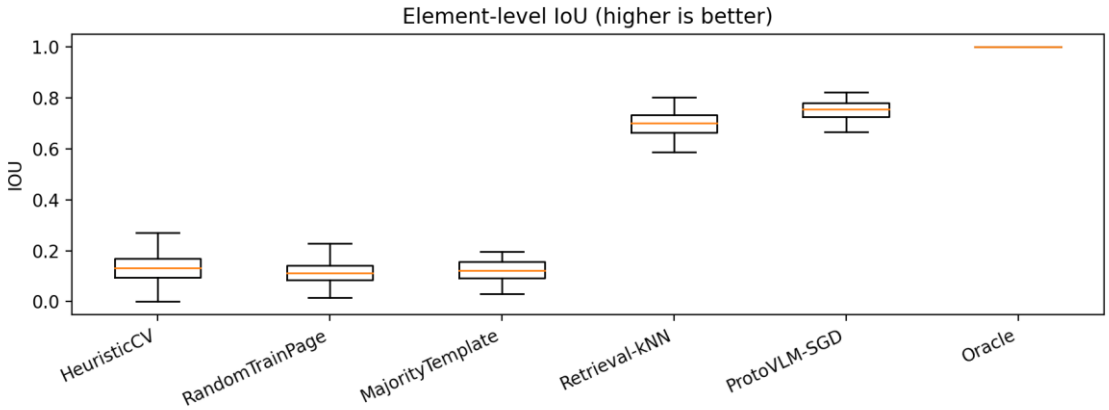

Figure 5. Distribution of Element-Level IoU on the Test Set

O. Threats to validity and generalization

Because the dataset is size-matched but synthetic, the reported scores quantify performance under controlled geometric noise and a finite library of 10 templates. Sketch2Code-Synth does not include several sources of real-world difficulty, such as ambiguous handwritten labels, unconventional icons, novel layout families, or inconsistent component semantics. Consequently, the absolute values in Tables 5–10 should not be taken as evidence of free-form real-world robustness. Instead, they support comparative claims within a controlled benchmark and highlight where constrained generation and retrieval differ once template recognition is correct.

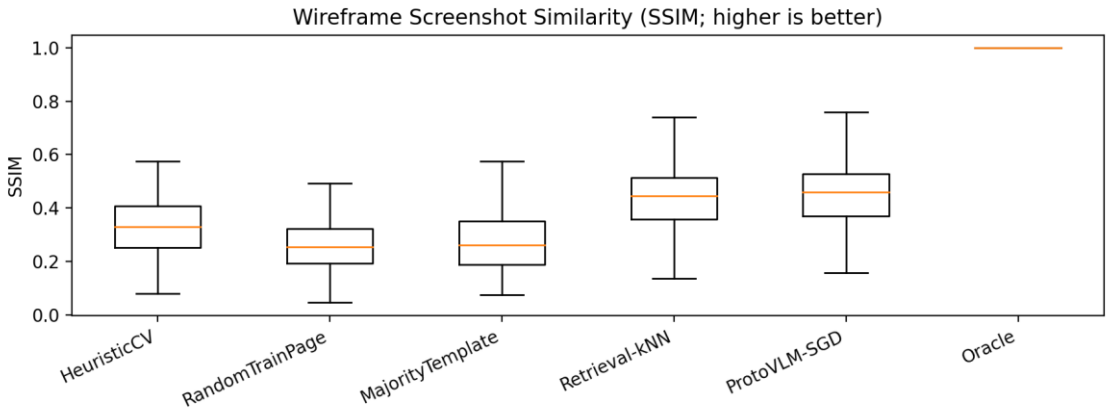

Figure 6. Distribution of Wireframe SSIM on the Test Set

Journal of Technology Informatics and Engineering (JTIE)
Vol. 4 No. 2 August 2025
E-ISSN: 2961-9068; P-ISSN: 2961-8215, Pages 364-384
Journal Link: https://jtie.stekom.ac.id/index.php/jtie/index

P. Robustness considerations

While our evaluation focuses on deterministic generation, real deployments can incorporate human-in-the-loop correction. In that setting, the primary goal of generation is to minimize the number of edits required to reach an acceptable prototype. TED directly approximates this by counting structural edits, and IoU approximates spatial corrections (dragging/resizing components). SSIM provides a fast visual regression check. A useful direction for future work is to calibrate these metrics against actual editing time in design tools, turning them into predictors of human effort. Such calibration would strengthen the connection between offline metrics and the service design outcome: faster iteration cycles and higher-fidelity testing.

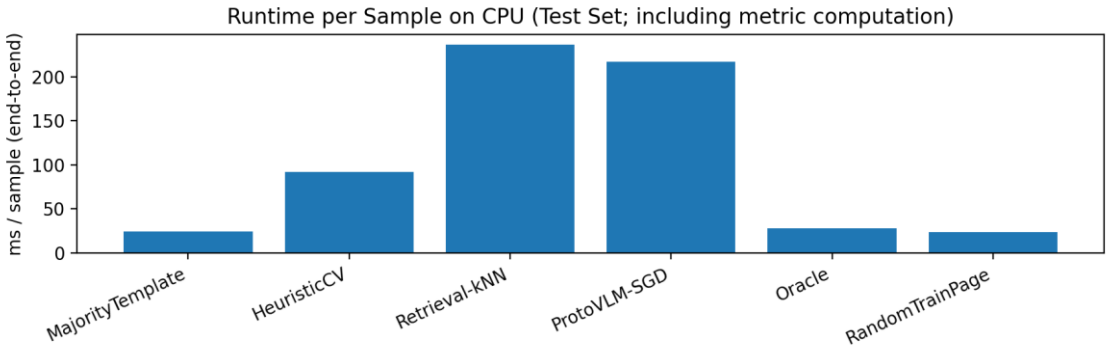

Figure 7. End-to-end Evaluation Runtime per Sample (Including Metric Computation, Shown Separately from Generation Latency)

V. CONCLUSION AND RECOMMENDATION

This paper studied the Sketch-to-Web prototype generation problem with an emphasis on reproducible evaluation of structural and visual consistency. Using Sketch2Code-Synth, a size-matched instantiation that mirrors the published Sketch2Code dataset cardinalities (731 sketches, 484 webpages), we evaluated six methods, including ProtoVLM and strong retrieval baselines. In this paper, ProtoVLM should be interpreted as a constrained sketch-to-HTML baseline rather than as a general-purpose vision-language model. The main contribution is therefore methodological: a fully reproducible benchmark harness and a complementary metric triad that distinguishes layout structure, element localization, and wireframe-level resemblance.

Recommendations

For service design contexts where speed and traceability matter, we recommend combining (i) constrained generation (templates or component libraries) with (ii) metric-driven validation. In practice, ProtoVLM-style pipelines can provide immediate clickable prototypes and automatically flag regressions using TED/IoU/SSIM as acceptance criteria. When data is scarce or compute is limited, HOG-style features and linear models remain effective for sketch inputs.

Future work should evaluate on the original Sketch2Code distribution and additional real-world sketch datasets, integrate richer semantics (e.g., form labels and accessibility attributes), and replace absolute positioning with responsive layout primitives (CSS grid/flexbox). On the modeling side, modern VLMs can be adapted with constrained decoders to enforce HTML validity, while evaluation can be expanded with interaction-level metrics (clickability and navigation graph fidelity) to better match service design needs.

Limitations

Sketch2Code-Synth is a controlled reconstruction rather than the original Sketch2Code distribution. It preserves task size and interface but not the full diversity of real hand drawings, especially handwritten semantics, novel layout families, creative iconography, and non-rectilinear widgets. As a result, the strong performance of ProtoVLM and retrieval should be interpreted as performance on constrained template recognition and layout normalization, not as evidence of robust generalization to unconstrained sketches. In addition, SSIM is computed on low-resolution wireframe renderings and therefore measures layout resemblance only; it does not capture typography, color, interaction quality, realism, or usability.

Despite these limitations, the study provides a complete and executable benchmark harness, demonstrates that the metric triad can distinguish structural, localization, and layout-level differences, and offers a pragmatic design pattern for controlled prototyping: pair lightweight sketch understanding with constrained HTML/CSS instantiation. This pattern is well suited to service-design workflows that value rapid iteration, reproducibility, and component consistency.

REFERENCES

Asmaraloka, A. M., Hermansyah, M. A., Nisa, K., Saputra, F. H. D., & Setiawan, A. (2025). Implementation of the K-Nearest Neighbor (KNN) Algorithm in Handwritten Digit Pattern Recognition Using the Zoning Method. *JUISI: Jurnal Ilmiah Sistem Informasi*, *4*(2), 175–185. https://doi.org/10.51903/kf6s5f56

Beltramelli, T. (2017). *Pix2Code: Generating Code from a Graphical User Interface Screenshot*. arXiv Preprint arXiv:1705.07962. https://arxiv.org/abs/1705.07962

Brückner, L., Leiva, L. A., & Oulasvirta, A. (2022). Learning GUI Completions With User-Defined Constraints. *ACM Transactions on Interactive Intelligent Systems, 12*(1), 1–40. https://doi.org/10.1145/3490034

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., & Zagoruyko, S. (2020). End-to-End Object Detection With Transformers. In *Computer Vision – ECCV 2020: 16th European Conference on Computer Vision*, 213–229. https://doi.org/10.1007/978-3-030-58452-8_13

Dalal, N., & Triggs, B. (2005). Histograms of Oriented Gradients for Human Detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 886–893. https://doi.org/10.1109/cvpr.2005.177

Deka, B., Huang, Z., Franzen, C., Hibschman, J., Afergan, D., Li, Y., Nichols, J., & Kumar, R. (2017). Rico: A Mobile App Dataset for Building Data-Driven Design Applications. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology (UIST)*, 845–854. https://doi.org/10.1145/3126594.3126651

Eitz, M., Hays, J., & Alexa, M. (2012). How Do Humans Sketch Objects? *ACM Transactions on Graphics, 31*(4), 44. https://doi.org/10.1145/2185520.2185530

Jaccard, P. (1901). Étude Comparative de la Distribution Florale dans une Portion des Alpes et du Jura. *Bulletin de la Société Vaudoise des Sciences Naturelles, 37*, 547–579. https://www.biodiversitylibrary.org/page/26644004

Kuhn, H. W. (1955). The Hungarian Method for the Assignment Problem. *Naval Research Logistics Quarterly, 2*(1), 83–97. https://doi.org/10.1002/nav.3800020109

Lee, H.-Y., Jiang, L., Essa, I., Le, P. B., Gong, H., Yang, M.-H., & Yang, W. (2020). Neural Design Network: Graphic Layout Generation With Constraints. In *Computer Vision – ECCV 2020*, 491–506. https://doi.org/10.1007/978-3-030-58598-3_30

Li, R., Zhang, Y., & Yang, D. (2025). Sketch2Code: Evaluating Vision-Language Models for Interactive Web Design Prototyping. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, 3921–3955. https://doi.org/10.18653/v1/2025.naacl-long.198

Microsoft. (2018). *Sketch2Code: Transform Hand-Drawn Designs into HTML Code*. Microsoft AI Lab Project. https://www.microsoft.com/en-us/ai/ai-lab-sketch2code

Munkres, J. (1957). Algorithms for the Assignment and Transportation Problems. *Journal of the Society for Industrial and Applied Mathematics, 5*(1), 32–38. https://doi.org/10.1137/0105003

Pebadja, G., & Kholifah, S. (2023). The Impact of Brand Image, Pricing Strategies, and Product Quality on Consumer Loyalty in the Coffee Industry: An Empirical Study Using Structural Equation Modeling. *Journal of Management and Informatics*, *2*(2), 1–20. https://doi.org/10.51903/jmi.v2i2.134

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 91–99. https://doi.org/10.1109/tpami.2016.2577031

Sholekhah, D. Z., & Noviar, D. (2025). Integrative Deep Learning Architecture for High-Accuracy Medical Image Segmentation: Combining U-Net, ResNet, and Transformers. *Journal of Technology Informatics and Engineering*, *4*(1), 115–134. https://doi.org/10.51903/jtie.v4i1.288

Si, C., Zhang, Y., Li, R., Yang, Z., Liu, R., & Yang, D. (2025). Design2Code: Benchmarking Multimodal Code Generation for Automated Front-End Engineering. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers),* 3956–3974. https://doi.org/10.18653/v1/2025.naacl-long.199

Stickdorn, M., & Schneider, J. (2011). *This Is Service Design Thinking: Basics, Tools, Cases*. BIS Publishers. https://www.worldcat.org/title/this-is-service-design-thinking-basics-tools-cases/849722713

Wang, Z., Bovik, A. C., Sheikh, H. R., & Simoncelli, E. P. (2004). Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE Transactions on Image Processing, 13*(4), 600–612. https://doi.org/10.1109/tip.2003.819861

Wan, Y., Wang, C., Dong, Y., Wang, W., Li, S., Huo, Y., & Lyu, M. R. (2024). Automatically Generating UI Code from Screenshot: A Divide-and-Conquer-Based Approach. *arXiv Preprint arXiv:2406.16386*. https://arxiv.org/abs/2406.16386

u, Y., Li, M., Cui, L., Huang, S., Wei, F., & Li, M. (2020). LayoutLM: Pre-Training of Text and Layout for Document Image Understanding. In *Proceedings of KDD*, 1192–1200. https://doi.org/10.1145/3394486.3403183

Yin, P., & Neubig, G. (2017). A Syntactic Neural Model for General-Purpose Code Generation. In *Proceedings of ACL*, 440–450. https://doi.org/10.18653/v1/P17-1041

Zhang, K., & Shasha, D. (1989). Simple Fast Algorithms for the Editing Distance Between Trees and Related Problems. *SIAM Journal on Computing, 18*(6), 1245–1262. https://doi.org/10.1137/0218062