SPRITE: From Static Mockups to Engine-Ready Game UI

Yunshu Bai
Shanghai University
Shanghai, China
baisherrywhite@shu.edu.cn
MiAO Worlds
Singapore, Singapore
cielobai@miao.company

RuiHao Li
MiAO Worlds
Singapore, Singapore
rioli@miao.company

Hao Zhang
MiAO Worlds
Singapore, Singapore
haozhang@miao.company

Chien Her Lim
MiAO Worlds
Singapore, Singapore
limchienher@miao.company

Ming Yan*
MiAO Worlds
Singapore, Singapore
mingyan@miao.company

Mengtian Li*
Shanghai University
Shanghai, China
mtli@shu.edu.cn

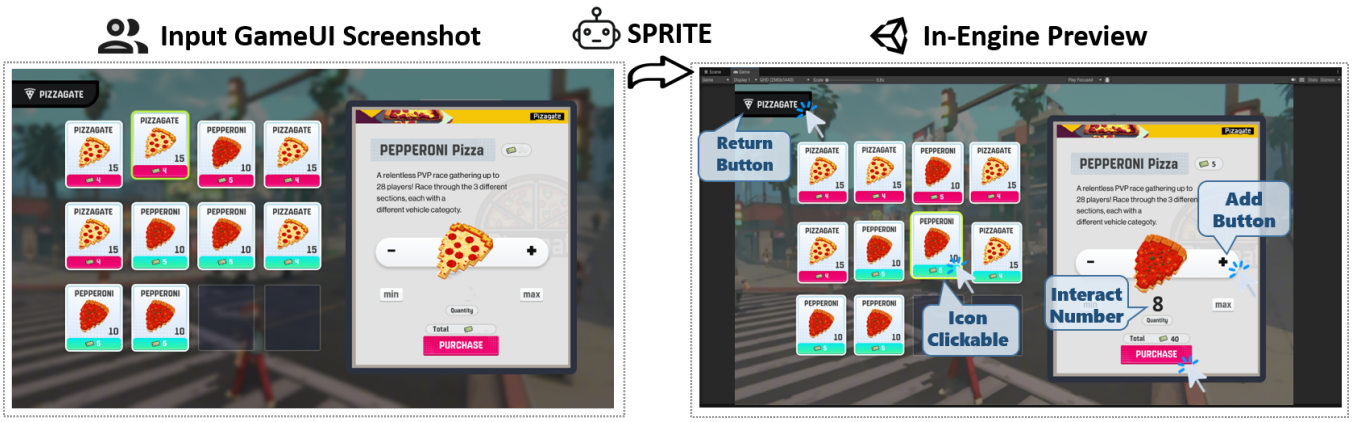

Figure 1: The SPRITE System.Transforming a raw GameUI screenshot (left) into editable engine assets (right). Unlike standard app interfaces, Game UIs are characterized by highly stylized aesthetics, irregular geometries, and complex visual hierarchies, making manual reconstruction a labor-intensive task that requires specialized expertise. By automating this reconstruction, SPRITE effectively bridges the gap between artistic vision and engine implementation to enable rapid gameplay iteration.

Abstract

Game UI implementation requires translating stylized mockups into interactive engine entities. However, current "Screenshot-to-Code" tools often struggle with the irregular geometries and deep visual hierarchies typical of game interfaces. To bridge this gap, we introduce SPRITE, a pipeline that transforms static screenshots into editable engine assets. By integrating Vision-Language Models (VLMs) with a structured YAML intermediate representation, SPRITE explicitly captures complex container relationships and non-rectangular layouts. We evaluated SPRITE against a curated Game UI benchmark and conducted expert reviews with professional developers to assess reconstruction fidelity and prototyping efficiency. Our findings demonstrate that SPRITE streamlines development by automating tedious coding and resolving complex nesting. By facilitating rapid in-engine iteration, SPRITE effectively blurs the boundaries between artistic design and technical implementation in game development.Project page: https://baiyunshu.github.io/sprite.github.io/

CCS Concepts

• **Human-centered computing** → **Systems and tools for interaction design**; **User interface programming**.

Keywords

Game UI, Automated Refactoring, Vision-Language Models, Generative AI

1 Introduction

Developing a Game User Interface (UI) is fundamentally an act of translation: developers must mentally map highly stylized, artistic visions into the rigid, logical structures of a game engine. Unlike utilitarian web or mobile apps, game interfaces serve as immersive narrative devices, often employing irregular geometries, diegetic elements, and deep visual hierarchies to sustain the player's suspension of disbelief. However, the cognitive load of this translation is immense. As with other forms of craftsmanship, the gap between visualizing a design and implementing it in an engine remains a

*Corresponding authors.

persistent bottleneck. Developers are currently forced to manually slice, measure, and reconstruct assets—a laborious "design-to-engine" grind that stifles creative iteration and often leads to visual discrepancies [30].

While "Screenshot-to-Code" technologies have shown promise in automating this process for standard web and mobile domains [3, 9], they operate on a fundamental mismatch when applied to games. Mainstream approaches, ranging from seminal Deep Learning models to recent methods based on Large Language Models (LLMs) such as GPT-4V and Gemini [26, 37], are primarily trained on standardized datasets like RICO and PubLayNet [8, 38]. These models assume a world of rectangular DOM elements and flow-based layouts. Attempting to adapt these web-centric models with modest extensions falls short because game engines rely on absolute spatial coordinates, deep custom scene graphs, and overlapping alpha-channeled sprites that lack direct HTML/CSS equivalents. Game UIs further defy this structural logic, characterized by highly stylized, non-standard widgets [5] and intricate, custom hierarchies that fundamentally diverge from the responsive logic of web interfaces [30]. Consequently, simply generating HTML code is insufficient for game engines, which require proprietary formats (e.g., Unity UXML) and precise spatial coordinates. Bridging this gap to bring the efficiency of automated reconstruction to the highly stylized, structurally complex domain of Game UI development thus presents a significant technical challenge.

Towards addressing this semantic and structural gap, we created **SPRITE**[1], a novel pipeline that operationalizes the translation from static mockups to engine-native assets. We postulated that by combining the semantic reasoning of Vision-Language Models (VLMs) with the precise localization of 2D foundation models, we could automate this reconstruction without sacrificing artistic fidelity. SPRITE employs a coarse-to-fine perception pipeline to parse non-standard elements and utilizes a YAML-based intermediate representation as a structural scaffold. This scaffold enforces hierarchical consistency, acting as a bridge that ensures visual hierarchies are precisely translated into the proprietary formats required by game engines. To evaluate SPRITE, we curated a professional-grade benchmark and conducted expert reviews with experienced developers. Our findings indicate that SPRITE's structure-aware workflow not only accelerates the iteration loop for professional game developers by eliminating manual implementation bottlenecks, but also lowers the technical barrier for non-specialists. By automating the translation of irregular assets and complex hierarchies, SPRITE empowers a broader spectrum of creators to engage in rapid, in-engine prototyping.

In summary, our contributions are as follows:

- **Instant UI Refactoring System:** SPRITE, a training-free pipeline leveraging a coarse-to-fine perception pipeline to instantly transform static screenshots into editable, engine-native assets.
- **Intermediate Bridge (Image-to-Engine):** We introduce a structure-preserving YAML schema as a semantic bridge, enforcing hierarchical consistency during code synthesis.
- **Professional-Grade Benchmark:** A high-fidelity dataset derived from industrial production environments, enabling rigorous evaluation of layout accuracy and usability against expert standards.

[1]SPRITE stands for **S**creenshot **P**arsing and **R**econstruction of **I**nterfaces via **T**raining-free **E**ngineering. Additionally, the name alludes to the "Sprite" asset type, a fundamental 2D graphic primitive used in the Unity engine.

2 Related Work

2.1 AI-Driven UI-to-Code Generation

While code generation from natural language has advanced rapidly [18, 19], generating executable code directly from visual inputs remains a challenging frontier. Early approaches relied on heuristic computer vision techniques [22] or deep learning encoders [3, 28, 35], but the field has recently shifted towards Multimodal Large Language Model (MLLM)-driven pipelines. To mitigate common MLLM hallucinations—such as element misclassification and layout misalignment—recent strategies employ divide-and-conquer prompting [6, 31, 33, 39] and domain-specific fine-tuning [13, 17, 36]. However, despite satisfactory performance in standardized web and mobile environments, these methods often yield simplistic outputs when applied to complex game scenarios [26]. Our approach addresses this limitation by specifically targeting the unique challenges of game interface reconstruction: irregular widget geometries and deep, custom scene graphs that remain intractable for standard layout analysis tools.

2.2 Fragmentation in Game UI/UX Engineering

Existing approaches typically bifurcate into visual layout generation versus logical structure correspondence. Visual-centric research [10, 12], including diffusion models [32], synthesizes high-fidelity aesthetics but often treats layouts as flattened images, overlooking the deep view hierarchies essential for engines. Conversely, logic-centric methods [24, 34] excel at structural matching but often sacrifice the pixel-perfect fidelity needed for production. Recent efforts attempt to bridge these streams. Web-based MLLM tools [21, 26] lack the structural rigidity required for games, while AutoGameUI [30] relies on often-unavailable paired design files. In contrast, SPRITE achieves direct reconstruction from a screenshot, generating engine-native assets without pre-aligned data.

2.3 Data Constraints and the Shift to Training-Free Alignment

The efficacy of UI modeling has traditionally relied on the *supervised learning* paradigm, training Transformer-based models to map latent representations to layouts [10, 12]. However, this approach is fundamentally bounded by data availability. Public datasets like RICO [8] and PubLayNet [38] provide only flattened RGB screenshots, lacking the raw RGBA layers and complex occlusion relationships inherent in game rendering pipelines. To bypass this scarcity, earlier studies employed *heuristic optimization* [4, 14, 15] (e.g., integer programming [7, 34]) to align visual elements with structures. While these training-free methods avoid data dependencies, their generalization is limited by rigid, handcrafted constraints. Addressing these limitations, our work adopts a *training-free* VLM-driven approach. Instead of relying on brittle heuristics or supervised metric learning [23, 24], we leverage the inherent semantic reasoning of state-of-the-art MLLMs [1, 2]—a zero-shot paradigm recently

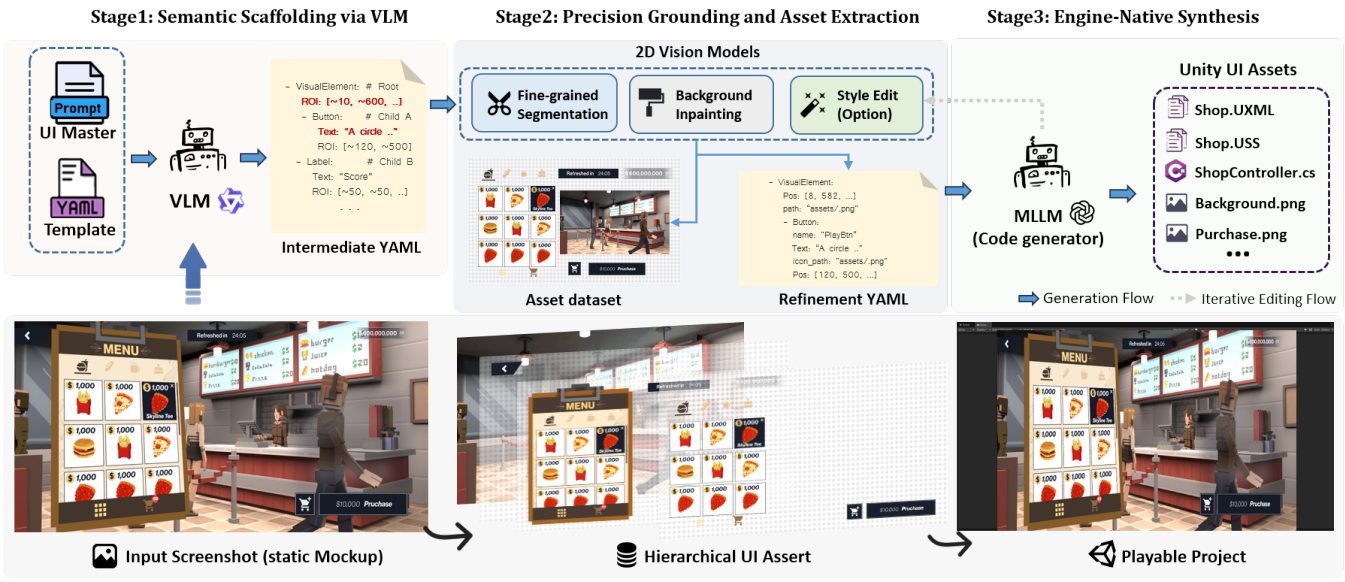

Figure 2: SPRITE Our system transforms mockups into engine assets via three stages: (1) Semantic Scaffolding, VLM infers a schema-guided hierarchical YAML; (2) Precision Grounding, utilizing 2D models for pixel-perfect extraction and geometric calibration; and (3) Engine-Native Synthesis, where an MLLM generates executable UXML/USS code and interaction logic.

proven effective in other game domains like procedural 3D map generation [11] to achieve precise alignment between visual pixels and engine-native structures, effectively bypassing the bottleneck of large-scale data annotation.

3 SPRITE System

3.1 Design Goals

SPRITE aims to bridge the fundamental semantic gap between static visual intent and functional game implementation. Informed by the "design-to-engine" bottlenecks identified in prior formative studies [30], we formulate three design goals to guide our system:

DG1: Fidelity Beyond Pixels: Semantic & Structural Recovery. Game UIs are characterized by irregular geometries and diegetic elements that defy standard web-based DOM logic. Therefore, the system must go beyond mere visual replication. It must accurately interpret the *semantics* of non-standard assets and reconstruct their deep, nested hierarchies, ensuring the output is a logically sound scene graph ready for engine interaction.

DG2: Zero-Shot Generalizability via Training-Free Architecture. To serve the diverse game development community, the system must be agnostic to visual styles without requiring costly domain-specific retraining. By leveraging a coarse-to-fine perception pipeline and a modular YAML intermediate representation, SPRITE aims to provide a robust, generalized bridge that adapts to various genres out-of-the-box.

DG3: Dual-Audience Empowerment: Accessibility & Augmentation. Recognizing the varying technical expertise within the community, SPRITE is designed to support a spectrum of user needs:

- *Accessible Creation for Novices:* Lowering the technical barrier to entry, enabling non-programmers to instantly transform sketches into functional prototypes.
- *Augmenting Workflows for Professionals:* Eliminating the tedious "grunt work" of pixel-level manual slicing and layout scaffolding, thereby freeing experts to focus on complex interaction logic and creative iteration.

3.2 The SPRITE Pipeline

SPRITE leverages a training-free, "coarse-to-fine" pipeline to bridge the gap between static screenshots and functional game engines. **As illustrated in Figure 2**, the pipeline decomposes complex UI reconstruction into three structured stages, minimizing manual labor while ensuring high-fidelity output (DG1, DG2).

3.2.1 Stage 1: Semantic Scaffolding via VLM. To handle the non-standard and highly stylized nature of game UIs, SPRITE employs a **schema-guided reasoning** strategy. We utilize a pre-defined YAML template that mirrors the hierarchical logic of engine formats (e.g., UXML), serving as the structural contract between visual pixels and code. Driven by an *Expert-Persona* prompt ("UI Master"), the VLM scans the screenshot to populate this template. It identifies functional components (e.g., HUDs, progress bars) and maps them onto a nested structure, constructing an initial **semantic scene graph** (DG2).

This intermediate representation ensures that high-level logical consistency is established before the system attempts pixel-level processing. To optimize this scaffolding, we explicitly adopt YAML instead of JSON for two strategic reasons: (1) **Token Efficiency:** YAML eliminates redundant syntactic overhead (e.g., extensive braces), reducing token consumption by approximately 20–30% and allowing the VLM to process denser layouts; (2) **UXML Semantic**

Alignment: YAML's indentation-based nesting naturally mirrors the structure of Unity's UXML, ensuring structural isomorphism for downstream engine synthesis. See the template in Figure 3.

```yaml
meta:
  img_path: "./input/ui_screenshot.png"
  canvas_size: [1920, 1080]
root:
  name: "ScreenCanvas"
  type: "VisualElement"
  layout: { display: "flex", width: "100%", height:
       "100%" }
  children:
    - name: "InventoryPanel"
      type: "VisualElement"
      layout: { position: "absolute", left: "10%",
           top: "10%" }
      seg_req:
        prompt: "dark translucent inventory panel
             background"
        roi: [190, 100, 1500, 900]
      seg_result: null
      children:
        - name: "TitleHeader"
          type: "VisualElement"
          seg_req:
            prompt: "metal header bar with rivets"
        - name: "ItemSlot_01"
          type: "Button"
          seg_req:
            prompt: ...
```

Figure 3: YAML template structure for UI scaffolding.

3.2.2 Stage 2: Precision Grounding and Asset Extraction. While VLMs excel at semantics, they often struggle with spatial precision. To mitigate this, SPRITE integrates 2D foundation models for geometric refinement (DG1). Guided by the Regions of Interest (ROI) defined in the YAML, the system performs **fine-grained segmentation** to extract standalone assets with alpha channels. Crucially, to ensure "engine-ready" completeness, an inpainting model performs **occlusion recovery**, filling background gaps left by extracted layers. The system then backfills the YAML template with precise bounding box coordinates and asset paths, effectively transitioning the scaffold from "fuzzy semantics" to "exact geometry".

3.2.3 Stage 3: Engine-Native Synthesis and Output. In the final stage, an LLM translates the calibrated YAML into executable technical assets. Treating the YAML as a structured specification, the model synthesizes Unity-native code (UXML/USS) that strictly adheres to the intended hierarchy. Furthermore, by interpreting semantic labels, the system infers interactive **affordances** (e.g., button hover states), enabling immediate prototyping. This process culminates in a **dual-value output** (DG3): providing a "no-code" gateway for novices to instantly realize their visions, while delivering an editable, layered "expert scaffold" that accelerates professional development workflows.

3.3 Implementation Details

SPRITE is architected as a modular, training-free pipeline that orchestrates a suite of state-of-the-art foundation models.

```
Act as an expert Game UI developer and systems architect
     . Perform a multi-level structural analysis of the
     provided game screenshot to synthesize a functional
      and hierarchical UI scaffold.

Include TWO types of elements:
- CONTAINER PANELS (parent): Popup dialogs, or boxes
     that CONTAINS buttons.
- INTERACTIVE ELEMENTS (child): Clickable buttons, icons
     , or sliders inside panels.

For each element, output in YAML format with the
     following structure:
- label: "component_name"
  bbox_2d: [x1, y1, x2, y2]
  seg_prompt: "Visual description for SAM segmentation"
  parent: null or "parent_label"

CONSTRAINTS: Container panels must have "parent: null".
     Elements INSIDE a panel must refer to the exact
     label of their parent. The seg_prompt must describe
      color, shape, and style to guide SAM. Output ONLY
     the YAML list.
```

Figure 4: System Prompt: UI Master Persona

3.3.1 Prompt Engineering and Visual Perception. For high-level semantic parsing and coarse component identification, we employ **Qwen3-VL** [2]. This initial parsing is driven by a carefully crafted system prompt (the "UI Master Persona"). Our prompt design follows three core rationales: (1) **Functional Decoupling** to force the VLM to filter aesthetic noise and isolate core UI logic; (2) **Structural Formalization** to mandate a `parent` field, transforming flat detection into a hierarchical scene graph; and (3) **Downstream Synergy** to generate descriptive visual cues that guide subsequent 2D foundation models. The core instruction is illustrated in Figure 4.

3.3.2 Geometric Extraction and Engine Synthesis. The semantic layer is augmented by a geometric stack: **GroundingDINO** [20] performs open-vocabulary detection based on the VLM's visual cues, while the **Segment Anything Model (SAM2)** [25] executes precise, pixel-level mask generation. To ensure assets are production-ready, **LaMa** [29] handles background inpainting, resolving occlusion artifacts to produce clean sprites. Finally, the core synthesis engine uses **GPT-5** [27] and **Claude 4.5 Sonnet** [1] to translate the structured YAML into Unity UXML and USS files. We apply few-shot prompting and chain-of-thought reasoning to maintain strict hierarchical consistency. Additionally, we integrate **FLUX** [16] to empower users with generative editing capabilities for stylized asset refinement.

4 Evaluation

4.1 GAMEUI Benchmark

To evaluate SPRITE in realistic scenarios, we curated the GAMEUI Benchmark, a specialized dataset explicitly targeting the structural complexity of gaming (Figure 6). Specifically, the benchmark comprises hundreds of meticulously selected interfaces sampled from multiple distinct game genres (e.g., RPG, FPS, Strategy, Casual). These interfaces were chosen to ensure high diversity in layout complexity (ranging from simple main menus to dense inventory

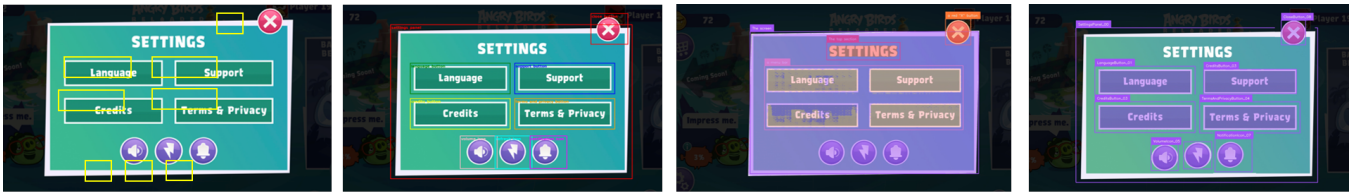

Figure 5: Qualitative Comparison. While VLMs (a-b) are limited to bounding boxes and the baseline (c) suffers from fragmentation, SPRITE (d) leverages semantic scaffolding to extract pixel-perfect assets for irregular shapes.

grids) and artistic styles. This rigorous selection process ensures that unlike general-purpose datasets, this collection features high-fidelity, production-level interfaces spanning diverse functional modules (e.g., HUDs, Settings, Inventories).

Each entry serves as a comprehensive **ground truth ecosystem** rather than a simple screenshot, comprising:

- **Figma-exported JSONs:** Detailing expert-defined structural hierarchies and interaction metadata.
- **Segmented Sprites:** High-quality extracted image components alongside corresponding text metadata.
- **Hand-authored Unity UXML/ USS:** Professional-grade templates serving as the synthesis gold standard.

As an ongoing project, the benchmark is **continuously expanding** to encompass more diverse UI functionalities. By processing these complex, non-standard layouts, we verified SPRITE's ability to maintain **structural integrity (DG2)** and generate engine-ready scaffolds across a wide range of functional game UI scenarios.

4.2 System Validation and Expert Evaluation

4.2.1 Qualitative Analysis of Asset Extraction. We performed a qualitative comparison against state-of-the-art VLM detectors and zero-shot segmentation baselines. As demonstrated in Figure 5, existing methods struggle with the unique visual language of games in two distinct ways. First, standard VLM approaches (Figure 5a-b) are inherently limited to coarse bounding boxes, making them incapable of isolating assets with complex, non-rectangular boundaries. Second, while pixel-level baselines (Figure 5c) attempt to extract exact shapes, they frequently suffer from severe fragmentation. Without strong semantic constraints to resolve referencing ambiguities, these models over-rely on the latent biases of their pixel-level training distributions, fracturing cohesive UI elements into illogical, unusable debris. In contrast, SPRITE (Figure 5d) overcomes both limitations by grounding the extraction process in **semantic scaffolding**. This top-down structural guidance allows our system to reliably extract pixel-perfect assets even for highly irregular and stylized shapes, ensuring the outputs are visually clean, logically grouped, and immediately engine-ready (DG1).

4.2.2 Expert Review and User Study. To assess SPRITE's practical utility, we conducted an expert evaluation with three senior UI/UX designers using the GAMEUI Benchmark. During these sessions, the experts interactively tested the functional prototypes and conducted a rigorous audit of the auto-segmented UI assets, structural hierarchies, and underlying UXML source files. Evaluated on a 10-point Likert scale (1 = entirely unusable, 10 = production-ready), the system achieved strong ratings in **Visual Fidelity (8.5/10)** and **Hierarchical Logic (8.0/10)**, with the designers confirming that the reconstructed assets closely mirrored industry standards. However, **Interaction Accuracy (7.0/10)** received a comparatively lower score, prompting a deeper analysis of the system's interactive scope and boundary conditions.

4.2.3 Interaction Scope and Failure Cases. Currently, SPRITE successfully infers basic interactive affordances such as button hover states, toggles, and simple click events derived from semantic labels. However, the system encounters typical failure modes when processing heavily overlapping translucent elements or highly abstract, diegetic UIs where visual boundaries are ambiguous. Additionally, complex temporal logic (e.g., drag-and-drop mechanics or the non-linear motion synthesis required for production polish) remains beyond the current generation scope, directly explaining the lower interaction score observed in our expert review. Despite these boundaries, experts acknowledged the efficiency of the automated workflow, validating SPRITE as a robust prototyping tool and explicitly motivating our future focus on dynamic simulation.

5 Discussion and Future Work

While SPRITE successfully operationalizes static reconstruction, we envision three critical trajectories to further bridge the design-to-engine gap:

From Static Visuals to Dynamic Interactivity. Addressing the limitation of static screenshots, future iterations of SPRITE will transition towards multi-frame temporal reasoning by leveraging advanced Video-LLMs. By analyzing short gameplay clips or UI interaction recordings, the system could autonomously extract complex temporal logic, such as non-linear animation curves, state-machine transitions, and conditional visual feedback loops (e.g., hover-to-click effects). This advancement will fundamentally shift the automated reconstruction paradigm from merely capturing the structural "look" of the interface to computationally replicating its dynamic "feel", ultimately generating ready-to-use engine animation controllers and event scripts directly from video references.

Disentanglement for Generative Design. We aim to evolve SPRITE into a parameterized generative design system by computationally disentangling the core structural hierarchy (UXML) from its visual presentation (USS). Mapping these decoupled components into a latent semantic space will empower developers to perform prompt-driven, global restyling—such as instantly swapping color

Figure 6: Visual representation of the GAMEUI Benchmark gallery. These representative samples demonstrate the system's ability to faithfully reconstruct a wide array of challenging layouts across different game genres. Ranging from high-density RPG inventory grids to complex, multi-level navigation systems, this gallery highlights the functional diversity and zero-shot generalizability explicitly targeted by SPRITE.

palettes and asset motifs—without altering the underlying functional logic. This capability will facilitate rapid A/B testing, enable procedural generation of UI variants for live-ops events, and drastically reduce the friction of continuous aesthetic iteration.

Human-AI Collaboration and Workflow Cognition. To address the socio-technical impact of automated UI reconstruction, our future work will investigate how tools like SPRITE reshape the division of labor between artists and programmers. We plan to conduct longitudinal studies to explore how users calibrate trust in automated outputs, what new breakdowns emerge during structural refactoring, and how mixed-initiative co-creation ultimately transforms the cognitive load of game development workflows.

6 Conclusion

In this work, we presented SPRITE, a training-free pipeline that helps bridge the Gulf of Execution between static artistic vision and engine-native implementation. By harmonizing VLM-based semantic scaffolding with the geometric precision of 2D foundation models and LLM code generation, SPRITE effectively automates the translation of flat mockups into functional Unity assets. Our evaluation via the GAMEUI Benchmark confirms that SPRITE not only maintains high structural integrity but also significantly streamlines the development workflow. By substantially reducing the tedious overhead of manual asset slicing and assembly, SPRITE delivers a dual impact: it lowers the technical barrier for novices to engage in rapid game creation, while enabling professional developers to better focus on complex interaction logic. Ultimately, SPRITE helps redefine the human-AI relationship in game engineering, shifting the prevailing paradigm from manual pixel reconstruction toward a collaborative framework of high-level, semantic co-creation.

References

[1] Anthropic. 2025. Introducing Claude Sonnet 4.5. https://www.anthropic.com/news/claude-sonnet-4-5. Accessed: 2026-01-20.

[2] Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xiong-Hui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Rongyao Fang, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Li Ying Meng, Xuancheng Ren, Xin yi Ren, Sibo Song, Yu-Chen Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yihe Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Botao Zheng, Humen Zhong, Jingren Zhou, Fanxi Zhou, Jingren Zhou, Yuanzhi Zhu, and Keming Zhu. 2025. Qwen3-VL Technical Report. *ArXiv* abs/2511.21631 (2025), 1–42.

[3] Tony Beltramelli. 2018. pix2code: Generating code from a graphical user interface screenshot. In *Proceedings of the ACM SIGCHI symposium on engineering interactive computing systems*. Association for Computing Machinery, New York, NY, USA, 1–6.

[4] Sacha Brisset, Romain Rouvoy, Lionel Seinturier, and Renaud Pawlak. 2021. Erratum: Leveraging Flexible Tree Matching to Repair Broken Locators in Web

Automation Scripts. *ArXiv* abs/2106.04916 (2021), 1–34.

[5] Sara Bunian, Kai Li, Chaima Jemmali, Casper Harteveld, Yun Fu, and Magy Seif Seif El-Nasr. 2021. VINS: Visual Search for Mobile User Interface Design. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 423, 14 pages. doi:10.1145/3411764.3445762

[6] Zhixiang Chi, Yanan Wu, Li Gu, Huan Liu, Ziqiang Wang, Yang Zhang, Yang Wang, and Konstantinos N. Plataniotis. 2025. Plug-in Feedback Self-adaptive Attention in CLIP for Training-free Open-Vocabulary Segmentation. *ArXiv* abs/2508.20265 (2025), 1–42.

[7] Niraj Ramesh Dayama, Simo Santala, Lukas Brückner, Kashyap Todi, Jingzhou Du, and Antti Oulasvirta. 2021. Interactive Layout Transfer. In *Proceedings of the 26th International Conference on Intelligent User Interfaces* (College Station, TX, USA) *(IUI '21)*. Association for Computing Machinery, New York, NY, USA, 70–80. doi:10.1145/3397481.3450652

[8] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. 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* (Québec City, QC, Canada) *(UIST '17)*. Association for Computing Machinery, New York, NY, USA, 845–854. doi:10.1145/3126594.3126651

[9] Zhen Feng, Jiaqi Fang, Bo Cai, and Yingtao Zhang. 2021. GUIS2Code: A Computer Vision Tool to Generate Code Automatically from Graphical User Interface Sketches. In *Proceedings of the 30th International Conference on Artificial Neural Networks (ICANN)* (Bratislava, Slovakia). Springer-Verlag, Berlin, Heidelberg, 53–65. doi:10.1007/978-3-030-86365-4_5

[10] Kamal Gupta, Justin Lazarow, Alessandro Achille, Larry S Davis, Vijay Mahadevan, and Abhinav Shrivastava. 2021. Layouttransformer: Layout generation and completion with self-attention. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE/CVF, Montreal, QC, Canada, 1004–1014.

[11] Lim Chien Her, Ming Yan, Yunshu Bai, Ruihao Li, and Hao Zhang. 2025. Zero-shot 3D Map Generation with LLM Agents: A Dual-Agent Architecture for Procedural Content Generation. *arXiv preprint arXiv:2512.10501* abs/2512.10501 (2025), 1–12.

[12] Naoto Inoue, Kotaro Kikuchi, Edgar Simo-Serra, Mayu Otani, and Kota Yamaguchi. 2023. LayoutDM: Discrete Diffusion Model for Controllable Layout Generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE/CVF, Vancouver, BC, Canada, 10167–10176.

[13] Yilei Jiang, Yaozhi Zheng, Yuxuan Wan, Jiaming Han, Qunzhong Wang, Michael R. Lyu, and Xiangyu Yue. 2025. ScreenCoder: Advancing Visual-to-Code Generation for Front-End Automation via Modular Multimodal Agents. *ArXiv* abs/2507.22827 (2025), 1–20.

[14] Harold W. Kuhn. 1955. The Hungarian method for the assignment problem. *Naval Research Logistics (NRL)* 2, 1-2 (1955), 83–97.

[15] Ranjitha Kumar, Jerry O. Talton, Salman Ahmad, and Scott R. Klemmer. 2011. Bricolage: example-based retargeting for web design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vancouver, BC, Canada) *(CHI '11)*. Association for Computing Machinery, New York, NY, USA, 2197–2206. doi:10.1145/1978942.1979262

[16] Black Forest Labs, Stephen Batifol, A. Blattmann, Frederic Boesel, Saksham Consul, Cyril Diagne, Tim Dockhorn, Jack English, Zion English, Patrick Esser, Sumith Kulal, Kyle Lacey, Yam Levi, Cheng Li, Dominik Lorenz, Jonas Muller, Dustin Podell, Robin Rombach, Harry Saini, Axel Sauer, and Luke Smith. 2025. FLUX.1 Kontext: Flow Matching for In-Context Image Generation and Editing in Latent Space. *ArXiv* abs/2506.15742 (2025), 1–19.

[17] Hugo Laurençon, Léo Tronchon, and Victor Sanh. 2024. Unlocking the conversion of Web Screenshots into HTML Code with the WebSight Dataset. *ArXiv* abs/2403.09029 (2024), 1–9.

[18] Triet Huynh Minh Le, Hao Chen, and Muhammad Ali Babar. 2020. Deep Learning for Source Code Modeling and Generation. *ACM Computing Surveys (CSUR)* 53 (2020), 1 – 38.

[19] Raymond Li, Loubna Ben Allal, Yangtian Zi, et al. 2023. StarCoder: may the source be with you! *Trans. Mach. Learn. Res.* 2023 (2023), 1–55.

[20] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu, et al. 2023. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. *arXiv preprint arXiv:2303.05499* abs/2303.0549 (2023), 1–33.

[21] Yuwen Lu, Alan Leung, Amanda Swearngin, Jeffrey Nichols, and Titus Barik. 2025. Misty: UI Prototyping Through Interactive Conceptual Blending. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. Association for Computing Machinery, New York, NY, USA, Article 1108, 17 pages. doi:10.1145/3706598.3713924

[22] Tuan Anh Nguyen and Christoph Csallner. 2015. Reverse Engineering Mobile Application User Interfaces with REMAUI (T). In *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, Lincoln, NE, USA, 248–259.

[23] Akshay Gadi Patil, Omri Ben-Eliezer, Or Perel, and Hadar Averbuch-Elor. 2020. READ: Recursive Autoencoders for Document Layout Generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, Seattle, WA, USA, 2316–2325.

[24] Akshay Gadi Patil, Manyi Li, Matthew Fisher, Manolis Savva, and Hao Zhang. 2021. LayoutGMN: Neural Graph Matching for Structural Layout Similarity. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE/CVF, Nashville, TN, USA, 11043–11052.

[25] Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya K. Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloé Rolland, Laura Gustafson, Eric Mintun, Junting Pan, Kalyan Vasudev Alwala, Nicolas Carion, Chao-Yuan Wu, Ross B. Girshick, Piotr Doll'ar, and Christoph Feichtenhofer. 2024. SAM 2: Segment Anything in Images and Videos. *ArXiv* abs/2408.00714 (2024), 1–42.

[26] Chenglei Si, Yanzhe Zhang, Ryan Li, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. 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)*. Association for Computational Linguistics, Albuquerque, New Mexico, USA, 3956–3974.

[27] Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. 2025. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267* abs/2601.03267 (2025), 1–61.

[28] Davit Soselia, Khalid Saifullah, and Tianyi Zhou. 2023. Learning UI-to-Code Reverse Generator Using Visual Critic Without Rendering. *arXiv preprint arXiv:2305.14637* abs/2305.14637 (2023), 1–10.

[29] Roman Suvorov, Elizaveta Logacheva, Anton Mashikhin, Anastasia Remizova, Arsenii Ashukha, Aleksei Silvestrov, Naejin Kong, Harshith Goka, Kiwoong Park, and Victor Lempitsky. 2022. Resolution-robust large mask inpainting with fourier convolutions. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*. IEEE/CVF, Waikoloa, HI, USA, 2149–2159.

[30] Zhongliang Tang, Mengchen Tan, Fei Xia, Qingrong Cheng, Hao Jiang, and Yongxiang Zhang. 2024. AutoGameUI: Constructing High-Fidelity Game UIs via Multimodal Learning and Interactive Web-Based Tool. *arXiv preprint arXiv:2411.03709* abs/2411.03709 (2024), 1–9.

[31] Yuxuan Wan, Chaozheng Wang, Yi Dong, Wenxuan Wang, Shuqing Li, Yintong Huo, and Michael R. Lyu. 2024. Automatically Generating UI Code from Screenshot: A Divide-and-Conquer-Based Approach. *ArXiv* abs/2406.16386 (2024), 241–253.

[32] Jialiang Wei, Anne-Lise Courbis, Thomas Lambolais, Gérard Dray, and Walid Maalej. 2025. On AI-Inspired User Interface Design. *IEEE Software* 42, 3 (2025), 50–58. doi:10.1109/MS.2025.3536838

[33] Fan Wu, Cuiyun Gao, Shuqing Li, Xinjie Wen, and Qing Liao. 2025. MLLM-Based UI2Code Automation Guided by UI Layout Information. *Proceedings of the ACM on Software Engineering* 2 (2025), 1123 – 1145.

[34] Pengfei Xu, Yifan Li, Zhijin Yang, Weiran Shi, Hongbo Fu, and Hui Huang. 2022. Hierarchical Layout Blending with Recursive Optimal Correspondence. *ACM Transactions on Graphics (TOG)* 41 (2022), 1 – 15.

[35] Yong Xu, Lili Bo, Xiaobing Sun, Bin Li, Jing Jiang, and Wei Zhou. 2021. image2emmet: Automatic code generation from web user interface image. *Journal of Software: Evolution and Process* 33 (2021), 241–253.

[36] Hao Yang, Weijie Qiu, Ru Zhang, Zhou Fang, Ruichao Mao, Xiaoyu Lin, Maji Huang, Zhaosong Huang, Teng Guo, Shuoyang Liu, and Hai Rao. 2025. UI-UG: A Unified MLLM for UI Understanding and Generation. *ArXiv* abs/2509.24361 (2025), 1–16.

[37] Houston H Zhang, Tao Zhang, Baoze Lin, Yuanqi Xue, Yincheng Zhu, Huan Liu, Li Gu, Linfeng Ye, Ziqiang Wang, Xinxin Zuo, et al. 2025. Widget2Code: From Visual Widgets to UI Code via Multimodal LLMs. *arXiv preprint arXiv:2512.19918* 2512.19918 (2025), 1–25.

[38] Xu Zhong, Jianbin Tang, and Antonio Jimeno Yepes. 2019. Publaynet: largest dataset ever for document layout analysis. In *2019 International conference on document analysis and recognition (ICDAR)*. IEEE, Sydney, NSW, Australia, 1015–1022.

[39] Ti Zhou, Yanjie Zhao, Xinyi Hou, Xiaoyu Sun, Kai Chen, and Haoyu Wang. 2025. DeclarUI: Bridging Design and Development with Automated Declarative UI Code Generation. *Proceedings of the ACM on Software Engineering* 2 (2025), 219 – 241.