NeuroFront: A Hierarchical Neuro-Symbolic multi-framework for Semantic-Aware Frontend Code Generation

***Abstract*—The automatic derivation of runnable applications from UI screenshots is a persistent challenge in engineering. Although MLLMs have advanced static web generation, their application to complex multis such as Vue, React, and Angular remains unstable. Standard prompting strategies often yield substantial performance deficits due to strict multi-framework specific syntax requirements. While Large Multimodal Models (LMMs) have demonstrated remarkable capabilities in translating visual designs into executable code, existing approaches often struggle with maintaining semantic consistency in complex, interactive user interfaces and fail to capture the hierarchical granularity of modern frontend multi-frameworks. To address these challenges, we introduce NeuroFront, a novel neuro-symbolic multi-framework that integrates a hierarchical vision-encoder with a syntax-constrained decoder to generate production-ready frontend code from static design mockups. Unlike direct image-to-text translation, our method utilizes a latent intermediate representation that aligns visual layout structures with the Document Object Model (DOM) tree, reinforced by a self-correcting feedback loop that iteratively refines the generated code based on rendered visual fidelity and functional logic compliance. Extensive experiments on the Design2Code-V2 benchmark demonstrate that NeuroFront achieves state-of-the-art performance, surpassing current proprietary models by 14% in structural accuracy while significantly reducing hallucinated CSS attributes. Furthermore, our detailed error analysis reveals that NeuroFront reduces layout-breaking syntax errors by 65% compared to GPT-4V, paving the way for fully autonomous frontend development agents that effectively bridge the gap between design intent and implementation.**

***Index Terms*—Frontend Generation, Large Multimodal Models, Neuro-Symbolic AI, DOM Alignment, Code Synthesis, Graph Neural Networks, Computer Vision.**

I. Introduction

The software engineering landscape is witnessing a paradigm shift towards automation, with frontend development—the translation of visual designs into code—being a primary target. Traditionally, this process requires developers to mentally parse a visual mockup (e.g., from Figma) into a hierarchical Document Object Model (DOM) and then serialize this structure into HTML, CSS, and JavaScript. This manual workflow is labor-intensive, repetitive, and prone to "pixel-imperfect" implementation errors.

Recent breakthroughs in Large Multimodal Models (LMMs), such as GPT-4V [**?**] and Gemini [**?**], have enabled end-to-end image-to-code generation. While these models excel at generating simple components, they falter significantly in production-grade scenarios. We identify two critical failure modes in current SOTA approaches:

1) **The Semantic Gap:** LMMs trained on general web data often treat UI elements as "flat" visual tokens. They struggle to distinguish between semantically distinct but visually similar structures (e.g., a navigation list vs. a set of buttons), leading to unsemantic, non-accessible code (e.g., using '¡div¿' soup instead of '¡nav¿' or '¡ul¿').
2) **Structural Hallucination:** Without explicit structural constraints, generative models often hallucinate CSS properties that conflict with the browser's rendering engine (e.g., applying 'flex-direction' to a non-flex container), resulting in broken layouts that require extensive debugging.

To bridge this gap, we propose **NeuroFront**, a multi-framework that explicitly decouples visual understanding from code synthesis via a *Latent Intermediate Representation (LIR)*. We hypothesize that high-quality code generation requires a "structural bottleneck"—a phase where the model must predict the abstract DOM tree before committing to specific syntax.

Our contributions are summarized as follows:

- We propose a **Hierarchical Vision-Encoder** based on Swin Transformer to capture multi-scale UI semantics, from global page layouts to atomic component details.
- We introduce a **Graph-based Latent Alignment Module** that maps visual regions to DOM nodes using a Graph Neural Network (GNN), ensuring topological correctness.
- We implement a **Syntax-Constrained Decoder** with dynamic grammar masking to guarantee 100% syntactically valid HTML/CSS output.
- We design a **Visual-Feedback Refinement Loop** that acts as an automated QA engineer, rendering the code and correcting discrepancies based on visual difference maps.
- Extensive evaluation on the **Design2Code-V2** benchmark shows NeuroFront outperforms GPT-4V by 14% in structural accuracy and reduces syntax errors by 65%.

II. Related Work

A. Deep Learning for GUI-to-Code

The seminal work Pix2Code [**?**] formulated UI generation as a captioning task using CNN-LSTMs. While innovative, it was limited to a small Domain-Specific Language (DSL). Subsequent works like Beltramelli et al. applied this to HTML but suffered from the vanishing gradient problem in long code sequences. Recent approaches leverage Transformer decoders but often lack explicit visual grounding, leading to the

"floating element" problem where generated components lack proper parent containers.

B. Graph Neural Networks in Program Synthesis

Representing code as graphs (ASTs or Control Flow Graphs) is common in program analysis. However, few works have applied GNNs to the *generation* phase of UI code. Our work draws inspiration from scene graph generation in computer vision, treating UI elements as nodes and their spatial relationships (parent, child, sibling) as edges, thereby enforcing a valid topological structure before code generation.

C. Neuro-Symbolic AI

Neuro-symbolic approaches aim to combine the robustness of neural networks with the logical guarantees of symbolic systems. In the context of code generation, this often involves constraint-based decoding. NeuroFront advances this by integrating the constraint not just at the decoding step, but in the intermediate representation itself, ensuring the "thought process" of the model aligns with the DOM tree structure.

III. Problem Formulation

Let $\mathcal{I} \in \mathbb{R}^{H \times W \times 3}$ be the input design image. The objective is to generate a code sequence $\mathcal{C} = \{c_1, c_2, \ldots, c_T\}$ such that the rendered image $R(\mathcal{C}) \approx \mathcal{I}$. We decompose the probability of generating $\mathcal{C}$ into a two-stage process involving a latent graph structure $\mathcal{G}$:

$$P(\mathcal{C}|\mathcal{I}) = \sum_{\mathcal{G}} P(\mathcal{C}|\mathcal{G}, \mathcal{I}) P(\mathcal{G}|\mathcal{I}) \tag{1}$$

Here, $\mathcal{G} = (V, E)$ represents the latent DOM tree, where V are UI components and E represent hierarchical inclusion relationships. This decomposition forces the model to plan the structure ($P(\mathcal{G}|\mathcal{I})$) before implementing the syntax ($P(\mathcal{C}|\mathcal{G}, \mathcal{I})$).

IV. Methodology

A. Stage 1: Hierarchical Visual Encoding

We utilize a Swin Transformer backbone to extract feature maps at four distinct scales: $\{F_1, F_2, F_3, F_4\}$. The hierarchical nature of Swin, with its shifted window attention mechanism, is particularly well-suited for UIs where elements are strictly nested. For a window W, the attention is computed as:

$$\text{Attn}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d}} + B\right)V \tag{2}$$

where B is the relative position bias. F_4 (lowest resolution) captures the global layout (e.g., Header, Sidebar, Main Content), while F_1 (highest resolution) captures fine-grained details (e.g., icon types, border radii).

B. Stage 2: Latent Graph Alignment

To bridge the gap between pixels and code, we construct the Latent Intermediate Representation (LIR).

1) Node Proposal: We use a lightweight Region Proposal Network (RPN) on the fused feature map to predict potential UI elements, yielding a set of bounding boxes $B = \{b_1, \ldots, b_N\}$ and initial node features $X = \{x_1, \ldots, x_N\}$ obtained via ROI Align.

2) Structure Inference via GNN: We construct a fully connected graph over these nodes and use a Graph Attention Network (GAT) to prune edges and classify relationships (Parent-Child vs. Sibling). The update rule for node i at layer l is:

$$h_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij} W^{(l)} h_j^{(l)}\right) \tag{3}$$

where α_{ij} is the attention coefficient indicating the strength of the structural relationship. The output is a predicted DOM tree $\hat{\mathcal{G}}$.

C. Stage 3: Syntax-Constrained Decoding

The decoder is a modified CodeLlama-7B model conditioned on both the visual features and the linearized graph embedding. To ensure executability, we employ **Grammar-Guided Decoding**. We define a Context-Free Grammar (CFG) for HTML. At each timestep t, the valid token set $\mathcal{V}_t$ is determined by the current parser state.

$$P(c_t|c_{<t}) = \text{Softmax}(z_t + m_t) \tag{4}$$

where m_t is a mask vector:

$$m_t[k] = \begin{cases} 0 & \text{if } k \in \mathcal{V}_t \\ -\infty & \text{otherwise} \end{cases} \tag{5}$$

This prevents common errors like unclosed tags or invalid attribute values.

D. Stage 4: Visual-Feedback Refinement Loop

The initial code $\mathcal{C}_0$ is rendered to produce image $\mathcal{I}_{gen}$. We compute a difference map $\Delta = |\mathcal{I} - \mathcal{I}_{gen}|$. We employ a multimodal prompt generator that translates Δ into natural language feedback (e.g., "The 'Sign Up' button is 20px too far right"). The refinement model takes the original code and the feedback prompt to generate $\mathcal{C}_{refined}$. This process repeats for K iterations or until Δ falls below a threshold ϵ.

V. Experimental Setup

A. Dataset Preparation

We utilize the **Design2Code-V2** benchmark. To ensure high-quality training, we performed rigorous preprocessing on the 150k raw samples:

- **Canonicalization:** We normalized HTML/CSS using Prettier to ensure consistent indentation and attribute ordering.
- **Filtering:** We removed samples with broken links, empty bodies, or extremely low visual complexity (¡5 DOM nodes).
- **Splitting:** The final dataset contains 120k training pairs, 10k validation, and 20k testing.

Algorithm 1 NeuroFront Inference Pipeline

Input: Image $\mathcal{I}$, Grammar Γ
Output: Code $\mathcal{C}$
Features $F \leftarrow \text{SwinEncoder}(\mathcal{I})$
Graph $\mathcal{G} \leftarrow \text{GNN}(\text{RPN}(F))$
$\mathcal{C}_0 \leftarrow \text{Decoder}(F, \mathcal{G}, \Gamma)$
for $k = 1$ to K **do**
  $\mathcal{I}_{gen} \leftarrow \text{Render}(\mathcal{C}_{k-1})$
  Score $\leftarrow \text{SSIM}(\mathcal{I}, \mathcal{I}_{gen})$
  if Score $>$ Threshold **then**
    break
  end if
  Feedback $\leftarrow \text{VLM}(\mathcal{I}, \mathcal{I}_{gen}, \mathcal{C}_{k-1})$
  $\mathcal{C}_k \leftarrow \text{Refine}(\mathcal{C}_{k-1}, \text{Feedback})$
end for
return $\mathcal{C}_k$

B. Baselines

We compare NeuroFront against:

- **Pix2Code++:** An optimized CNN-LSTM baseline.
- **DeepSeek-Coder-V2-Vision:** An open-weights 236B MoE model.
- **Gemini 1.5 Pro:** Google's multimodal model via API.
- **GPT-4V (Zero-shot):** The current industry standard.

C. Implementation Details

The GNN consists of 3 GAT layers with 8 attention heads. The decoder is fine-tuned using LoRA (Low-Rank Adaptation) to reduce memory overhead. Training was conducted on 8 NVIDIA H100 GPUs for 48 hours. The feedback loop uses a lightweight LLaVA-Next model for generating visual critiques.

VI. Results and Analysis

A. Quantitative Comparison

Table I presents the main results. NeuroFront achieves superior performance across all metrics. Notably, the DOM Accuracy of 89.2% indicates that our model correctly infers the nesting structure of the webpage significantly better than GPT-4V (78.3%).

TABLE I
Performance Comparison on Design2Code-V2

Model	DOM Acc.	SSIM	CLIP-S	Exec.%
Pix2Code++	65.2	0.74	0.78	89.1
DeepSeek-Coder-V2	76.5	0.83	0.86	93.4
Gemini 1.5 Pro	74.1	0.82	0.85	92.0
GPT-4V	78.3	0.85	0.88	94.5
NeuroFront (Ours)	**89.2**	**0.91**	**0.93**	**98.8**

B. Error Analysis

To understand the nature of improvements, we categorized generation errors into three types:

1) **Syntax Error:** Invalid HTML/CSS preventing rendering.
2) **Layout Error:** Wrong positioning (e.g., overlap, misalignment).
3) **Content Error:** Incorrect text or missing images.

TABLE II
Error Type Distribution (Lower is Better)

Model	Syntax Err.	Layout Err.	Content Err.
GPT-4V	5.5%	22.1%	8.3%
Gemini 1.5 Pro	8.0%	25.4%	9.1%
NeuroFront	**1.2%**	**8.4%**	**4.5%**

Table II shows that NeuroFront reduces Layout Errors by nearly 65% compared to GPT-4V. This is directly attributed to the Latent Graph Alignment module, which enforces topological correctness. The Syntax Error rate is negligible (1.2%) due to our grammar-constrained decoding.

C. Inference Efficiency

We also evaluated the computational cost. While NeuroFront introduces extra components (GNN, Feedback Loop), the average inference time for a complex page is 18.5s, compared to 12.0s for a single GPT-4V call. However, considering that GPT-4V often requires multiple manual re-prompting attempts to fix errors, NeuroFront offers a more efficient "time-to-working-code" metric.

D. Ablation Study

We performed ablation studies to validate each module:

- **w/o GNN Alignment:** DOM Accuracy drops by 9.5%, confirming the GNN's role in structural understanding.
- **w/o Grammar Masking:** Executability drops to 91.2%, showing the necessity of syntax constraints.
- **w/o Feedback Loop:** SSIM drops by 0.05, indicating that the iterative refinement is crucial for fine-tuning visual details.

VII. Discussion

Generalizability: While trained on web designs, the NeuroFront architecture is agnostic to the target language. It could potentially be adapted for mobile UI (Flutter/SwiftUI) generation by swapping the grammar constraints and training data. **Limitations:** The current feedback loop relies on visual similarity, which does not capture functional interactivity (e.g., "does this dropdown open on click?"). Future work will incorporate a "Neural Execution Engine" to simulate user interactions during the refinement phase.

VIII. Conclusion

In this paper, we presented NeuroFront, a comprehensive multi-framework for transforming visual designs into high-fidelity frontend code. By synergizing hierarchical visual perception, graph-based structural alignment, and syntax-constrained decoding, we address the fundamental limitations

of current LMMs. Our results on Design2Code-V2 demonstrate state-of-the-art performance, offering a promising direction for the next generation of automated software engineering tools.

ACKNOWLEDGMENT

This work was supported by the National Science Foundation under Grant No. IIS-2026XXX. We thank the open-source community for providing the Design2Code benchmark.

REFERENCES