Computers, Materials & Continua

Tech Science Press

https://doi.org/10.32604/cmc.2026.084499

ARTICLE

Vision-Based Frontend Extraction and LLM-Enhanced Web Honeypot Framework

Guan Yang[1], Shiyan Kang[1], Bo Chen[2,3] and Yu Wang[4,*]

[1]School of Computer Science, Zhongyuan University of Technology, Zhengzhou, China
[2]School of Mathematical Sciences, Shenzhen University, Shenzhen, China
[3]Guangdong Provincial Key Laboratory of Intelligent Information Processing, Shenzhen University, Shenzhen, China
[4]School of Software, Henan University of Engineering, Zhengzhou, China

*Corresponding Author: Yu Wang. Email: stonchor@163.com

Received: 23 April 2026; Accepted: 15 June 2026; Published: 13 August 2026

ABSTRACT: Web honeypots serve as foundational technologies for active deception, attracting attackers and extracting actionable threat intelligence. To address the challenges associated with manual and labor-intensive frontend construction, this paper presents the HFG framework, a security-oriented frontend generation framework designed for the large-scale deployment of heterogeneous Web-service decoy nodes. HFG utilizes a vision-to-code architecture integrating a PVT-CoT visual encoder, multi-scale adaptive fusion, visual token compression, a visual prefix bridge, and a Qwen2-LoRA code decoder. The model is trained on WebSight-derived data and evaluated on both the WebSight-derived test set and the Design2Code benchmark. General reconstruction metrics, including Block-Match, Text, Position, Color, and CLIP, are used together with honeypot-specific metrics, namely Honeypot-Critical Component Recall and Honeypot Interaction Readiness, to assess visual fidelity, reconstruction quality, component preservation, and interaction readiness. Experimental results on Design2Code show that HFG-7B achieves competitive reconstruction performance, with 51.2 Block-Match, 77.8 Text, and 85.2 CLIP, among other metrics. Under the same 1.5B decoder scale, HFG-1.5B significantly outperforms the PVT-Qwen-1.5B baseline in honeypot-specific fidelity, improving HCCR from 0.46 to 0.68 and HIR from 0.39 to 0.57. With a larger 7B decoder, HFG-7B further improves HCCR and HIR to 0.76 and 0.64, respectively. These results demonstrate that HFG renders visually plausible interfaces while better preserving honeypot-critical components, providing a locally trainable, resource-conscious, and practical solution for generating inspectable Web honeypot frontends.

KEYWORDS: Active defense; Web honeypot; large language models; multimodal content generation; vision transformer

1 Introduction

Web applications have become core components of modern information systems and now support a wide range of services, including e-commerce platforms, enterprise portals, and online administrative systems. As these services become more widely deployed, the attack surface exposed by Web applications has also become increasingly complex and diverse. Common attack paradigms, such as SQL injection, cross-site scripting, and malicious form exploitation, threaten application security and increase the burden on conventional defense mechanisms. Web honeypots provide an active defense mechanism by emulating realistic Web environments to attract adversaries and collect behavioral evidence for subsequent analysis [1–3].

Copyright © 2026 The Authors. Published by Tech Science Press. This work is licensed under a Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Despite their utility, traditional Web honeypots remain difficult to construct and deploy at scale. For a single target, crawler-based cloning and manual sanitization can often produce usable frontend decoys. However, large-scale deployment of heterogeneous Web honeypot nodes usually requires different crawling scripts, resource localization rules, JavaScript handling strategies, backend-interface removal, and manual inspection for different Web systems. These target-specific operations increase deployment cost and make the construction process difficult to standardize [4]. Recent advances in deep visual modeling have made screenshot-conditioned webpage reconstruction increasingly feasible, enabling frontend code to be synthesized from webpage screenshots [5–8]. In this work, the considered scenario is not full website cloning, but frontend decoy construction for heterogeneous Web-facing services. The defender uses screenshots of target-like interfaces as visual specifications and generates inspectable HTML/CSS pages intended for isolated deployment, while backend behavior is separately implemented through controlled honeypot handlers.

To address these challenges, this paper presents a security-oriented framework for generating Web honeypot frontends that integrates a PVT-CoT visual encoder, multi-scale feature fusion, a visual prefix bridge, and an LLM-based code decoder [9–11]. Specifically, the PVT-CoT encoder extracts hierarchical visual representations from webpage screenshots and enhances local context modeling for small interface components. The multi-scale fusion module then combines shallow component details with deeper layout semantics, and the visual prefix bridge maps the compressed visual tokens into a decoder-compatible embedding space. Conditioned on these visual prefixes, the LLM-based decoder generates HTML/CSS frontend code. The generated output is an inspectable HTML/CSS page that preserves the visible deception surface and key interaction components while avoiding direct reuse of production backend logic. During deployment, the generated frontend elements can be bound to controlled honeypot handlers, whereas LLM-based backend service emulation and multi-turn adversarial interaction are left as future work.

2 Related Work

This section reviews related work in three areas, covering honeypot technology, image-to-code generation, and LLM-assisted code generation and cybersecurity.

Honeypot technology is a fundamental component of deception-based defense, which aims to attract adversaries by emulating realistic system environments and collecting behavioral evidence for analysis [1–3]. Early research explored tarpitting, service emulation, and virtualized deception mechanisms to mislead attackers [2,12]. Subsequent developments introduced virtual honeypots capable of emulating configurable hosts and network services, thereby improving deployment flexibility and resistance to simple fingerprinting. Malware-collection platforms and SSH honeypot studies further show that honeypot systems can be specialized for different attack-observation scenarios [13,14]. In industrial control scenarios, specialized honeypots have been developed to emulate programmable logic controllers, thereby extending deception-based defense to critical infrastructure environments [15]. Despite these advancements, Web-facing honeypot construction still often relies on manual development, template-based generation, or crawler-based replication to obtain frontend pages. These approaches are often labor-intensive and difficult to standardize across heterogeneous Web services. Although crawler-based cloning can handle some dependency issues through sanitization, it often requires site-specific resource localization, script filtering, link rewriting, backend-interface removal, and manual inspection, which increases the engineering and auditing cost of isolated honeypot deployment. As a result, efficiently obtaining realistic, controllable, and deployment-ready frontend decoys remains a key challenge for Web honeypot construction. Recent studies have further extended honeypot research from static decoy construction to adaptive deployment, dynamic interaction, and deception-oriented cyber defense. For example, adaptive honeypot mechanisms and multiphase deployment strategies have been proposed to respond to changing attack paths and dynamic cyber threats [16–18]. Other studies have

investigated interactive Web honeypots and the broader role of honeypots in modern deception-based cybersecurity strategies [19–21]. More recently, generative AI has also been explored for honeypot construction and dynamic deception content generation, including generative-AI-assisted honeypots and dynamic file honeypots [22,23]. These studies demonstrate the increasing importance of adaptability and realism in honeypot systems.

Image-to-code generation aims to automatically synthesize interface or frontend code from UI screenshots. Early approaches demonstrated the feasibility of this task using neural image-to-sequence models [5,24], but were largely limited to synthetic GUI datasets and domain-specific intermediate representations. More recent studies have explored pretraining datasets, structured parsing objectives, and benchmark protocols to improve code consistency and evaluation reliability. For example, Pix2Struct uses screenshot parsing as a pretraining task for visual language understanding [6], while WebSight provides large-scale paired webpage screenshots and HTML code for screenshot-to-HTML generation [7]. Design2Code further evaluates multimodal code generation on real-world webpages and provides automatic metrics for frontend reconstruction quality [8]. However, these methods are mainly designed for general UI prototyping, webpage reconstruction, or automated front-end engineering, and they do not explicitly consider the security-oriented requirements of honeypot frontend deployment. For honeypot deployment, the generated interface should preserve attack-facing interaction cues, remain separated from real backend behavior, and support deployment auditability. From a modeling perspective, Transformer architectures provide a self-attention mechanism for modeling long-range dependencies [25,26], while PVT-based backbones use pyramid representations that are suitable for dense visual prediction [9]. Building upon this line of research, this work adopts a PVT-based visual encoder to extract multi-scale webpage features, which are subsequently fused and passed to an LLM-based decoder for frontend code generation.

Large Language Models have demonstrated strong capabilities in language generation and few-shot task adaptation [27–29], leading to their increasing adoption in cybersecurity tasks [30,31]. Existing studies have explored their applications in honeypot analysis, terminal honeypot dialogue, Web API deception, and multi-protocol cyber deception [32–36], where they enable more flexible and context-aware responses than traditional rule-based mechanisms. In this work, the LLM is mainly used as a code decoder for frontend generation rather than as a complete backend interaction engine. The generated frontend pages can be bound to controlled honeypot handlers after deployment, while LLM-based backend service emulation and multi-turn adversarial interaction are considered future research directions. Therefore, this paper focuses on security-oriented frontend decoy generation, aiming to improve visual fidelity, honeypot-critical component preservation, and deployment auditability of Web honeypot interfaces.

3 Research Design

3.1 Overview of the Proposed Framework

This work focuses on generating security-oriented frontend decoy interfaces for Web honeypot nodes. The operational scenario considered in this paper is the large-scale construction of heterogeneous Web-facing decoys, including login portals, management panels, device configuration pages, and service dashboards. In this setting, the defender may use screenshots of target-like interfaces as visual specifications rather than directly reusing production frontend code. For a single target, crawler-based cloning and manual sanitization can often produce usable frontend pages. However, across heterogeneous Web systems, crawler-based construction usually requires site-specific crawling scripts, resource localization rules, JavaScript handling strategies, link rewriting, backend-interface removal, and manual inspection. These target-specific operations increase engineering and auditing costs and make standardized frontend decoy construction difficult. Therefore, the objective of this work is not to reproduce a complete production website, but to

reconstruct the visible attack-facing surface from a webpage screenshot and convert it into a frontend decoy interface that can be inspected and bound to controlled honeypot handlers.

Fig. 1 presents the operational overview and deployment boundary of the proposed honeypot frontend generation (HFG) framework. The framework takes a webpage screenshot as input and outputs browser-renderable HTML/CSS code. The generated frontend is expected to preserve the visible layout, component hierarchy, visual style, interaction cues, and visible service-context text of the target interface. Unlike general screenshot-to-code generation, the proposed task emphasizes honeypot-specific requirements, including the preservation of attack-facing components, separation from production backend behavior, and suitability for isolated deployment.

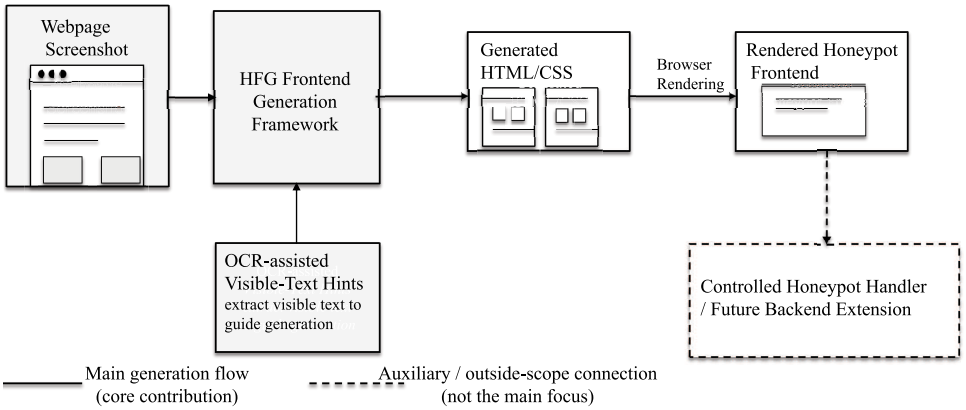

Figure 1: Operational overview and deployment boundary of the proposed HFG framework.

In the main generation flow, the webpage screenshot is processed by the HFG framework, which outputs HTML/CSS code corresponding to the visible page structure. The generated code is then rendered in a browser to obtain a honeypot frontend decoy. The rendered frontend serves as the attack-facing interface that adversaries interact with during deployment. Since the framework generates a frontend decoy rather than a complete production service, the output is intended to provide a visually plausible and controllable interface instead of replicating backend behavior or business logic.

Optional OCR-assisted visible-text hints are used as an auxiliary input to improve the preservation of text-sensitive frontend cues. These cues include button labels, form labels, navigation items, warning messages, version strings, copyright information, and other service-context text visible in the screenshot. The OCR module extracts text only from the input screenshot and does not access reference HTML or ground-truth code. Therefore, OCR-assisted hints provide additional visual-text grounding without introducing access to reference code during inference.

The rendered frontend defines the interaction surface exposed to adversaries in deployment. Visible interactive elements, such as login forms, submission controls, upload entries, and administrative links, are retained to define the interaction points of the decoy interface rather than to reproduce the original backend behavior. These interaction points can later be handled by defender-controlled honeypot logic. As indicated by the dashed path in Fig. 1, this runtime handling is outside the core generation module. This separation clarifies the scope of this work by limiting the proposed framework to auditable frontend decoy generation, while backend emulation and adversarial interaction management remain deployment-side extensions.

3.2 Vision-Guided Frontend Generation Model

The frontend generation model takes a webpage screenshot as visual input and produces an HTML/CSS decoy interface intended for isolated Web honeypot deployment. As shown in Fig. 2, the model combines a PVT-CoT visual encoder, multi-scale adaptive fusion, visual token compression, a visual prefix bridge, and a Qwen2-LoRA code decoder [9–11]. The visual encoder extracts hierarchical webpage representations from the screenshot, and the decoder generates structured frontend code conditioned on visual tokens and decoder-side textual inputs.

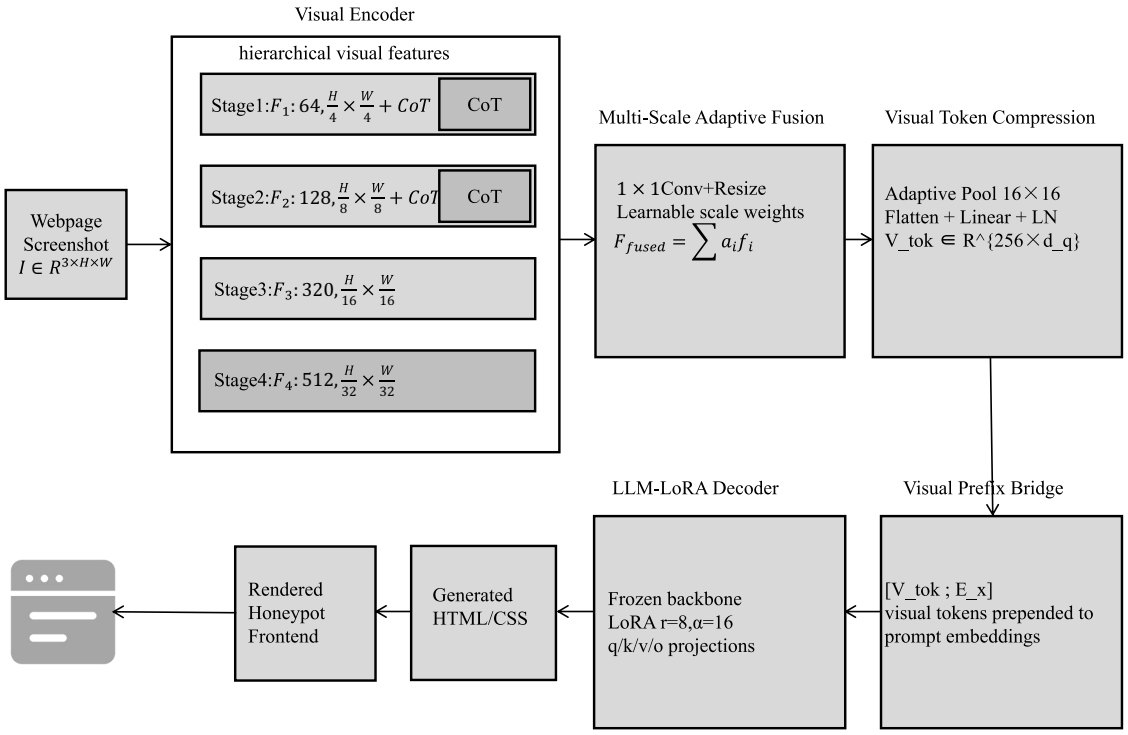

Figure 2: Architecture of the proposed vision-guided frontend generation model.

Given an input screenshot $I \in \mathbb{R}^{3\times H\times W}$, the PVT-based visual encoder produces four stages of hierarchical features, which are denoted as follows.

$$\{F_1, F_2, F_3, F_4\} = \mathcal{E}_{\mathrm{PVT}}(I), \tag{1}$$

where $F_1 \in \mathbb{R}^{64\times H/4\times W/4}$, $F_2 \in \mathbb{R}^{128\times H/8\times W/8}$, $F_3 \in \mathbb{R}^{320\times H/16\times W/16}$, and $F_4 \in \mathbb{R}^{512\times H/32\times W/32}$. These hierarchical features match the visual structure of webpages, where global layouts coexist with small elements such as input fields, buttons, icons, warning banners, version labels, and copyright information.

PVT is used because its pyramid structure and spatial reduction attention reduce visual token redundancy when processing high-resolution webpage screenshots [9]. For a feature sequence $X \in \mathbb{R}^{N\times C}$ with $N = H_i W_i$, queries are computed from the original sequence, whereas keys and values are computed from the spatially reduced sequence.

$$Q = XW_Q, \quad X_{\mathrm{sr}} = \mathrm{SR}_r(X), \quad K = X_{\mathrm{sr}}W_K, \quad V = X_{\mathrm{sr}}W_V, \tag{2}$$

where $\mathrm{SR}_r(\cdot)$ denotes spatial reduction with ratio r. This reduces the number of key and value tokens from N to N/r^2. The attention operation is written as follows.

$$\mathrm{SRA}(X) = \mathrm{Softmax}\left(\frac{QK^{\top}}{\sqrt{d_k}}\right)V, \tag{3}$$

where d_k denotes the dimension of the key vectors. The resulting attention computation reduces cost while retaining global dependency modeling, which helps process webpage screenshots with high spatial resolution and complex layouts.

Although PVT provides hierarchical visual features, small components relevant to honeypot deception may be weakened during progressive downsampling. To improve local component perception, Contextual Transformer refinement is applied to the high-resolution visual stages [10]. CoT layers are applied to the shallow feature maps F_1 and F_2, which retain more spatial detail for small interface elements.

$$\widetilde{F}_i = \begin{cases} \mathrm{CoT}(F_i), & i \in \{1, 2\}, \\ F_i, & i \in \{3, 4\}. \end{cases} \tag{4}$$

This refinement strengthens local context modeling for security-relevant frontend cues, while the deeper PVT stages retain global layout semantics.

The four visual stages differ in channel dimension and spatial resolution. A multi-scale adaptive fusion module forms a unified representation for the decoder. Each feature map is first projected to a common channel dimension by a 1×1 convolution and resized to a shared spatial resolution.

$$\bar{F}_i = \phi_i(\widetilde{F}_i), \tag{5}$$

where $\phi_i(\cdot)$ denotes the stage-specific projection and resizing operation. The projected features are then fused using learnable scale weights.

$$F_{\mathrm{fused}} = \sum_{i=1}^{4} \alpha_i \bar{F}_i, \quad \alpha_i = \frac{\exp(w_i)}{\sum_{j=1}^{4} \exp(w_j)}. \tag{6}$$

Adaptive fusion combines shallow component details with deeper structural semantics and supports the reconstruction of page layout and interaction-related elements.

Feeding dense visual feature maps directly into the language decoder would introduce unnecessary computational cost. The fused feature map is therefore compressed into a fixed-length visual token sequence. It is first pooled to a 16×16 grid and then flattened into 256 visual tokens.

$$Z = \mathrm{Flatten}\left(\mathrm{Pool}_{16\times16}(F_{\mathrm{fused}})\right), \tag{7}$$

where $Z \in \mathbb{R}^{256\times d_f}$, and d_f denotes the channel dimension after multi-scale fusion. A linear projection followed by layer normalization aligns the token dimension with the hidden dimension of the decoder.

$$V_{\mathrm{tok}} = \mathrm{LN}(ZW_v + b_v), \tag{8}$$

where $V_{\mathrm{tok}} \in \mathbb{R}^{256\times d_q}$, $W_v \in \mathbb{R}^{d_f\times d_q}$, and d_q denotes the hidden dimension of the Qwen2 decoder. In the HFG-1.5B setting, $d_q = 1536$.

The visual prefix bridge maps the compressed visual tokens to embeddings that can be consumed by the decoder.

$$E_v = \mathcal{B}(V_{\text{tok}}). \tag{9}$$

The decoder input is formed by placing the visual prefix before the textual and code-side embeddings.

$$E = [E_v; E_x]. \tag{10}$$

Here, E_x denotes the decoder-side input embeddings. During training, E_x consists of the generation instruction, OCR-assisted visible-text hints, and shifted ground-truth HTML/CSS tokens used for teacher forcing. During inference, E_x consists of the generation instruction, OCR-assisted visible-text hints, and previously generated HTML/CSS tokens.

$$E_x = \begin{cases} [E_{\text{inst}}; E_{\text{ocr}}; E_{y_{<t}}], & \text{training}, \\ [E_{\text{inst}}; E_{\text{ocr}}; E_{\hat{y}_{<t}}], & \text{inference}. \end{cases} \tag{11}$$

This distinction keeps reference HTML/CSS as a supervised signal during training and prevents it from being used during inference.

The Qwen2-based code decoder generates HTML/CSS tokens autoregressively.

$$p(y_t \mid I, x, y_{<t}) = p_\theta\left(y_t \mid E_v, E_x\right), \tag{12}$$

where x denotes the decoder-side textual input, including the generation instruction and optional OCR-assisted visible-text hints. The model is trained with the standard next-token prediction loss.

$$\mathcal{L}_{\text{gen}} = -\sum_{t=1}^{T} \log p_\theta(y_t \mid E_v, E_{\text{inst}}, E_{\text{ocr}}, y_{<t}). \tag{13}$$

The loss is computed only on the target HTML/CSS tokens, while the visual prefix, instruction tokens, and OCR hint tokens are masked out from loss computation. To reduce training cost, the Qwen2 backbone is kept frozen and adapted using LoRA. LoRA parameters are inserted into the attention projection layers, including q, k, v, and o projections. This parameter-efficient adaptation keeps most decoder parameters fixed and adapts the model to screenshot-conditioned frontend generation.

At inference time, the model receives only the webpage screenshot, the generation instruction, and optional OCR-assisted visible-text hints. The output is an HTML/CSS page intended to reconstruct the visible attack-facing interface. The generated page can then be rendered in a browser as a honeypot frontend decoy.

3.3 Parameter-Efficient Four-Stage Training Strategy

Training the visual encoder and the language decoder with full-parameter end-to-end optimization is computationally expensive and may degrade the pretrained code-generation capability of the decoder. To address this issue, a parameter-efficient four-stage training strategy is adopted, as illustrated in Fig. 3. The strategy first aligns screenshot-derived visual representations with the decoder, then adapts the decoder through LoRA, and subsequently adjusts selected layers of the pretrained PVT backbone for webpage-oriented visual adaptation. The final stage further exposes the model to more diverse webpage patterns.

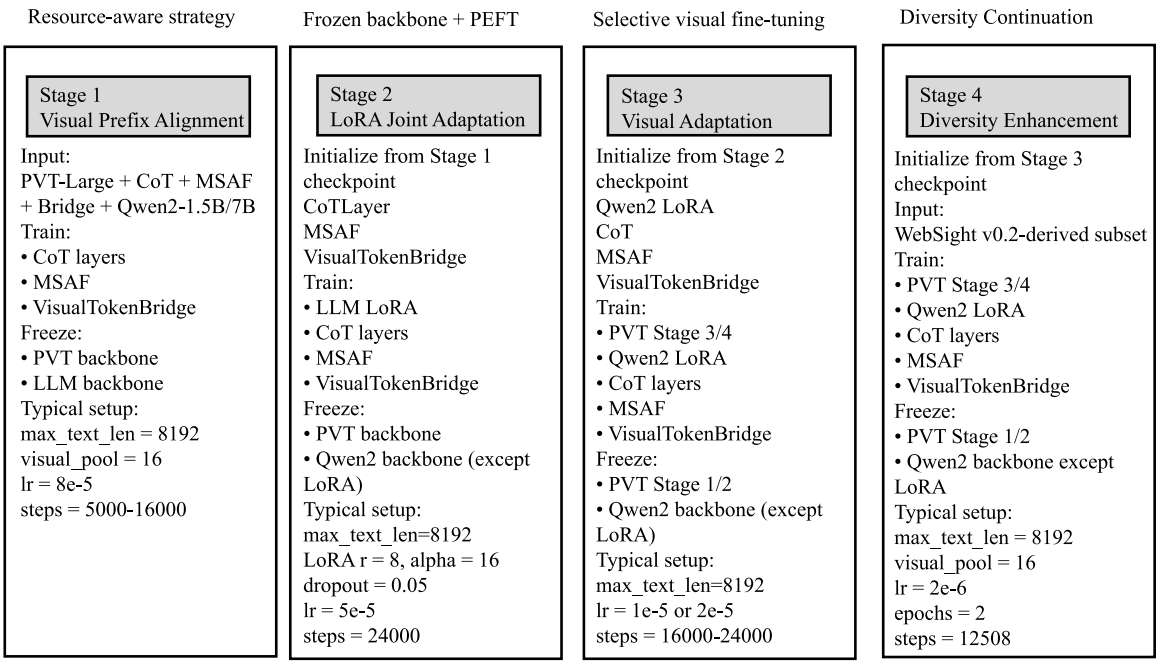

Figure 3: Four-stage parameter-efficient training strategy of the proposed framework.

In Stage 1, visual prefix alignment is performed with both the pretrained PVT backbone and the Qwen2 decoder backbone kept frozen. The trainable parts are limited to the CoT refinement layers, the multi-scale adaptive fusion module, and the Visual Token Bridge. This stage aligns the visual representations extracted from screenshots with the hidden space of the decoder. Keeping the language decoder frozen allows the visual prefix to be learned without updating the pretrained code decoder.

In Stage 2, LoRA-based adaptation is applied to the Qwen2 decoder. LoRA parameters are inserted into the q, k, v, and o attention projection layers, while the original Qwen2 backbone remains frozen. The CoT layers, MSAF, and Visual Token Bridge remain trainable during this stage. Compared with full-parameter fine-tuning, this stage adapts the decoder to screenshot-conditioned HTML/CSS generation while keeping the number of updated parameters limited.

In Stage 3, selected layers of the pretrained PVT backbone are unfrozen for visual adaptation. The purpose is not to train the visual encoder from scratch, but to adjust part of the pretrained visual representation to the webpage frontend generation task. In our implementation, later PVT layers are updated because they encode layout and structural semantics, while earlier layers remain frozen to preserve basic visual patterns such as edges, colors, and local textures. These selected PVT layers are updated together with the CoT layers, MSAF, Visual Token Bridge, and LoRA parameters. This selective update allows the visual encoder to better capture webpage layout patterns and component structures without changing the entire PVT backbone.

In Stage 4, continuation training is performed on the WebSight v0.2-derived subset to expose the model to more diverse webpage patterns. This stage uses the same trainable modules as Stage 3 and further introduces diverse webpage layouts, visual styles, and frontend structures. This continuation stage is expected to improve robustness when the model is evaluated on webpages with denser text, longer code structures, and more heterogeneous component layouts.

Across all stages, input screenshots are resized to 512×512, the visual pooling size is 16×16, and the maximum training text length is $T_{\max} = 8192$. The training objective follows the autoregressive next-token

prediction loss defined in Section 3.2. OCR-assisted visible-text hints are included in the decoder-side input when enabled. They are extracted only from the input screenshot and do not access reference HTML/CSS during inference.

This progressive training strategy reduces the optimization burden and improves training stability. It keeps most pretrained parameters frozen and updates only task-related modules, LoRA parameters, and selected PVT layers at the appropriate stage. It also separates visual-prefix alignment, decoder adaptation, and selective visual adaptation into different phases. As a result, the model learns screenshot-conditioned frontend generation while limiting unnecessary changes to the pretrained visual and language backbones.

3.4 Deployment Interface and Backend Extension

The proposed framework generates frontend decoy interfaces for Web honeypot nodes, rather than complete Web services. The generated output is an inspectable HTML/CSS page intended to preserve the visible attack-facing interface in the target screenshot. During deployment, visible elements such as login forms, submit buttons, upload entries, and administrative links are retained as frontend interaction points that can be handled by defender-controlled honeypot logic.

This separation clarifies the safety boundary of deployment. The generated frontend exposes only the visible interaction surface, while runtime behavior remains separated from production logic. The generated page can therefore be inspected and modified before deployment, and its interaction points can be handled without exposing production services.

LLM-based backend emulation remains outside the scope of the present evaluation. Such a module may maintain interaction states, interpret request paths and parameters, and generate context-aware responses in multi-turn adversarial interactions. However, this paper does not claim or evaluate a complete LLM-driven backend honeypot. The quantitative evaluation focuses on frontend renderability, visual fidelity, honeypot-critical component preservation, frontend interaction readiness, code inspectability, and deployment efficiency.

4 Experiments

4.1 Experimental Setup

The experiments evaluate whether the proposed framework can generate frontend decoy interfaces suitable for Web honeypot deployment. Unlike general screenshot-to-code evaluation, this work focuses on renderability, visual fidelity, and honeypot-critical component preservation. The evaluation therefore considers general reconstruction metrics together with frontend interaction readiness and deployment-oriented code inspectability.

WebSight-derived data [7] are used for model training, and Design2Code [8] is used for external evaluation. The main training set is constructed from WebSight v0.1 and contains 30,000 screenshot–HTML pairs. To improve the diversity of honeypot-like frontend appearances, the subset is sampled to cover different page categories, layout structures, text densities, component distributions, and interaction patterns, including login-like pages, dashboards, form-based pages, navigation-heavy pages, content pages, and multi-section webpages. The dataset is split at the sample level with an 8:1:1 ratio, resulting in 24,000 training samples, 3000 validation samples, and 3000 internal test samples. A smaller 3000-sample subset is used for pipeline verification rather than final model comparison. In addition, a WebSight v0.2-derived subset with 6254 training samples is used for continuation training to expose the model to more diverse webpage styles. Design2Code contains 484 real-world webpage screenshots and is used only for external evaluation.

No Design2Code sample is used for training, validation, prompt tuning, or checkpoint selection. Table 1 summarizes the datasets used in the experiments.

Table 1: Datasets used in the experiments.

Dataset	Usage	Samples	Split	Purpose
WebSight v0.1-derived	Train/Val/Test	30,000	8:1:1	Main training and in-domain test
WebSight v0.1 smoke subset	Verification	3000	8:1:1	Pipeline verification
WebSight v0.2-derived	Continuation	6254 train	Train	Diversity enhancement
Design2Code	External test	484	Test only	Out-of-domain evaluation

To reduce the risk of data leakage, duplicate checking is performed between the WebSight-derived training subsets and the Design2Code evaluation set. Exact image hashes and perceptual hashes are compared for webpage screenshots. When source text is available, normalized textual signatures are also compared after removing formatting and whitespace differences. Samples identified as exact duplicates or near-duplicates are excluded from the training subset. This procedure reduces the possibility that screenshot-to-code models benefit from memorized visual layouts.

For controlled comparison, three model variants are evaluated under the same data split, prompt protocol, OCR-assisted hinting setting, generation budget, rendering pipeline, and checkpoint selection rule. PVT-Qwen-1.5B is used as the visual fine-tuning baseline. It adopts the original PVT encoder and Qwen2-1.5B decoder, without CoT refinement or multi-scale adaptive fusion. HFG-1.5B is the proposed compact model variant, which integrates PVT-CoT visual encoding, MSAF, visual token compression, visual prefix bridging, and Qwen2-1.5BLoRA decoding. HFG-7B keeps the same visual architecture and replaces the decoder with Qwen2-7B to analyze the quality–cost trade-off introduced by decoder scaling. External multimodal systems and benchmark-reported results are treated only as reference comparisons, because their training data, model scale, context window, inference environment, and prompt design are not fully controllable.

The prompt is treated as part of the evaluation protocol. A reconstruction-oriented prompt asks the model to generate a complete HTML/CSS implementation that visually matches the input screenshot. The prompt emphasizes layout, color, spacing, component hierarchy, visible text, and interaction elements, and discourages generic template content that is not visible in the screenshot. For all reported models, the same OCR-assisted visible-text hinting protocol is used during generation. The OCR module extracts visible text from the input screenshot and appends the recognized strings to the generation instruction as auxiliary hints. It does not access reference HTML, ground-truth code, or external webpage sources. All models are therefore evaluated under the screenshot-conditioned generation setting.

Unless otherwise specified, the input screenshot is resized to 512×512, the visual pooling size is 16×16, and the number of visual tokens is $N_v = 256$. The Qwen2 decoder is adapted using LoRA with rank $r = 8$, scaling factor $\alpha = 16$, and dropout 0.05. LoRA is applied to the q, k, v, and o projection layers, while the Qwen2 backbone remains frozen. The models are trained using the four-stage parameter-efficient strategy described in Section 3.3. During inference, the maximum generation length is set to support long HTML/CSS outputs, and the generated content is truncated after the first complete HTML document when applicable. All generated pages are rendered under the same browser viewport before metric computation.

The generated frontend pages are evaluated from two complementary perspectives. First, general frontend reconstruction quality is measured using Design2Code-style metrics, including Block-Match, Text,

Position, Color, and CLIP [8]. Block-Match evaluates the recovery of webpage block organization, Text measures visible text preservation, Position assesses spatial alignment, Color reflects visual style consistency, and CLIP measures visual-semantic similarity between the rendered output and the reference screenshot. Second, honeypot-oriented frontend fidelity is evaluated using Honeypot-Critical Component Recall (HCCR) and Honeypot Interaction Readiness (HIR), which measure whether attack-facing components and safe frontend interaction points are preserved in the generated decoy interface.

Honeypot-Critical Component Recall (HCCR) is used to evaluate security-oriented component preservation. Let $\mathcal{C}_{\text{ref}}$ and $\mathcal{C}_{\text{gen}}$ denote the honeypot-critical component categories detected in the reference and generated pages, respectively. HCCR is defined as

$$\text{HCCR} = \frac{|\mathcal{C}_{\text{ref}} \cap \mathcal{C}_{\text{gen}}|}{|\mathcal{C}_{\text{ref}}|}.$$

The HCCR category list is defined according to Web honeypot deployment requirements and covers surface cues, interaction cues, and security or attribution cues. Surface cues include titles, navigation regions, headers, and footers. Interaction cues include username or email inputs, password inputs, submit buttons, upload entries, forms, and administrative links. Security and attribution cues include warning banners, error messages, version labels, copyright information, and service-identification text. If a reference sample contains no honeypot-critical component, its HCCR is marked as N/A and excluded from the HCCR average.

Since HCCR measures component presence rather than interaction readiness, Honeypot Interaction Readiness (HIR) is reported as a complementary metric. Let $\mathcal{I}_{\text{ref}}$ denote the interaction components in the reference page. For each $c \in \mathcal{I}_{\text{ref}}$, $\text{Ready}(c) = 1$ if the corresponding component in the generated page is visible, can be located by browser automation, accepts the expected user action, and does not trigger navigation to external or production endpoints. Otherwise, $\text{Ready}(c) = 0$. HIR is defined as

$$\text{HIR} = \frac{1}{|\mathcal{I}_{\text{ref}}|} \sum_{c \in \mathcal{I}_{\text{ref}}} \text{Ready}(c).$$

If a reference page contains no interaction component, its HIR is marked as N/A and excluded from the HIR average. HIR does not evaluate complete backend service behavior. It measures whether the generated frontend retains safe frontend interaction points that can be handled by defender-controlled honeypot logic during deployment.

4.2 Controlled Comparison on the WebSight-Derived Test Set

This subsection evaluates the effect of the proposed architectural design on the WebSight-derived internal test set. Since this test set is sampled from the same data source as the training set but contains disjoint samples, it provides an in-domain setting for controlled comparison. All model variants are trained and evaluated under the same data split, reconstruction prompt, OCR-assisted hinting protocol, LoRA configuration, rendering pipeline, and validation-based checkpoint selection rule.

The comparison uses three controlled variants. PVT-Qwen-1.5B serves as the visual fine-tuning baseline, using the original PVT encoder and Qwen2-1.5B decoder without CoT refinement or multi-scale adaptive fusion. HFG-1.5B uses the same decoder scale but adds the proposed PVT-CoT encoder and MSAF module, so the difference between PVT-Qwen-1.5B and HFG-1.5B reflects the combined effect of local visual refinement and multi-scale feature fusion. HFG-7B keeps the same visual architecture as HFG-1.5B and replaces the decoder with Qwen2-7B to analyze the effect of decoder scaling.

Frontend reconstruction quality is evaluated using Block-Match, Text, Position, Color, and CLIP. Block-Match measures webpage block organization, Text evaluates visible text preservation, Position measures spatial alignment, Color evaluates visual style consistency, and CLIP reflects visual-semantic similarity between the rendered output and the reference screenshot. These metrics are used because visually similar HTML/CSS implementations may differ substantially at the source-code level, making code-level n-gram similarity less suitable for this task.

Table 2 reports the in-domain comparison results. Under the same decoder scale, HFG-1.5B improves over PVT-Qwen-1.5B across most visual metrics, suggesting that the proposed visual refinement and fusion design benefits screenshot-conditioned frontend reconstruction. The improvements in Block-Match, Position, and Color indicate better recovery of webpage regions, spatial layout, and visual style. The Text metric changes only slightly from PVT-Qwen-1.5B to HFG-1.5B, whereas HFG-7B shows a much larger gain, which suggests that decoder capacity has a stronger influence on text-heavy generation. HFG-7B further improves Block-Match and Text, while the gains in Position, Color, and CLIP are more moderate. This pattern indicates that decoder scaling mainly benefits long-structure generation and text preservation in this setting. HFG-1.5B still offers a more resource-conscious balance between reconstruction quality and model size.

Table 2: Controlled comparison on the WebSight-derived internal test set.

Model	Block-Match	Text	Position	Color	CLIP
PVT-Qwen-1.5B	64.3	65.3	67.2	58.9	78.7
HFG-1.5B	72.4	66.2	82.9	78.5	88.2
HFG-7B	80.3	87.3	83.4	79.6	89.6

The WebSight-derived results are expected to be higher than the external Design2Code results because the former follows an in-domain setting, whereas Design2Code contains real-world webpages with different layout and content distributions. This experiment is therefore used to examine the proposed architecture under controlled in-domain conditions. The following subsection evaluates cross-domain generalization on Design2Code and compares the proposed models with representative multimodal systems.

4.3 External Comparison on Design2Code

This subsection evaluates the external performance of the proposed models on the Design2Code benchmark and compares them with representative multimodal systems. Compared with the WebSight-derived internal test set, Design2Code contains real-world webpages with denser visible text, more varied layouts, and longer frontend structures. It therefore provides a more challenging setting for evaluating screenshot-conditioned frontend generation outside the training data source.

Table 3 reports the Design2Code results. The results of PVT-Qwen-1.5B, HFG-1.5B, and HFG-7B are obtained under the same OCR-assisted generation protocol, rendering pipeline, and metric computation procedure. External multimodal systems and benchmark-reported results are included as reference comparisons, because their training data, model scale, inference environment, context-window configuration, and prompt design are not fully controllable. Therefore, these results are used to contextualize the proposed models, rather than to define a strictly controlled baseline.

Compared with PVT-Qwen-1.5B, HFG-1.5B improves most Design2Code metrics. The gains in Block-Match and Position indicate better recovery of webpage regions and spatial organization, while the gains in Color and CLIP suggest improved visual style preservation and visual-semantic consistency. These

results suggest that the proposed visual refinement and fusion design remains useful under the external Design2Code setting.

Table 3: External comparison on the Design2Code benchmark.

Model	Block-Match	Text	Position	Color	CLIP
Idefics2-8B	46.7	80.3	55.9	58.9	81.7
DeepSeek-VL-7B	39.7	77.0	64.6	63.8	84.5
LLaVA 1.6-7B	50.4	87.9	69.1	63.4	84.6
WebSight VLM-8B	55.9	86.6	77.3	79.4	87.6
Design2Code-18B	78.5	96.4	74.3	67.0	85.8
Qwen2-VL-7B	46.9	78.1	69.3	63.6	84.3
Gemini 1.0 Pro Vision	80.2	94.6	72.3	66.2	84.4
GPT-4V	85.8	97.4	80.5	73.3	86.9
Claude 3 Opus	90.2	97.5	77.9	71.4	87.0
GPT-4o	93.0	98.2	85.5	84.1	90.4
PVT-Qwen-1.5B	27.8	59.3	53.2	51.3	73.4
HFG-1.5B	37.6	60.2	61.3	61.8	80.4
HFG-7B	51.2	77.8	70.2	62.4	85.2

Among the proposed variants, HFG-7B obtains the strongest overall results, with the largest gain appearing in the Text metric. This pattern suggests that a larger decoder is helpful for text preservation and long-structure HTML/CSS generation. This improvement, however, comes with a larger decoder and higher computational cost. HFG-1.5B provides a more resource-conscious trade-off between reconstruction quality and model size, which is relevant to Web honeypot deployment scenarios that favor local training and private inference. For Web honeypot frontend generation, structural consistency, layout fidelity, and visual plausibility remain important because the generated page is used as an attacker-facing decoy interface.

Compared with large proprietary multimodal systems, the proposed models still show clear gaps in several reconstruction metrics, especially Text and Block-Match. This gap is reasonable, given the larger model scales and broader visual-language pretraining of those systems. Nevertheless, this work does not aim to outperform proprietary general-purpose multimodal models across all webpage reconstruction metrics. Instead, the proposed framework aims to provide a locally trainable and security-oriented frontend generation model for Web honeypot deployment under moderate model scale. The following subsection further examines this security-oriented objective through honeypot-critical component preservation.

4.4 Honeypot-Oriented Component Fidelity

General webpage reconstruction metrics mainly evaluate visual similarity between the rendered page and the reference screenshot. However, visual similarity alone is not sufficient for Web honeypot deployment. A generated frontend may obtain acceptable visual scores but still omit attack-facing components such as username or password fields, submit buttons, upload entries, warning messages, version labels, and administrative cues. These components often occupy small visual regions, yet they affect the credibility of the decoy and its ability to guide attacker interaction. This subsection therefore examines honeypot-critical component preservation.

Table 4 reports HCCR and HIR for the controlled model variants. HCCR measures whether honeypot-critical component categories are preserved, while HIR evaluates whether the recovered interaction components remain visible, operable, and suitable for defender-controlled handling. In implementation, HCCR and HIR are computed using a Python-based evaluation script that combines rendered-page DOM parsing, keyword matching, OCR-assisted visible-text matching, and browser automation. Component categories are detected from both HTML structure and rendered visual evidence. For HIR, browser automation is used to check whether the recovered interaction elements are visible, locatable, and able to accept the expected user action without navigating to external or production endpoints.

Table 4: Honeypot-oriented component fidelity of different model variants.

Model	WebSight HCCR	WebSight HIR	Design2Code HCCR	Design2Code HIR
PVT-Qwen-1.5B	0.58	0.51	0.46	0.39
HFG-1.5B	0.73	0.66	0.68	0.57
HFG-7B	0.81	0.72	0.76	0.64

Compared with PVT-Qwen-1.5B, HFG-1.5B improves both HCCR and HIR on the WebSight-derived test set and the Design2Code benchmark. Since these two models use the same decoder scale and evaluation protocol, the improvement suggests that the proposed visual refinement and multi-scale fusion design helps recover security-relevant frontend components. HFG-7B obtains the highest HCCR and HIR among the controlled variants, suggesting that a larger decoder helps preserve interaction components and longer frontend structures. Nevertheless, HFG-1.5B provides a more resource-conscious balance between component fidelity and model size.

The improvement is more evident on Design2Code, where webpages contain denser text, more complex layouts, and more varied interface structures. PVT-Qwen-1.5B can reconstruct the general page appearance, but it tends to omit small interaction elements and weak service-context cues. In contrast, HFG-1.5B better preserves credential-related regions, buttons, forms, navigation entries, and warning or attribution text. This result is consistent with the architectural motivation of CoT refinement and multi-scale adaptive fusion, where high-resolution local features support small-component preservation and deeper features provide layout context for structured frontend generation.

Table 5 further reports grouped HCCR results on Design2Code to show which component types benefit from the proposed design. Surface cues are generally easier to preserve because they are associated with larger visual regions and global layout structures. Interaction cues are more challenging because the model must reconstruct semantically meaningful frontend elements, including input boxes, password fields, submit buttons, forms, and upload entries. Security and attribution cues are the most difficult group because they often appear as small text regions, including version strings, warning messages, copyright lines, and service-identification labels. HFG-1.5B consistently improves all three groups compared with PVT-Qwen-1.5B, and the gain is especially relevant to interaction cues and security or attribution cues. Equivalently, the lower HCCR values for interaction cues and security or attribution cues indicate that these components account for most omissions in weaker baselines.

Fig. 4 shows a qualitative example of honeypot-critical component preservation. The figure compares the reference screenshot with browser-rendered outputs from different models. Compared with PVT-Qwen-1.5B, HFG-1.5B better preserves the login panel, input fields, navigation entries, and warning region. HFG-7B further improves visible labels and form contents in this example. These qualitative observations are consistent with the quantitative HCCR results.

Table 5: Grouped HCCR on the Design2Code benchmark.

Model	Surface Cues	Interaction Cues	Security/Attribution Cues	Overall
PVT-Qwen-1.5B	0.63	0.41	0.34	0.46
HFG-1.5B	0.78	0.63	0.55	0.68
HFG-7B	0.84	0.71	0.65	0.76

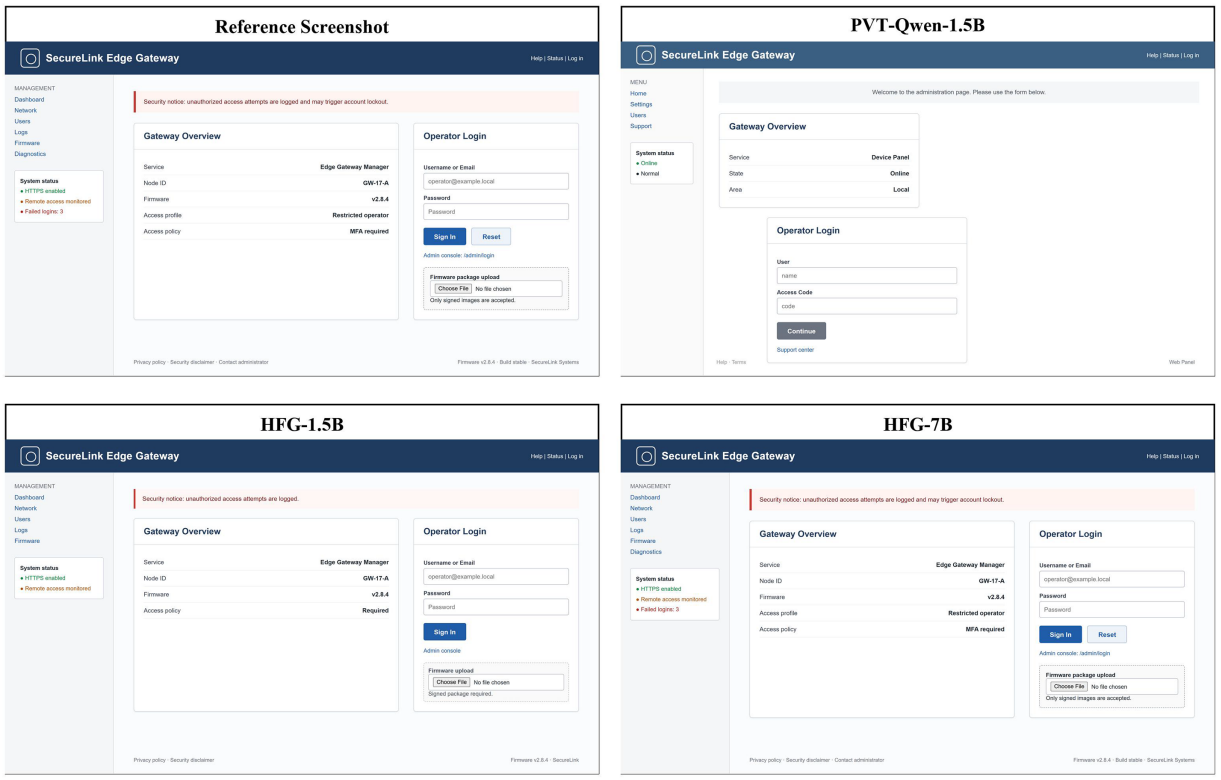

Figure 4: Qualitative example of honeypot-critical component preservation.

Overall, the component-level results show that Web honeypot frontend generation should be evaluated as both a visual reconstruction task and a security-oriented component preservation task. The proposed HFG architecture improves the preservation of attack-facing and service-context components by strengthening local visual representation and multi-scale feature fusion. This improvement supports the generation of frontend decoys that are visually plausible, interaction-ready, and meaningful for honeypot deployment.

4.5 Quality–Scale and Deployment-Oriented Analysis

In addition to generation quality and honeypot-critical component preservation, model scale affects the practicality of Web honeypot frontend generation. In large-scale Web honeypot deployment, defenders may need to generate multiple decoy interfaces for heterogeneous services, while training and inference resources are often limited. A practical model therefore needs to balance reconstruction quality, component fidelity, and model cost.

Table 6 summarizes the quality–scale trade-off among the controlled models. PVT-Qwen-1.5B and HFG-1.5B use the same Qwen2-1.5B decoder scale, while HFG-1.5B introduces PVT-CoT and MSAF to

improve visual grounding. The improvement from PVT-Qwen-1.5B to HFG-1.5B is therefore achieved without scaling the language decoder. This result suggests that the visual representation design contributes to honeypot-critical component recovery, rather than relying only on decoder scaling. HFG-7B further improves HCCR, suggesting that decoder scaling can help preserve components in longer HTML/CSS outputs. The larger model, however, also increases training and inference cost. Thus, HFG-1.5B provides a more resource-conscious balance, while HFG-7B serves as a quality-oriented variant when more computational resources are available.

Table 6: Quality–scale trade-off of controlled model variants.

Model	Decoder Scale	Visual Design	Approx. Total Scale	Design2Code HCCR
PVT-Qwen-1.5B	1.5B	PVT	~1.8B	0.46
HFG-1.5B	1.5B	PVT-CoT+MSAF	~1.8B	0.68
HFG-7B	7B	PVT-CoT+MSAF	~7.3B	0.76

The comparison shows that HFG-1.5B improves component fidelity over PVT-Qwen-1.5B while using the same decoder scale. This is relevant to deployment-oriented scenarios where local training, private inference, and moderate hardware requirements are preferred. Although HFG-7B achieves the highest HCCR, its additional gain comes from a much larger decoder. The 1.5B variant is therefore more suitable as the default model when local deployment cost is a concern, while the 7B variant can be used when higher generation quality is preferred and sufficient resources are available.

From a deployment perspective, generated frontend pages should be inspected before use in a honeypot environment. In particular, external URLs, remote scripts, non-local form actions, and unintended service paths should be checked to reduce the risk of unintended connections to production systems. This inspection requirement is consistent with the security-oriented goal of this work, which is to generate inspectable frontend decoys while keeping runtime behavior under defender control.

Overall, the analysis indicates that the proposed framework improves honeypot-oriented frontend generation through a more favorable quality–scale trade-off. HFG-1.5B offers a practical balance between model scale and component fidelity, whereas HFG-7B provides a higher-quality option at increased computational cost.

5 Security and Ethical Considerations

The proposed framework is intended for authorized Web honeypot deployment and security research. It generates frontend decoy interfaces for controlled defensive environments, rather than production services or phishing systems. Before deployment, generated HTML/CSS code should be inspected to remove unintended external URLs, remote scripts, non-local form actions, production-related paths, or sensitive identifiers. This inspection is necessary because a visually plausible decoy interface may still contain unsafe references if generated code is deployed without review. Therefore, the generated frontend should be treated as an auditable artifact that requires validation before being exposed to adversarial traffic.

This paper does not evaluate an LLM-driven backend honeypot, and the proposed model does not directly generate runtime responses to attacker-controlled inputs. This boundary reduces the immediate risks of prompt injection, cross-turn inconsistency, LLM fingerprinting, and unintended generation of harmful operational content. If LLM-based backend service emulation is introduced in future deployments, additional safeguards will be required, including input filtering, state constraints, response auditing, latency control, and policies that prevent the system from generating exploitable payloads, real credentials,

or misleading administrative claims. Honeypot deployment should also follow organizational authorization, logging policies, and applicable legal requirements to ensure that deception-based defense remains controlled and accountable.

6 Conclusion and Future Work

This paper investigates security-oriented frontend decoy generation for Web honeypot nodes. Different from general webpage cloning or complete backend honeypot emulation, the proposed task focuses on generating browser-renderable and visually plausible HTML/CSS decoy interfaces from webpage screenshots. This formulation addresses a practical requirement in large-scale Web honeypot deployment, where defenders need believable frontend decoy interfaces while reducing dependence on production source code, backend endpoints, third-party scripts, and uncontrolled external resources.

HFG is proposed as a vision-guided framework for honeypot frontend generation. HFG employs a PVT-CoT visual encoder to capture multi-scale webpage features and strengthen the representation of small security-relevant interface components. A multi-scale adaptive fusion module integrates hierarchical visual representations, and a visual prefix bridge maps compressed visual tokens to a Qwen2-based code decoder. The decoder is adapted through LoRA to limit the number of updated parameters while retaining the pretrained decoder as the main code-generation backbone. OCR-assisted visible-text hints are further used to support the preservation of visible text cues.

Experimental results show that HFG improves frontend reconstruction quality and honeypot-critical component preservation over the PVT-Qwen-1.5B baseline. The results suggest that CoT refinement and multi-scale visual fusion help recover local interaction elements and service-context cues. The HFG-7B variant further improves text preservation and long-structure generation, while HFG-1.5B provides a more resource-conscious balance between generation quality and model scale. These findings support the feasibility of vision-guided frontend generation for constructing practical honeypot decoy interfaces.

Several limitations remain. The generated pages focus on reconstructing the visible decoy interface rather than reproducing complete production source code or backend business logic. Dense text regions, small service-context cues, and highly complex layouts may still be reconstructed imperfectly. In addition, this paper evaluates frontend generation quality and component preservation, but does not include closed-loop evaluation with real attack traffic.

Future work will first improve fine-grained visual-text grounding and then extend the deployment-side interaction capability. Fine-grained visual-text grounding can be improved through stronger OCR supervision, component-aware data construction, and layout-aware decoding. Future work will also examine resource-conscious model composition, where compact visual generators, code decoders, OCR modules, and component detectors are combined under constrained computational resources. LLM-based backend service emulation, including state tracking, context-aware response generation, and multi-turn adversarial interaction, will be studied separately as part of dynamic Web honeypot systems.

Acknowledgement: The authors would like to thank all individuals who provided helpful support and constructive suggestions during the preparation of this manuscript.

Funding Statement: This work was supported by the Henan Province Science and Technology Tackling Key Problems Plan Project (Grant Nos. 252102210173 and 252103810209), the Key Research Projects of Higher Education Institutions in Henan Province (Grant Nos. 25A520051, 24A520011, and 24A520008), the Key Research and Development Program of Henan Province (Grant No. 261111211200), the Natural Science Foundation of Henan Province (Grant No. 252300421507), the Henan Provincial Higher Education Teaching Reform Research and Practice Project (Grant No. 2026SJGLX192), and the Research Project on Education and Teaching Reform of Henan University of Engineering

(Grant No. 2024JYYB020). National Natural Science Foundation of China (62573298). Guangdong Provincial Key Laboratory (Grant 2023B1212060076).

Author Contributions: The authors confirm contribution to the paper as follows: study conception and design: Guan Yang and Yu Wang; methodology: Guan Yang, Shiyan Kang and Bo Chen; data collection: Guan Yang and Shiyan Kang; model implementation and experiments: Guan Yang and Shiyan Kang; analysis and interpretation of results: Guan Yang, Bo Chen and Yu Wang; draft manuscript preparation: Yu Wang and Shiyan Kang; supervision: Yu Wang. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: The datasets used in this study are publicly available. The WebSight dataset is available at https://huggingface.co/datasets/HuggingFaceM4/WebSight. The Design2Code dataset is available at https://huggingface.co/datasets/SALT-NLP/Design2Code.

Ethics Approval: Not applicable. This study did not involve human participants or animals.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Cohen F. A note on the role of deception in information protection. Comput Secur. 1998;17(6):483–506. doi:10.1016/s0167-4048(98)80071-0.
2. Provos N. A virtual honeypot framework. In: Proceedings of the 13th USENIX Security Symposium; 2004 Aug 9–13; San Diego, CA, USA.
3. Kreibich C, Crowcroft J. Honeycomb: creating intrusion detection signatures using honeypots. SIGCOMM Comput Commun Rev. 2004;34(1):51–6.
4. Tsouvalas B, Nikiforakis N. Defense of the clones: securing web applications with automatic honeypot generation and deployment. In: Proceedings of the 2025 APWG Symposium on Electronic Crime Research (eCrime); 2025 Nov 4–7; San Diego, CA, USA. p. 1–17.
5. Beltramelli T. pix2code: generating code from a graphical user interface screenshot. In: Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems; 2018 Jun 19–22; Paris, France. p. 1–6.
6. Lee K, Josh M, Turc IR, Hu H, Liu F, Eisenschlos JM, et al. Pix2Struct: screenshot parsing as pretraining for visual language understanding. In: Proceedings of the 40th International Conference on Machine Learning; 2023 Jul 23–29; Honolulu, HI, USA. p. 18893–912.
7. Laurençon H, Tronchon L, Sanh V. Unlocking the conversion of web screenshots into HTML code with the WebSight dataset. arXiv:2403.09029. 2024.
8. Si C, Zhang Y, Li R, Yang Z, Liu R, Yang D. 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); 2025 Apr 29–May 4; Albuquerque, New Mexico. Stroudsburg, PA, USA: ACL; 2025. p. 3956–74.
9. Wang W, Xie E, Li X, Fan DP, Song K, Liang D, et al. PVT v2: improved baselines with pyramid vision transformer. Comp Visual Med. 2022;8(3):415–24. doi:10.1007/s41095-022-0274-8.
10. Li Y, Yao T, Pan Y, Mei T. Contextual transformer networks for visual recognition. IEEE Trans Pattern Anal Mach Intell. 2023;45(2):1489–500. doi:10.1109/tpami.2022.3164083.
11. Wang P, Bai S, Tan S, Wang S, Fan Z, Bai J, et al. Qwen2-VL: enhancing vision-language model's perception of the world at any resolution. arXiv:2409.12191. 2024.
12. Liston T. Welcome to my tarpit: the tactical and strategic use of LaBrea. Dshield.org White paper, 2001[EB/OL]. 2001 [cited 2026 Jan 1]. Available from: http://www.threenorth.com/LaBrea/.
13. Baecher P, Koetter M, Holz T, Dornseif M, Freiling F. The Nepenthes platform: an efficient approach to collect malware. In: Recent advances in intrusion detection. Berlin/Heidelberg, Germany: Springer; 2006. p. 165–84.
14. Valli I, Rabadi P, Woodward A. Patterns and pattern: an investigation into SSH activity using Kippo honeypots. In: Proceedings of the 11th Australian Digital Forensics Conference; 2013 Dec 2–4; Perth, Australia. p. 141–9.

15. López-Morales E, Rubio-Medrano C, Doupé A, Shoshitaishvili Y, Wang R, Bao T, et al. HoneyPLC: a next-generation honeypot for industrial control systems. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security; 2020 Nov 9–13; Virtual. p. 279–91. doi:10.1007/978-3-031-16613-6_8.
16. Abdul Kareem S, Sachan RC, Malviya RK. AI-driven adaptive honeypots for dynamic cyber threats. SSRN. 2024. doi:10.2139/ssrn.4966935.
17. Gao Y, Zhang G, Xing C. A multiphase dynamic deployment mechanism of virtualized honeypots based on intelligent attack path prediction. Secur Commun Netw. 2021;2021(7):6378218. doi:10.1155/2021/6378218.
18. Beltrán-López P, Gil Pérez M, Vasilomanolakis E, Nespoli P. Reactive cyber deception: stealth-based adaptive redirection to on-demand honeypots with AI-driven data generation. Comput Netw. 2026;281(2):112203. doi:10.1016/j.comnet.2026.112203.
19. Abewa YT. Dynamic interactive honeypot for web application security. Int J Wirel Microw Technol. 2024;14(6):1–14. doi:10.5815/ijwmt.2024.06.01.
20. Morić Z, Dakić V, Regvart D. Advancing cybersecurity with honeypots and deception strategies. Informatics. 2025;12(1):14. doi:10.3390/informatics12010014.
21. Aradi Z, Bánáti A. The role of honeypots in modern cybersecurity strategies. In: Proceedings of the 2025 IEEE 23rd World Symposium on Applied Machine Intelligence and Informatics (SAMI); 2025 Jan 23–25; Stará Lesná, Slovakia. p. 189–96.
22. Gizzarelli E. Honeypot and Generative AI [Ph.D. thesis]. Turin, Italy: Politecnico di Torino; 2024.
23. Strogov V, Ulasen S. Leveraging generative AI for dynamic file honeypots: insights and implementation. Cyber Secur A Peer Rev J. 2025;8(4):392. doi:10.69554/ztax4067.
24. Ramachandran P, Liu P, Le Q. Unsupervised pretraining for sequence-to-sequence learning. In: Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing; 2017 Sep 7–11; Copenhagen, Denmark. p. 383–91.
25. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, et al. Attention is all you need. In: Proceedings of the 31st International Conference on Neural Information Processing Systems; 2017 Dec 4–9; Long Beach, CA, USA. p. 6000–10.
26. Devlin J, Chang MW, Lee K, Toutanova K. BERT: pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers); 2019 Jun 2–7; Minneapolis, MN, USA. p. 4171–86.
27. Radford A, Narasimhan K, Salimans T, Sutskever I. Improving language understanding by generative pre-training. OpenAI. 2018 [cited 2026 Jan 1]. Available from: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.
28. Radford A, Wu J, Child R, Luan D, Amodei D, Sutskever I. Language models are unsupervised multitask learners. OpenAI. 2019 [cited 2026 Jan 1]. Available from: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.
29. Brown TB, Mann B, Ryder N, Subbiah M, Kaplan JD, Dhariwal P, et al. Language models are few-shot learners. In: Proceedings of the 34th International Conference on Neural Information Processing Systems; 2020 Dec 6–12; Vancouver, BC, Canada. p. 1877–901.
30. Alqahtani H, Kumar G. Large language models for cybersecurity intelligence: a systematic review of emerging threats, defensive capabilities, and security evaluation frameworks. Comput Mater Contin. 2026;87(3):1–10. doi:10.32604/cmc.2026.077367.
31. Yang A, Kang F, Bu W. TinySecGPT: small-parameter LLMS can outperform large-parameter LLMS in cybersecurity. Comput Mater Contin. 2026;87(2):1–10. doi:10.32604/cmc.2025.073979.
32. Ilg N, Germek D, Duplys P, Menth M. Beekeeper: accelerating honeypot analysis with LLM-driven feedback. IEEE Access. 2025;13:168034–54. doi:10.1109/access.2025.3613118.
33. Newsham L, Hyland R, Prince D. Inducing personality in LLM-based honeypot agents: measuring the effect on human-like agenda generation. arXiv:2503.19752. 2025.

34. Wang P, Wang L, Qian H, Gai W, Zheng Z, Zhang P, et al. LLM-THP: a large language model-powered terminal honeypot dialogue framework. In: Proceedings of the 2025 IEEE International Conference on High Performance Computing and Communications (HPCC); 2025 Aug 13–15; Exeter, UK. p. 816–23.
35. Sezgin A, Boyacı A. DecoyPot: a large language model-driven web API honeypot for realistic attacker engagement. Comput Secur. 2025;154(1):104458. doi:10.2139/ssrn.5009535.
36. Safargalieva A, Rüffer A, Vasilomanolakis E. OHRA: dynamic multi-protocol LLM-based cyber deception. In: Secure IT Systems. Cham, Switzerland: Springer Nature; 2026. p. 109–28.