Phase-wise MLLM Tuning for Multi-framework WebUI Code Generation

Haoran Ma[1*] **Linxiao Li**[2*] **Chenyue Wang**[2] **Haochen Sui**[3] **Jiechao Gao**[4†]

[1]Harvard University [2]The University of Sydney
[3]University of Michigan [4]Stanford University

Abstract

Multimodal large language models (MLLMs) achieve strong performance in translating WebUI screenshots into HTML/CSS. However, when they faced multiple frameworks (React/Vue/Angular), they often suffer from negative transfer that leads to compilation failures. These failures commonly stem from mixing framework specific syntax and violating framework constraints. This paper studies supervised fine-tuning for multi-framework WebUI code generation and proposes a phase-wise MLLM tuning method driven by compatibility and heterogeneity signals across frameworks. We first apply a normalization mapping that preserves each framework's native syntax while suppressing noise from variable naming and literal content. We then estimate cross framework heterogeneity and compatibility offline, and automatically derive training groups and their ordering by maximizing a phased objective. Finally, we perform phase-wise adapter tuning to reduce cross framework interference. Experiments on multiple benchmarks show that our proposed method improves both generation quality and compilation success rate compared with baselines.

1 Introduction

Generating executable front-end code from WebUI screenshots has become an important objective in automated software engineering [1]. Automating this process reduces the manual effort required to implement user interfaces and shortens the path from visual design to deployable code [2]. Early systems relied on computer vision pipelines that extracted structural elements from interface images and then used sequence models to emit HTML/CSS [3–6]. With the emergence of MLLMs, recent work has shifted toward end-to-end approaches that produce executable code directly from screenshots by jointly modeling visual inputs and code representations [7–9].

Despite this progress, most existing studies assume that the generated code targets HTML/CSS [8, 10]. In practical development, however, WebUI projects often need to support several widely used frameworks, including React, Vue, and Angular, which differ in syntax conventions, component organization, and engineering constraints [11]. When a model is trained jointly across such heterogeneous targets, or transferred from a model trained on a single framework, shared parameters receive supervision from distributions that favor different output patterns [12]. This can lead to cross-framework negative transfer [13], where generated code mixes syntax from different frameworks or violates framework-specific rules, ultimately causing compilation failures [14].

The root of this difficulty lies not only in the output distributions but also in the training dynamics induced by each framework. Heterogeneous target distributions produce gradient directions on the shared parameters that are not necessarily aligned across frameworks [15, 16]. When these signals are

*Equal contribution.
†Corresponding author: jiechao@stanford.edu.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

aggregated without regard to their relationships, shared updates may reflect inconsistent objectives, resulting in gradient interference and unstable optimization [17, 18]. This raises a natural question: In multi-framework WebUI code generation, can training strategies explicitly account for differences and compatibility among frameworks in order to reduce cross-framework interference?

To investigate this question, this paper studies supervised fine-tuning for multi-framework WebUI code generation and introduces a phase-based training strategy for MLLMs. The method first normalizes each framework's code to suppress superficial variation such as variable names and literal content. It then characterizes each pair of frameworks through two signals—a heterogeneity measure over normalized code distributions and a compatibility measure based on pre-gradients at the shared-adapter insertion point—and uses these signals to partition frameworks into training phases and order them. During training, the model combines a shared adapter with framework-specific adapters and updates them phase by phase, so that frameworks with more consistent optimization behavior are learned together. The resulting partition is derived offline and admits a generalization-bound interpretation.

Our contributions are summarized as follows:

- We propose a theoretical framework for phased-wise tuning of MLLMs in multi-framework WebUI code generation, where an offline method is used to obtain phase partition.
- We introduce a heterogeneity measure based on normalized codes and a compatibility measure based on offline pre-gradients, and use these two signals to automatically derive a phase partition and training schedule for multi-phase adapter finetuning.
- We build a multi-framework finetuning dataset based on DesignBench [11] and WebCode2M [10], and evaluate our method on DesignBench [11] and Design2Code [1]; the results consistently outperform baselines.

2 Related works

2.1 WebUI Code Generation

Early front-end code generation approaches relied on end-to-end CNN–RNN image-to-sequence models that translated screenshots into intermediate framework-specific languages [4, 6]. Subsequent work moved toward direct HTML generation [5] and improved structural fidelity through explicit layout recognition [19] or computer-vision pipelines such as OCR and object detection to recover hierarchical UI representations [3, 20]. With the emergence of multimodal large language models, recent progress has been driven by large-scale synthetic training data [7], layout-aware hierarchical planning [9, 21], and decomposition-based generation strategies [22, 23], together with post-training and efficiency improvements [24–26]. However, most existing studies focus on generation within a single frontend framework [11, 2, 27]. In practical deployment, applications often require supporting multiple frameworks that differ in syntax conventions, component abstractions, and engineering constraints. Directly transferring models trained for one framework to another—e.g., through prompt adaptation or lightweight fine-tuning—can lead to increased compilation errors and structural inconsistencies, highlighting the challenge of cross-framework robustness [11].

2.2 MLLM

Multimodal large language models (MLLMs) extend pretrained language models to visual inputs by integrating a vision encoder with a language backbone through projection or connector modules that map visual features into token space [28, 29]. Training typically proceeds via large-scale caption or interleaved image–text supervision, followed by visual instruction tuning to improve alignment and controllability [30, 31]. Recent advances further enhance perception and reasoning through higher-resolution visual tokenization, extended context windows, and refined post-training strategies [32–34].

2.3 LLM Fine-Tuning

Fine-tuning large language models has evolved from full-parameter updates to parameter-efficient strategies that adjust only a limited subset of parameters [35, 36]. Methods such as adapters [37],

prefix tuning [38], and low-rank adaptation [39] introduce lightweight modules that enable task adaptation with reduced memory and computational overhead. These approaches have demonstrated strong performance across language and multimodal tasks and are widely used in practice [40, 41]. Nevertheless, most fine-tuning methods are designed for single-task or single-framework scenarios [37, 38, 42]. When multiple frameworks or environments are optimized jointly, shared parameter updates may introduce gradient interference and negative transfer, particularly under heterogeneous output distributions or distinct engineering constraints [16]. Addressing such cross-framework interaction requires training strategies that explicitly account for framework compatibility in addition to parameter efficiency [17, 12, 43].

3 Method

3.1 Problem Setting

Given a webpage screenshot x and a target frontend framework identifier $f \in \{\texttt{React}, \texttt{Vue}, \texttt{Angular}, \texttt{Vanilla}\}$, our goal is to generate runnable frontend code y_f in the specified framework under its corresponding syntactic and compilation constraints, such that its rendered output matches the screenshot x as closely as possible in both structural layout and visual appearance. We denote the training set by $\mathcal{D} = \{(x_i, y_i, f_i)\}_{i=1}^{N}$, where N is the total number of training samples. For each framework f, we define the framework-specific subset as $\mathcal{D}_f = \{(x_i, y_i, f_i) \in \mathcal{D} \mid f_i = f\}$, which contains all pairs associated with framework f.

3.2 Model Parameters and Loss

We freeze the MLLM's parameters Θ and introduce adapters for k frameworks with parameters $W = \{w^s, w_1, \ldots, w_k\}$, where w^s denotes the shared parameters across all frameworks and w_j denotes the framework-specific parameters. The modified MLLM is defined as:

$$\mathcal{F}(x; \Theta, W) = \mathrm{MLLM}(x; \Theta, w^s, w_1, \ldots, w_k). \tag{1}$$

To explicitly control the effective capacity introduced by adapters, we assume a bounded-complexity condition on W, namely that there exists a constant $\beta > 0$ such that $||w||_2 \leq \beta$ for all $w \in W$. This constraint prevents any single framework-specific adapter from dominating the optimization dynamics, and it also enables a unified analysis in which the shared adapter w^s and the framework-specific adapters $\{w_j\}_{j=1}^{k}$ are treated under the same norm budget.

Given framework-specific datasets, we finetune the adapters by minimizing the average expected risk across frameworks:

$$\mathcal{L}_{train} = \frac{1}{k} \sum_{j=1}^{k} \mathbb{E}_{(x,y,f)\sim\mathcal{D}j} \big[\ell(\mathcal{F}(x; \Theta, W), y)\big]. \tag{2}$$

Here, $\ell(\hat{y}, y)$ denotes a sequence-level supervised loss under teacher forcing. Although our evaluation focuses on compilation success, optimizing $\mathcal{L}_{train}$ provides a differentiable surrogate that encourages the generated code to follow the target framework syntax and conventions encoded in $\mathcal{D}_j$. Notably, the explicit framework index j in the risk decomposition makes it possible to analyze how shared parameters w^s and framework-specific parameters w_j jointly affect the training dynamics across heterogeneous frameworks.

Assumption 3.1 (Lipschitz)**.** The loss $\ell(\hat{y}, y)$ is L-Lipschitz with respect to $\hat{y}$. Moreover, there exists a constant $\gamma > 0$ such that for any $W = (w^s, w_j)$ and $W^{'} = (w^{s^{'}}, w_j^{'})$ and any input x,

$$\|\mathcal{F}(x; \Theta, W) - \mathcal{F}(x; \Theta, W^{'})\| \leq \gamma * \|W - W^{'}\|_{2,1} = \gamma(\|w^s - w^{s^{'}}\|_2 + \sum_{j=1}^{k} \|w_j - w_j^{'}\|_2) \tag{3}$$

This assumption links perturbations of adapter parameters to changes in the model output, and it will be used to upper bound cross-framework deviations induced by different adapter updates.

3.3 Heterogeneity

For each framework dataset j, let Q_j denote the empirical distribution induced by its normalized code representations. For two framework datasets $\mathcal{D}_i$ and $\mathcal{D}_j$, we define their framework heterogeneity as

$$\Delta_{i,j} = h(\mathcal{D}_i, \mathcal{D}_j) \triangleq \mathcal{H}(Q_i, Q_j), \tag{4}$$

where $\mathcal{H}(\cdot,\cdot) \in [0,1]$ is a symmetric and bounded divergence between distributions. In this paper, we adopt the normalized Jensen-Shannon divergence. Here, Q_j is not constructed from raw code strings directly, but from normalized code distributions, in order to reduce the influence of superficial stylistic variations on the resulting statistics. The details are deferred to Appendix B. This definition measures the statistical shift between frameworks in the output space. If two frameworks admit more similar normalized code distributions, then their target code patterns, local structures, and compositional regularities are also more aligned, which makes stable transfer through shared parameters more likely.

Assumption 3.2 (Risk Bridge for Proxy heterogeneity)**.** There exist a constant $\kappa \geq 0$ and a residual $\epsilon_h \geq 0$ such that for any admissible parameters and any pair of frameworks $\mathcal{D}_i, \mathcal{D}_j$, we have

$$\left|\mathcal{L}(\mathcal{D}_i;\Theta,W) - \mathcal{L}(\mathcal{D}_j;\Theta,W)\right| \leq \kappa\Delta_{i,j} + \varepsilon_h. \tag{5}$$

This assumption states that the heterogeneity defined on $Q(y)$ is not merely an empirical statistic, but can upper bound the risk shift between frameworks. The residual term ε_h absorbs the portion of framework shift that cannot be fully explained by output distributions alone, such as input-side variation, latent semantic mismatch, or annotation noise.

3.4 Offline Compatibility Estimation

We choose a backbone layer index r and place the shared adapter parameter w^s at the output of layer $r-1$, namely the same insertion point used during training. For each framework $\mathcal{D}_j$, the offline shared gradient at initialization is defined as

$$\bar{g}_j = \nabla_{w^s}\mathcal{L}(\mathcal{D}_j;\Theta,W=w^s,\{w\})\Big|_{w^s=0,w=0}. \tag{6}$$

It represents the first-order update direction preferred by framework $\mathcal{D}_j$ on the shared adapter under MLLM and zero-initialized adapters. The compatibility between two frameworks is defined by

$$S(\mathcal{D}_i,\mathcal{D}_j) = \frac{1+\cos(\bar{g}_i,\bar{g}_j)}{2} \in [0,1]. \tag{7}$$

A larger value indicates better alignment of the induced shared update directions, and thus stronger suitability for being grouped into the same phase.

For a candidate phase P, with weights $\{\alpha_j\}_{j\in P}$ satisfying $\alpha_j \geq 0$ and $\sum_{j\in P}\alpha_j = 1$, we define the aggregate offline gradient and its compatibility margin as

$$\bar{g}_P = \sum_{j\in P}\alpha_j\bar{g}_j, \qquad \bar{m}(P) = \min_{j\in P}\frac{\langle \bar{g}_j,\bar{g}_P\rangle}{\|\bar{g}_P\|_2^2}. \tag{8}$$

When $\bar{m}(P) > 0$, the phase is first-order non-conflicting with respect to the shared adapter. We only consider candidate phases satisfying $\|\bar{g}_P\|_2 > 0$; phases with vanishing aggregate offline gradients are treated as infeasible.

For the considered phase P, let g_j denote the current shared gradient induced by framework $\mathcal{D}_j$, and let $g_P = \sum_{j\in P}\alpha_j g_j$ be its aggregate gradient.

Assumption 3.3 (Offline-to-online directional stability)**.** There exists $\varepsilon_g \geq 0$ such that $\|g_j - \bar{g}_j\|_2 \leq \varepsilon_g$ for all frameworks $\mathcal{D}_j$ in the current phase P.

Lemma 3.1 (Compatibility-preserving shared update)**.** *Assume that $\mathcal{L}(\mathcal{D}_j;\Theta,W)$ is L_s-smooth with respect to w^s for every framework $\mathcal{D}_j$. Define the online compatibility margin by*

$$m(P) = \min_{j\in P}\frac{\langle g_j,g_P\rangle}{\|g_P\|_2^2}. \tag{9}$$

If $m(P) > 0$ and the shared update $w^{s,+} = w^s - \eta g_P$ uses a step size $\eta \leq 2m(P)/L_s$, then

$$\mathcal{L}(\mathcal{D}_j;\Theta,W^+) \leq \mathcal{L}(\mathcal{D}_j;\Theta,W), \qquad \forall j \in P. \tag{10}$$

Lemma 3.1 shows that compatibility is not merely a heuristic similarity score. Once the online margin is positive, the aggregated shared update is descent-compatible for all frameworks in the phase. The justification for using the offline gradients $\{\bar{g}_j\}$ as surrogates of their online counterparts is deferred to Appendix C.

3.5 Phase-wise Generalization Bounds

Theorem 3.1 (Phase-wise concurrent generalization bound)**.** *Let $P \subseteq \{1, \dots, k\}$ be a candidate phase and let $n_P = \sum_{j \in P} |\mathcal{D}_j|$. Under the Lipschitz assumption, the bounded-complexity condition on W, and Assumption 3.2, for any $\delta > 0$, with probability at least $1 - \delta$, every admissible model satisfies*

$$\sum_{j \in P} \alpha_j \mathcal{L}(\mathcal{D}_j; \Theta, W) \leq \sum_{j \in P} \alpha_j \widehat{\mathcal{L}}(\mathcal{D}_j; \Theta, W) + \Gamma(P) + \kappa \sum_{\substack{i<j \\ i,j \in P}} h(\mathcal{D}_i, \mathcal{D}_j) + \epsilon_h + O\left(\sqrt{\frac{\ln(1/\delta)}{n_P}}\right). \tag{11}$$

where $\widehat{\mathcal{L}}$ is empirical risk and

$$\Gamma(P) \triangleq 2L\gamma \left(\beta + \sum_{j \in P} \alpha_j \beta\right) \leq 4L\gamma\beta. \tag{12}$$

Theorem 3.1 shows that the phase risk is controlled by the empirical loss, the adapter complexity, and the total internal heterogeneity inside the phase. Let $\Pi = (P_1, \dots, P_M)$ and $\bigcup_{t=1}^{M} P_t = \{1, \dots, k\}$. For each phase P_t, let $\{\alpha_j^t\}_{j \in P_t}$ be nonnegative weights such that $\sum_{j \in P_t} \alpha_j^t = 1$.

The heterogeneity term in Theorem 3.1 motivates penalizing internally divergent phases. In contrast, the compatibility term is introduced through Lemma 3.1, which shows that positively aligned shared gradients yield descent-compatible shared-adapter updates for all frameworks in a phase. Thus, our partition objective combines the heterogeneity penalty from the generalization bound with the compatibility reward from the shared-update analysis.

Motivated by this bound and the offline compatibility score in Eq. (7), we define the partition objective

$$G(P_1, \dots, P_M) = -\sum_{t=1}^{M} \left[\sum_{\substack{i,j \in P_t \\ i<j}} h(\mathcal{D}_i, \mathcal{D}_j) - \lambda \sum_{\substack{i,j \in P_t \\ i<j}} S(\mathcal{D}_i, \mathcal{D}_j) + \mu_s \|w^{s,t}\|_2^2 + \mu_f \sum_{j \in P_t} \|w_j^t\|_2^2 \right], \tag{13}$$

where $\lambda, \mu_s, \mu_f > 0$ and M is the number of phase. A larger value of G therefore corresponds to smaller internal heterogeneity, larger constructive compatibility, and lower per-phase adapter complexity. The norm-based capacity term in Eq.(13) serves as a computable surrogate for the complexity contribution $\Gamma(P_t)$ in Eq.(11), since the bounded adapter complexity condition controls both quantities through the shared and framework-specific adapter norm budgets.

Theorem 3.2 (Multi-Phase Partition with Compatibility-Capacity Maximization)**.** *Let*

$$(P_1^\star, \dots, P_M^\star) = \arg \max_{\Pi:\ \bar{m}(P_t) > C_{P_t}\epsilon_g,\ \forall t} G(P_1, \dots, P_M). \tag{14}$$

C_{P_t} is the stability constant from Proposition C.5 that converts offline margin into online margin. Suppose that every selected phase $P_t^\star$ satisfies the condition of Lemma 3.1. Then, under the assumptions of Theorem 3.1, the phase-wise training trajectory satisfies

$$R_{\max}(P_1^\star, \dots, P_M^\star) \leq \left[1 - G(P_1^\star, \dots, P_M^\star)\right]_+ + O\left(\sqrt{\frac{\ln(1/\delta)}{N}}\right), \tag{15}$$

where $R_{\max} \triangleq \max_{t \in \{1,\dots,M\}} \sum_{j \in P_t^\star} \alpha_j^t L(D_j; \Theta, W^t)$, $N = \sum_{j=1}^{k} |D_j|$, G denotes the monotone normalized objective, and $[u]_+ = \max\{0, u\}$.

Theorem 3.2 directly explains the training rule of our method. We first choose the partition that maximizes Eq. (13), and then perform phase-wise adapter optimization on the selected phases. The gating rule is defined in Appendix E. The process of algorithm is shown in Appendix A.

Table 1: Dataset statistics per framework.

Framework	Train	Val	Test	Avg. Tokens
Vanilla	2,814	312	300	1,023
React	2,773	308	295	1,187
Vue	2,681	298	284	1,142
Angular	2,594	288	277	1,309
Total	10,862	1,206	1,156	1,165

4 Experiments

4.1 Experimental Setup

Datasets and Metrics. We evaluate on two public benchmarks. **DesignBench** [11] supports four front-end frameworks (Vanilla, React, Vue, Angular) and covers Generation, Editing, and Repair tasks; we report Generation results in the main paper, evaluated on the held-out test split summarized in Table 1, and Editing/Repair in Appendix G. **Design2Code** [1] provides 484 real-world Vanilla webpage samples with five fine-grained automatic metrics. For training, we combine screenshot–code pairs from DesignBench [11] and WebCode2M [10] into a multi-framework fine-tuning set (Table 1). On DesignBench we report **CLIP** visual similarity and **CSR** (Compilation Success Rate); on Design2Code we report Block Match, Text, Position, Color, and CLIP. CLIP is computed between the rendered page of each generated code and the target screenshot; since uncompilable code admits no faithful rendering, CLIP is averaged over samples whose code compiles successfully, whereas CSR is computed over all test samples of the framework.

Model and Training. We use `Qwen2.5-VL-7B-Instruct` as the frozen backbone. One shared Adapter ($r = 16$, $\alpha = 32$) is inserted at layers 16–27; four framework-specific Adapters ($r = 8$, $\alpha = 16$) are inserted at layers 20–27. The offline partition algorithm yields **Phase 1**: {React, Vue} ($S = 0.79$, $\Delta = 0.18$); **Phase 2**: {Vanilla, Angular} ($S = 0.31$, $\Delta = 0.73$). We train 3 epochs per phase with batch size 32, learning rate 2×10^{-4}, and cosine warmup. Offline compatibility (synergy) gradients are computed at layer 15 over $K = 100$ batches per framework. All fine-tuning experiments use 3 random seeds; we report mean scores in the main comparison and ablation tables, and mean$\pm$std in the additional ablations (Appendix F) and the MLLM-as-judge evaluations (Appendix G).

Baselines. We compare against: (1) *frontier MLLMs (zero-shot)*: Claude-3.7 [44], GPT-4o [45], Gemini-2.0-Flash [46], Llama-3.2-11B [47], Qwen2.5-VL-7B [48]; (2) *structured generation*: Screen-Coder [23]; (3) *efficiency-oriented*: EfficientUICoder [24]; (4) *specialized baselines on Design2Code*: GPT-4V and the Design2Code-18B model [1]; and (5) *internal ablations* (Section 4.3).

4.2 Main Results

Multi-framework Generation on DesignBench. Table 2 shows that our method improves both visual fidelity and compilation reliability in the multi-framework generation setting. The gains are especially meaningful because they are obtained across frameworks with substantially different syntax systems, rather than being concentrated in a single easy framework. Compared with strong zero-shot MLLMs, our method closes the gap between general visual-code understanding and framework-aware executable generation. The improvement on Angular is particularly informative: this framework imposes stricter component and template constraints, and the result suggests that framework-specific Adapters together with gated shared updates help the model learn compilation-critical conventions. Compared with the ungated shared-specific baseline, the full method further demonstrates that offline relationship signals contribute beyond adding more trainable parameters.

Validation on Design2Code. Table 3 evaluates whether the proposed multi-framework tuning procedure remains effective on a Vanilla-only benchmark. The results show that the method does not sacrifice standard HTML/CSS generation ability while learning multiple frontend frameworks.

Table 2: Results on DesignBench Generation task. Best results are **bold**; second best are underlined.

Method	Vanilla		React		Vue		Angular		Avg.	
	CLIP	CSR	CLIP	CSR	CLIP	CSR	CLIP	CSR	CLIP	CSR
Claude-3.7	0.7832	**1.0000**	0.7891	0.9492	0.7934	0.6655	0.6223	0.6029	0.7470	0.8044
GPT-4o	0.7635	**1.0000**	0.6763	0.9119	0.7734	0.6092	0.5867	0.5379	0.7000	0.7648
Gemini-2.0	0.7412	**1.0000**	0.7611	0.8407	0.6997	0.1408	0.6006	0.4838	0.7007	0.6163
Llama-3.2-11B	0.6304	**1.0000**	0.6401	0.4915	0.3241	0.5810	0.5834	0.5415	0.5445	0.6535
Qwen2.5-VL-7B	0.7411	**1.0000**	0.6875	0.1085	0.6452	0.1197	0.5186	0.0866	0.6481	0.3287
ScreenCoder	0.7821	**1.0000**	0.7534	0.9232	0.7609	0.7887	0.6234	0.7822	0.7300	0.8735
EfficientUICoder	0.7623	**1.0000**	0.7312	0.8915	0.7423	0.7629	0.5934	0.7232	0.7073	0.8444
Specific Adapter only	0.7612	**1.0000**	0.7389	0.8621	0.7521	0.7617	0.6814	0.7196	0.7334	0.8359
Shared+Specific (no gate)	0.7701	**1.0000**	0.7523	0.8791	0.7634	0.7735	0.7012	0.7389	0.7468	0.8479
Ours	**0.7956**	**1.0000**	**0.8124**	**0.9582**	**0.8231**	**0.8310**	**0.7489**	**0.8135**	**0.7950**	**0.9007**

Table 3: Results on Design2Code benchmark. Best results are **bold**; second best are underlined.

Method	Block Match	Text	Position	Color	CLIP
GPT-4o	93.0	**98.2**	85.5	84.1	90.4
Claude-3.7	90.2	97.5	77.9	71.4	87.0
GPT-4V	85.8	97.4	80.5	73.3	86.9
Design2Code-18B	78.5	96.4	74.3	67.0	85.8
ScreenCoder	84.3	97.1	79.2	75.6	87.3
EfficientUICoder	81.7	96.8	77.4	72.3	86.1
Ours	**94.7**	97.9	**87.2**	**85.6**	**91.3**

Instead, the shared training phases appear to improve layout and visual reconstruction, suggesting that cross-framework data provide useful structural supervision even when the evaluation target is Vanilla code. The only metric where a frontier model remains slightly ahead is text matching, which is less directly tied to framework syntax and more sensitive to language-level copying. Overall, the Design2Code results indicate that our framework-level specialization does not come at the cost of general webpage generation quality.

4.3 Ablation Study

Architecture Ablation. Table 4 separates the effect of shared knowledge, framework-specific capacity, phase-wise scheduling, and gating. The specific-only variant lacks a mechanism for sharing reusable layout and component patterns, while the shared-only variant cannot sufficiently preserve framework-dependent syntax conventions. Combining shared and specific Adapters improves this balance, but the ungated version still treats all frameworks as equally suitable for shared updates. Adding phase-wise training already improves the optimization path by controlling which frameworks are learned together, and the full gated version further refines shared updates according to offline relationship scores.

Gating Component Ablation. Table 5 examines whether the gate should be driven by heterogeneity, compatibility, or both. The discrepancy-only and synergy-only variants each improve over unstructured gating, showing that both output-distribution distance and gradient-direction alignment contain useful training signals. However, neither signal alone is sufficient: discrepancy identifies when shared updates may be risky, while synergy identifies when shared updates are likely to be beneficial. The combined gate performs best because it jointly suppresses updates for highly heterogeneous frameworks and preserves shared learning for compatible ones. The weaker performance of uniform and random gates—both of which fall below the ungated phase-wise baseline (CLIP 0.7741 / CSR 0.8712 in Table 4)—further shows that the effect is not merely due to adding a scalar weight:

Table 4: Architecture ablation.

Variant	CLIP	CSR
Specific Adapter only	0.7334	0.8359
Shared Adapter only	0.7408	0.8412
Shared+Specific (no gate, no phase)	0.7468	0.8479
Shared+Specific (no gate, with phase)	0.7741	0.8712
Ours (gate + phase)	**0.7950**	**0.9007**

Table 5: Gating component ablation.

Gating Variant	CLIP	CSR
Discrepancy-only gate	0.7803	0.8762
Synergy-only gate	0.7779	0.8734
Uniform gate ($a_m = 0.5$)	0.7698	0.8681
Random gate	0.7643	0.8627
Ours (combined gate)	**0.7950**	**0.9007**

mis-calibrated gating is harmful rather than neutral, so the gain depends on the framework-specific values inferred from offline statistics. All gating variants in Table 5 use the same two-phase schedule as our full method.

Phase Partition Ablation. Table 6 evaluates whether the partition itself matters, beyond the presence of multiple training phases. The single-phase baseline mixes all frameworks and therefore exposes the shared Adapter to conflicting output patterns throughout training. Random or manually altered two-phase partitions perform worse than the automatically selected schedule, indicating that grouping decisions cannot be reduced to arbitrary batching. Increasing to $M = 3$ remains competitive but does not surpass the selected two-phase schedule, while fully separating all frameworks weakens useful cross-framework transfer.

Table 6: Phase partition ablation.

Phase Configuration	CLIP	CSR
$M = 1$	0.7608	0.8649
$M = 2$: reversed order	0.7817	0.8789
$M = 2$: Vanilla+React / Vue+Angular	0.7834	0.8812
$M = 3$	0.7921	0.8951
$M = 4$	0.7801	0.8709
Ours ($M = 2$)	**0.7950**	**0.9007**

Table 7: Synergy estimation strategy ablation.

Synergy Variant	CLIP	CSR	Extra Cost
Online gradient synergy	0.7916	0.8908	$+23\%$
No synergy ($S = \mathbf{0}$)	0.7782	0.8745	0
Ours (offline pre-gradient)	**0.7950**	**0.9007**	$+2\%$

Synergy Estimation: Offline vs. Online. Table 7 compares the proposed offline pre-gradient estimate with online gradient statistics. The offline estimate achieves comparable performance while avoiding the substantial overhead of repeatedly collecting online gradient information during training. This supports the central assumption behind our compatibility measure. The no-synergy variant performs worse, confirming that phase construction needs a positive compatibility signal rather than relying only on distributional heterogeneity. Thus, offline pre-gradients provide a cost-effective proxy for deciding which frameworks should share training phases. Additional ablations on gradient layer selection, discrepancy distance function, and code normalization strength are reported in Appendix F.

4.4 Analysis

Discrepancy–Synergy Statistics. Figure 1 visualizes the two offline relationship matrices used by the phase selection algorithm. The matrices reveal that heterogeneity and compatibility are not redundant: React and Vue form the clearest compatible group, which explains why they are placed together in the first phase. Vanilla and Angular show a much larger discrepancy and weaker synergy; their grouping is therefore better interpreted as the residual second phase under a two-phase budget, with subsequent gating disabling shared updates for the most isolated framework.

Gating Distribution and Theoretical Correlation. Figure 2 shows that the framework-level gates reflect the offline relationship scores: React and Vue, which are mutually compatible and close in normalized code distributions, receive the largest gates (0.96 and 0.95), Vanilla an intermediate gate (0.49), and Angular a fully closed gate (0.00), as it is simultaneously the most heterogeneous and the least compatible framework, so its training relies entirely on the framework-specific Adapter. The

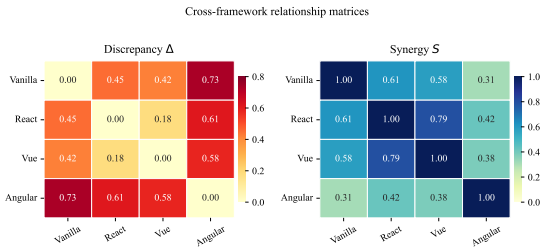

Figure 1: Offline discrepancy (Δ, i.e., heterogeneity, left) and synergy (S, i.e., compatibility, right) matrices.

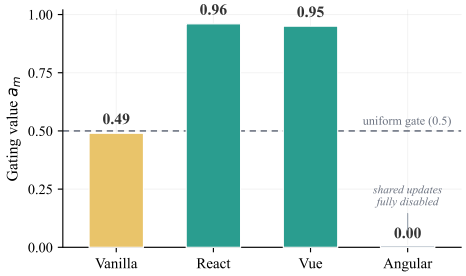

Figure 2: Per-framework gating values a_m derived deterministically from the offline statistics; the dashed line marks the uniform gate (a_m=0.5).

observed gate pattern qualitatively aligns with the framework-wise gains over the no-gate baseline, providing empirical support for the gating mechanism.

4.5 Error Analysis

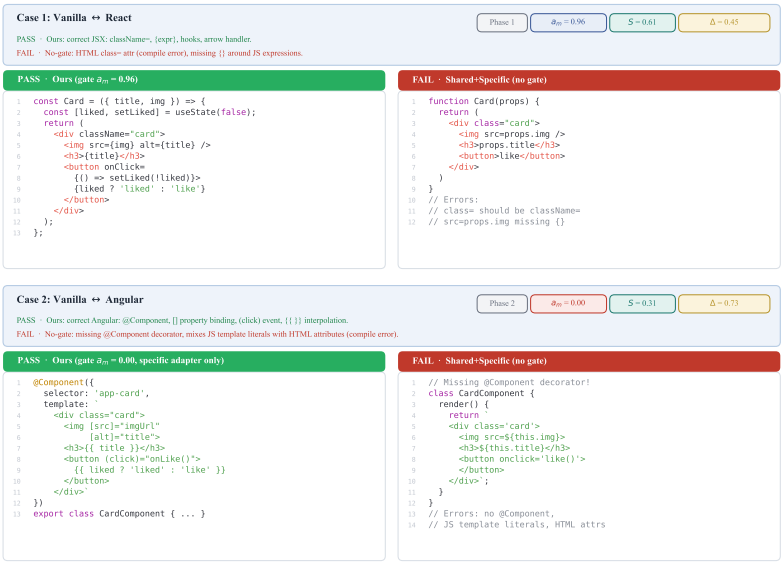

Figure 3: Qualitative code generation examples stratified by (Δ, S). Each row shows our method (center, green border) vs. Shared+Specific no-gate baseline (right, red border).

Qualitative Examples. Figure 3 presents representative examples stratified by (Δ, S) values. The top row (Vanilla vs. React, $\Delta = 0.45$, $S = 0.61$) shows our method correctly generating JSX functional components with hooks while preserving shared structural layout. The bottom row (Vanilla vs. Angular, $\Delta = 0.73$, $S = 0.31$) shows the gating mechanism fully disabling shared updates ($a_m = 0.00$) so the specific Adapter correctly captures Angular template binding syntax.

5 Conclusion

We presented a phase-wise adapter tuning method for multi-framework WebUI code generation, using offline heterogeneity from normalized code distributions and compatibility from shared-adapter pre-gradient alignment to guide framework partitioning. The resulting training schedule, together with lightweight per-framework gating, reduces cross-framework interference without requiring additional online gradient computation.

References

[1] C. Si, Y. Zhang, R. Li, Z. Yang, R. Liu, and D. Yang, "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, pp. 3956–3974.

[2] T. Zhou, Y. Zhao, X. Hou, X. Sun, K. Chen, and H. Wang, "Declarui: Bridging design and development with automated declarative ui code generation," *Proceedings of the ACM on Software Engineering*, vol. 2, no. FSE, pp. 219–241, 2025.

[3] T. A. Nguyen and C. Csallner, "Reverse engineering mobile application user interfaces with remaui (t)," in *2015 30th IEEE/ACM international conference on automated software engineering (ASE)*. IEEE, 2015, pp. 248–259.

[4] T. Beltramelli, "pix2code: Generating code from a graphical user interface screenshot," in *Proceedings of the ACM SIGCHI symposium on engineering interactive computing systems*, 2018, pp. 1–6.

[5] C. Chen, T. Su, G. Meng, Z. Xing, and Y. Liu, "From ui design image to gui skeleton: a neural machine translator to bootstrap mobile gui implementation," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 665–676.

[6] Y. Xu, L. Bo, X. Sun, B. Li, J. Jiang, and W. Zhou, "image2emmet: Automatic code generation from web user interface image," *Journal of Software: Evolution and Process*, vol. 33, no. 8, p. e2369, 2021.

[7] H. Laurençon, L. Tronchon, and V. Sanh, "Unlocking the conversion of web screenshots into html code with the websight dataset," *arXiv preprint arXiv:2403.09029*, 2024.

[8] S. Yun, R. Thushara, M. Bhat, Y. Wang, M. Deng, J. Wang, T. Tao, J. Li, H. Li, P. Nakov *et al.*, "Web2code: A large-scale webpage-to-code dataset and evaluation framework for multimodal llms," *Advances in neural information processing systems*, vol. 37, pp. 112 134–112 157, 2024.

[9] Y. Gui, Y. Wan, Z. Li, Z. Zhang, D. Chen, H. Zhang, Y. Su, B. Chen, X. Zhou, W. Jiang *et al.*, "Uicopilot: Automating ui synthesis via hierarchical code generation from webpage designs," in *Proceedings of the ACM on Web Conference 2025*, 2025, pp. 1846–1855.

[10] Y. Gui, Z. Li, Y. Wan, Y. Shi, H. Zhang, B. Chen, Y. Su, D. Chen, S. Wu, X. Zhou *et al.*, "Webcode2m: A real-world dataset for code generation from webpage designs," in *Proceedings of the ACM on Web Conference 2025*, 2025, pp. 1834–1845.

[11] J. Xiao, M. Wang, M. H. Lam, Y. Wan, J. Liu, Y. Huo, and M. R. Lyu, "Designbench: A comprehensive benchmark for mllm-based front-end code generation," *arXiv preprint arXiv:2506.06251*, 2025.

[12] D. Kollias, V. Sharmanska, and S. Zafeiriou, "Distribution matching for heterogeneous multitask learning: a large-scale face study," *arXiv preprint arXiv:2105.03790*, 2021.

[13] T. Yu, S. Kumar, A. Gupta, S. Levine, K. Hausman, and C. Finn, "Gradient surgery for multi-task learning," *Advances in neural information processing systems*, vol. 33, pp. 5824–5836, 2020.

[14] J. Wu, E. Schoop, A. Leung, T. Barik, J. P. Bigham, and J. Nichols, "Uicoder: Finetuning large language models to generate user interface code through automated feedback," in *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, 2024, pp. 7511–7525.

[15] O. Sener and V. Koltun, "Multi-task learning as multi-objective optimization," *Advances in neural information processing systems*, vol. 31, 2018.

[16] B. Liu, X. Liu, X. Jin, P. Stone, and Q. Liu, "Conflict-averse gradient descent for multi-task learning," *Advances in neural information processing systems*, vol. 34, pp. 18 878–18 890, 2021.

[17] T. Standley, A. Zamir, D. Chen, L. Guibas, J. Malik, and S. Savarese, "Which tasks should be learned together in multi-task learning?" in *International conference on machine learning*. PMLR, 2020, pp. 9120–9132.

[18] C. Fifty, E. Amid, Z. Zhao, T. Yu, R. Anil, and C. Finn, "Efficiently identifying task groupings for multi-task learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 27 503–27 516, 2021.

[19] A. A. J. Cizotto, R. C. T. De Souza, V. C. Mariani, and L. dos Santos Coelho, "Web pages from mockup design based on convolutional neural network and class activation mapping," *Multimedia Tools and Applications*, vol. 82, no. 25, pp. 38 771–38 797, 2023.

[20] J. Wu, X. Zhang, J. Nichols, and J. P. Bigham, "Screen parsing: Towards reverse engineering of ui models from screenshots," in *The 34th Annual ACM Symposium on User Interface Software and Technology*, 2021, pp. 470–483.

[21] Y. Gui, Z. Li, Z. Zhang, G. Wang, T. Lv, G. Jiang, Y. Liu, D. Chen, Y. Wan, H. Zhang *et al.*, "Latcoder: Converting webpage design to code with layout-as-thought," in *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, 2025, pp. 721–732.

[22] Y. Wan, C. Wang, Y. Dong, W. Wang, S. Li, Y. Huo, and M. Lyu, "Divide-and-conquer: Generating ui code from screenshots," *Proceedings of the ACM on Software Engineering*, vol. 2, no. FSE, pp. 2099–2122, 2025.

[23] F. Wu, C. Gao, S. Li, X.-C. Wen, and Q. Liao, "Mllm-based ui2code automation guided by ui layout information," *Proceedings of the ACM on Software Engineering*, vol. 2, no. ISSTA, pp. 1123–1145, 2025.

[24] J. Xiao, Z. Zhang, Y. Wan, Y. Huo, Y. Liu, and M. R. Lyu, "Efficientuicoder: Efficient mllm-based ui code generation via input and output token compression," *arXiv preprint arXiv:2509.12159*, 2025.

[25] X. Zhao, D. Jiang, Z. Zeng, L. Chen, H. Qiu, J. Huang, Y. Zhong, L. Zheng, Y. Cao, and L. Ma, "Vincicoder: Unifying multimodal code generation via coarse-to-fine visual reinforcement learning," *arXiv preprint arXiv:2511.00391*, 2025.

[26] L. Chen, X. Zhao, Z. Zeng, J. Huang, L. Zheng, Y. Zhong, and L. Ma, "Breaking the sft plateau: Multimodal structured reinforcement learning for chart-to-code generation," *arXiv preprint arXiv:2508.13587*, 2025.

[27] J. Wu, H. Ma, S. Tang, Y. Xu, X. Kang, and J. Gao, "MulFCoder: Framework-conditioned multi-agent for MLLM-based multi-framework front-end code generation," in *Forty-third International Conference on Machine Learning*, 2026. [Online]. Available: https://openreview.net/forum?id=xUW3aH4ie3

[28] J.-B. Alayrac, J. Donahue, P. Luc, A. Miech, I. Barr, Y. Hasson, K. Lenc, A. Mensch, K. Millican, M. Reynolds *et al.*, "Flamingo: a visual language model for few-shot learning," *Advances in neural information processing systems*, vol. 35, pp. 23 716–23 736, 2022.

[29] J. Li, D. Li, S. Savarese, and S. Hoi, "Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models," in *International conference on machine learning*. PMLR, 2023, pp. 19 730–19 742.

[30] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Visual instruction tuning," *Advances in neural information processing systems*, vol. 36, pp. 34 892–34 916, 2023.

[31] L. Chen, J. Li, X. Dong, P. Zhang, C. He, J. Wang, F. Zhao, and D. Lin, "Sharegpt4v: Improving large multi-modal models with better captions," in *European Conference on Computer Vision*. Springer, 2024, pp. 370–387.

[32] H. Liu, C. Li, Y. Li, B. Li, Y. Zhang, S. Shen, and Y. J. Lee, "Llavanext: Improved reasoning, ocr, and world knowledge," 2024.

[33] G. Team, P. Georgiev, V. I. Lei, R. Burnell, L. Bai, A. Gulati, G. Tanzer, D. Vincent, Z. Pan, S. Wang *et al.*, "Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context," *arXiv preprint arXiv:2403.05530*, 2024.

[34] J. Zhu, W. Wang, Z. Chen, Z. Liu, S. Ye, L. Gu, H. Tian, Y. Duan, W. Su, J. Shao *et al.*, "Internvl3: Exploring advanced training and test-time recipes for open-source multimodal models," *arXiv preprint arXiv:2504.10479*, 2025.

[35] N. Ding, Y. Qin, G. Yang, F. Wei, Z. Yang, Y. Su, S. Hu, Y. Chen, C.-M. Chan, W. Chen *et al.*, "Parameter-efficient fine-tuning of large-scale pre-trained language models," *Nature machine intelligence*, vol. 5, no. 3, pp. 220–235, 2023.

[36] V. Lialin, V. Deshpande, X. Yao, and A. Rumshisky, "Scaling down to scale up: A guide to parameter-efficient fine-tuning," *arXiv preprint arXiv:2303.15647*, 2023.

[37] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly, "Parameter-efficient transfer learning for nlp," in *International conference on machine learning*. PMLR, 2019, pp. 2790–2799.

[38] X. L. Li and P. Liang, "Prefix-tuning: Optimizing continuous prompts for generation," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2021, pp. 4582–4597.

[39] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen *et al.*, "Lora: Low-rank adaptation of large language models." *Iclr*, vol. 1, no. 2, p. 3, 2022.

[40] E. Xie, L. Yao, H. Shi, Z. Liu, D. Zhou, Z. Liu, J. Li, and Z. Li, "Difffit: Unlocking transferability of large diffusion models via simple parameter-efficient fine-tuning," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 4230–4239.

[41] Y.-L. Sung, J. Cho, and M. Bansal, "Vl-adapter: Parameter-efficient transfer learning for vision-and-language tasks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 5227–5237.

[42] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "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, pp. 4171–4186.

[43] J. Pfeiffer, A. Kamath, A. Rücklé, K. Cho, and I. Gurevych, "Adapterfusion: Non-destructive task composition for transfer learning," in *Proceedings of the 16th conference of the European chapter of the association for computational linguistics: main volume*, 2021, pp. 487–503.

[44] Anthropic, "Claude 3.7 sonnet," Anthropic, Tech. Rep., 2025. [Online]. Available: https://www.anthropic.com/claude/claude-3-7-sonnet

[45] OpenAI, "GPT-4o," OpenAI, Tech. Rep., 2024. [Online]. Available: https://openai.com/index/hello-gpt-4o/

[46] Google DeepMind, "Gemini 2.0 flash," Google, Tech. Rep., 2025. [Online]. Available: https://deepmind.google/technologies/gemini/flash/

[47] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[48] S. Bai, K. Chen, X. Liu, J. Wang, W. Ge, S. Song, K. Wang, S. Wang, P. Tang, Y. Wen *et al.*, "Qwen2.5-vl technical report," *arXiv preprint arXiv:2502.13923*, 2025.

A Algorithm process

Algorithm 1 Phase-wise adapter tuning

Require: Frozen MLLM parameters Θ; framework datasets $\{\mathcal{D}_j\}_{j=1}^{k}$; shared adapter w^s; framework-specific adapters $\{w_j\}_{j=1}^{k}$; number of phases M.

Compute normalized code distributions and heterogeneity scores $\Delta_{i,j}$ for all framework pairs.
Compute offline pre-gradients $\{\bar{g}_j\}_{j=1}^{k}$ and compatibility scores $S(\mathcal{D}_i, \mathcal{D}_j)$.
Construct the feasible partition set using $\bar{m}(P_t) > C_{P_t}\epsilon_g$.
Select $\Pi^\star = (P_1^\star, \ldots, P_M^\star)$ by maximizing G in Eq. (13).
Compute framework gates $\{a_m\}_{m=1}^{k}$ from average heterogeneity and compatibility.
for $t = 1$ to M **do**
    Sample training batches only from frameworks in $P_t^\star$.
    Update the shared adapter w^s using the gate-weighted shared update.
    Update framework-specific adapters $\{w_j : j \in P_t^\star\}$.
    Keep framework-specific adapters $\{w_j : j \notin P_t^\star\}$ frozen.
end for
return tuned adapters $w^s, \{w_j\}_{j=1}^{k}$.

B Construction of Normalized Code Distributions

To estimate output-space differences across domains in a stable and reproducible manner, we do not construct empirical distributions directly from raw code strings. Instead, we first normalize the code and then estimate its n-gram statistics. For a target code sample y from domain $\mathcal{D}_j$, let its normalized form be denoted by

$$\widetilde{y} = \mathrm{Norm}(y). \tag{16}$$

The normalization mapping $\mathrm{Norm}(\cdot)$ consists of four steps. First, we standardize whitespace, indentation, line breaks, and quotation styles, and remove comments or formatting artifacts that do not affect functionality. This reduces the sensitivity of distribution estimation to code style. Second, we abstract away local variable names, temporary function names, auto-generated class names, and random identifiers, so that free naming choices do not dominate the resulting statistics. Third, we perform lightweight normalization for local patterns that are semantically equivalent but syntactically heterogeneous across domains, which improves comparability while preserving the essential code structure. Finally, we tokenize the normalized code and estimate its empirical unigram and bigram distributions.

Formally, let $Q_j^{(1)}$ and $Q_j^{(2)}$ denote the unigram and bigram empirical distributions of domain $\mathcal{D}_j$ lifted to a shared disjoint support, respectively. We define empirical code distribution of domain j as

$$Q_j = (1-\omega)Q_j^{(1)} + \omega Q_j^{(2)}, \qquad \omega \in [0,1]. \tag{17}$$

By default, we set $\omega = \frac{1}{2}$.

Given two domains $\mathcal{D}_i$ and $\mathcal{D}_j$, their discrepancy is computed by the normalized Jensen–Shannon divergence

$$\Delta_{i,j} = \frac{1}{\log 2}\left[\frac{1}{2}\mathrm{KL}(Q_i \| M_{ij}) + \frac{1}{2}\mathrm{KL}(Q_j \| M_{ij})\right], \qquad M_{ij} = \frac{1}{2}(Q_i + Q_j), \tag{18}$$

which ensures that $\Delta_{i,j} \in [0,1]$.

The purpose of this construction is not to remove all output differences across domains, but to suppress superficial perturbations that are irrelevant to the task while preserving the statistics that reflect target code patterns, structural motifs, and local compositional regularities. The resulting Q_j is therefore both computable and stable, making it a suitable proxy for the discrepancy term in stage partition.

C Proofs and Justifications for Offline Compatibility

C.1 Proof of Lemma 3.1

For a fixed phase P, recall the shared update

$$w^{s,+} = w^s - \eta g_P, \qquad g_P = \sum_{j \in P} \alpha_j g_j. \tag{19}$$

Fix an arbitrary framework $\mathcal{D}_j \in P$ and define

$$F_j(w^s) \triangleq \mathcal{L}(\mathcal{D}_j; \Theta, W), \tag{20}$$

where all non-shared parameters are fixed at their current values. Since F_j is L_s-smooth with respect to w^s, we have

$$F_j(w^{s,+}) \le F_j(w^s) + \langle \nabla_{w^s} F_j(w^s),\, w^{s,+} - w^s \rangle + \frac{L_s}{2} \|w^{s,+} - w^s\|_2^2. \tag{21}$$

Substituting

$$w^{s,+} - w^s = -\eta g_P \tag{22}$$

and using

$$\nabla_{w^s} F_j(w^s) = g_j, \tag{23}$$

we obtain

$$F_j(w^{s,+}) \le F_j(w^s) - \eta \langle g_j, g_P \rangle + \frac{L_s \eta^2}{2} \|g_P\|_2^2. \tag{24}$$

By the definition of $m(P)$,

$$\langle g_j, g_P \rangle \ge m(P) \|g_P\|_2^2, \qquad \forall j \in P. \tag{25}$$

Therefore,

$$F_j(w^{s,+}) \le F_j(w^s) - \eta m(P) \|g_P\|_2^2 + \frac{L_s \eta^2}{2} \|g_P\|_2^2. \tag{26}$$

Rearranging yields

$$F_j(w^{s,+}) - F_j(w^s) \le \left(-\eta m(P) + \frac{L_s \eta^2}{2} \right) \|g_P\|_2^2. \tag{27}$$

Hence, whenever

$$\eta \le \frac{2\, m(P)}{L_s}, \tag{28}$$

the right-hand side is non-positive, which implies

$$\mathcal{L}(\mathcal{D}_j; \Theta, W^+) \le \mathcal{L}(\mathcal{D}_j; \Theta, W), \qquad \forall j \in P. \tag{29}$$

This proves Lemma 3.1.

C.2 Why can offline gradients serve as surrogates of online shared gradients?

The key observation is that the offline gradient is computed at exactly the same insertion location as the shared adapter used in training, namely the output of layer $r-1$. Even before the shared adapter is explicitly optimized, the loss still admits a well-defined first-order sensitivity with respect to a virtual perturbation at that location.

Proposition C.1 (Virtual perturbation equivalence)**.** Let $H^{r-1}(x)$ denote the hidden representation at the output of layer $r-1$. Introduce a virtual perturbation variable Δ at the shared insertion point and consider the perturbed representation

$$H^{r-1}(x) \mapsto H^{r-1}(x) + \Delta. \tag{30}$$

Then, at $\Delta = 0$, the gradient of the loss with respect to the perturbation equals the gradient with respect to a zero-initialized additive shared adapter:

$$\nabla_\Delta \mathcal{L}(\mathcal{D}_j; \Theta, \Delta)\Big|_{\Delta=0} = \nabla_{w^s} \mathcal{L}(\mathcal{D}_j; \Theta, W)\Big|_{w^s=0,\, w=0}. \tag{31}$$

Proof. Consider the additive shared adapter

$$A_{w^s}(H^{r-1}) = H^{r-1} + w^s. \tag{32}$$

At initialization $w^s = 0$, the adapter output coincides with the unmodified representation, and the Jacobian of A_{w^s} with respect to w^s is the identity map. By the chain rule,

$$\nabla_{w^s}\mathcal{L} = \left(\frac{\partial A_{w^s}(H^{r-1})}{\partial w^s}\right)^\top \nabla_{A_{w^s}(H^{r-1})}\mathcal{L}. \tag{33}$$

Evaluating at $w^s = 0$ gives

$$\nabla_{w^s}\mathcal{L}\Big|_{w^s=0} = \nabla_{H^{r-1}}\mathcal{L}, \tag{34}$$

which is exactly the gradient with respect to the virtual perturbation Δ at $\Delta = 0$. □

Proposition C.1 explains why the offline shared gradient remains well-defined even before the shared adapter is explicitly trained.

Proposition C.2 (Sample-average characterization)**.** Assume that differentiation and expectation can be interchanged. Then the framework-level offline shared gradient satisfies

$$\bar{g}_j = \nabla_{w^s}\mathbb{E}_{(x,y)\sim\mathcal{D}_j}\left[\ell(x,y;\Theta,W)\right]\Big|_{w^s=0,\ w=0} = \mathbb{E}_{(x,y)\sim\mathcal{D}_j}\left[\nabla_{w^s}\ell(x,y;\Theta,W)\Big|_{w^s=0,\ w=0}\right]. \tag{35}$$

Proof. The result follows directly from the interchange of differentiation and expectation applied to the empirical loss over $\mathcal{D}_j$. □

Proposition C.2 justifies estimating $\bar{g}_j$ by averaging sample-wise gradients within each framework.

Proposition C.3 (First-order consistency for general shared adapters)**.** Suppose the shared adapter inserted at layer $r-1$ admits the local expansion

$$A_{w^s}(H^{r-1}) = H^{r-1} + J(H^{r-1})\, w^s + \mathcal{O}(\|w^s\|_2^2) \tag{36}$$

around $w^s = 0$, where $J(H^{r-1})$ is the Jacobian of the adapter output with respect to w^s. Then

$$\nabla_{w^s}\mathcal{L}(\mathcal{D}_j;\Theta,W)\Big|_{w^s=0,\ w=0} = J(H^{r-1})^\top \nabla_{H^{r-1}}\mathcal{L}(\mathcal{D}_j;\Theta). \tag{37}$$

Hence the hidden-state sensitivity at the insertion point determines the first-order shared update direction up to a fixed linear projection.

Proof. By the chain rule,

$$\nabla_{w^s}\mathcal{L} = \left(\frac{\partial A_{w^s}(H^{r-1})}{\partial w^s}\right)^\top \nabla_{A_{w^s}(H^{r-1})}\mathcal{L}. \tag{38}$$

Evaluating this identity at $w^s = 0$ and using the stated local expansion yields (37). □

Proposition C.3 shows that the offline gradient proxy is not restricted to a purely additive adapter parameterization. It remains valid for any shared adapter that is linearizable around the initialization point.

Proposition C.4 (Gradient drift bound under bounded adapter updates)**.** Assume that, for each framework $\mathcal{D}_j$, the shared gradient $\nabla_{w^s}\mathcal{L}(\mathcal{D}_j;\Theta,W)$ is L_g-Lipschitz with respect to W. If the adapter trajectory stays in a bounded neighborhood,

$$\|W\|_2 \le \rho, \tag{39}$$

then for every framework j,

$$\|g_j - \bar{g}_j\|_2 \le L_g\rho. \tag{40}$$

Proof. By the L_g-Lipschitz continuity of the shared gradient,

$$\|\nabla_{w^s}\mathcal{L}(\mathcal{D}_j;\Theta,W)-\nabla_{w^s}\mathcal{L}(\mathcal{D}_j;\Theta,0)\|_2\le L_g\|W-0\|_2. \tag{41}$$

Since

$$\bar{g}_j=\nabla_{w^s}\mathcal{L}(\mathcal{D}_j;\Theta,W)\Big|_{w^s=0,\ w=0}, \tag{42}$$

the claim follows from $\|W\|_2\le\rho$. □

Proposition C.4 explains why the offline gradient can serve as a meaningful surrogate of the online shared gradient in the frozen-backbone regime. Since only adapters are updated and their trajectory is norm-controlled, the online shared gradients remain close to their initialization counterparts.

Proposition C.5 (From offline margin to online margin)**.** For a phase P, define

$$\bar{g}_P=\sum_{j\in P}\alpha_j\bar{g}_j,\qquad \bar{m}(P)=\min_{j\in P}\frac{\langle\bar{g}_j,\bar{g}_P\rangle}{\|\bar{g}_P\|_2^2}. \tag{43}$$

Assume that $\|g_j-\bar{g}_j\|_2\le\varepsilon_g$ for all $j\in P$, and that there exist constants

$$\|\bar{g}_j\|_2\le G_P,\qquad \|\bar{g}_P\|_2\ge\gamma_P>\varepsilon_g. \tag{44}$$

Then there exists a phase-dependent constant $C_P>0$ such that

$$m(P)\ge\bar{m}(P)-C_P\varepsilon_g. \tag{45}$$

In particular, if $\bar{m}(P)>C_P\varepsilon_g$, then $m(P)>0$.

Proof. Let $\Delta_j=g_j-\bar{g}_j$ and $\Delta_P=g_P-\bar{g}_P=\sum_{j\in P}\alpha_j\Delta_j$. Then $\|\Delta_P\|_2\le\varepsilon_g$. For any $j\in P$,

$$\langle g_j,g_P\rangle=\langle\bar{g}_j,\bar{g}_P\rangle+\langle\Delta_j,\bar{g}_P\rangle+\langle\bar{g}_j,\Delta_P\rangle+\langle\Delta_j,\Delta_P\rangle. \tag{46}$$

Therefore,

$$|\langle g_j,g_P\rangle-\langle\bar{g}_j,\bar{g}_P\rangle|\le\varepsilon_g(\gamma_P+G_P)+\varepsilon_g^2. \tag{47}$$

Moreover,

$$\|g_P\|_2\ge\|\bar{g}_P\|_2-\|\Delta_P\|_2\ge\gamma_P-\varepsilon_g, \tag{48}$$

and thus

$$\|g_P\|_2^2\ge(\gamma_P-\varepsilon_g)^2. \tag{49}$$

Combining the perturbation bound for the numerator with the lower bound for the denominator yields

$$\frac{\langle g_j,g_P\rangle}{\|g_P\|_2^2}\ge\frac{\langle\bar{g}_j,\bar{g}_P\rangle}{\|\bar{g}_P\|_2^2}-C_P\varepsilon_g \tag{50}$$

for some constant C_P depending only on (G_P,γ_P). Taking the minimum over $j\in P$ proves (45). □

This justifies using (7) and (8) during phase construction before the actual multi-phase optimization begins.

D Proofs of the Generalization Results

D.1 Proof of Theorem 3.1

Proof. Fix a candidate phase $P\subseteq\{1,\dots,k\}$ and define the phase-level weighted risk

$$\mathcal{L}_P(\Theta,W)=\sum_{j\in P}\alpha_j\mathcal{L}(\mathcal{D}_j;\Theta,W),\qquad \widehat{\mathcal{L}}_P(\Theta,W)=\sum_{j\in P}\alpha_j\widehat{\mathcal{L}}(\mathcal{D}_j;\Theta,W). \tag{51}$$

By Assumption 3.1 and the bounded adapter complexity condition, the admissible adapter class has bounded effective capacity. Standard uniform-convergence arguments for Lipschitz losses over norm-restricted parameter classes imply that, with probability at least $1-\delta$,

$$\mathcal{L}_P(\Theta,W)\le\widehat{\mathcal{L}}_P(\Theta,W)+\Gamma(P)+O\left(\sqrt{\frac{\ln(1/\delta)}{n_P}}\right), \tag{52}$$

where

$$\Gamma(P) = 2L\gamma \left(\beta + \sum_{j \in P} \alpha_j \beta \right). \tag{53}$$

Next, by the risk bridge assumption for proxy heterogeneity, the residual shift among frameworks inside the same phase is controlled by their pairwise heterogeneity. Absorbing constant normalizers into κ, we have

$$\sum_{j \in P} \alpha_j \mathcal{L}(\mathcal{D}_j; \Theta, W) \leq \sum_{j \in P} \alpha_j \widehat{\mathcal{L}}(\mathcal{D}_j; \Theta, W) + \Gamma(P) + \kappa \sum_{\substack{i<j \\ i,j \in P}} h(\mathcal{D}_i, \mathcal{D}_j) + \epsilon_h + O\left(\sqrt{\frac{\ln(1/\delta)}{n_P}} \right). \tag{54}$$

This is exactly the claimed phase-wise concurrent generalization bound. □

D.2 Proof of Theorem 3.2

Proof. Let $\Pi = (P_1, \ldots, P_M)$ be any feasible phase partition satisfying

$$\bar{m}(P_t) > C_{P_t} \epsilon_g, \qquad \forall t. \tag{55}$$

By Proposition C.5, this condition implies $m(P_t) > 0$ for every phase. Lemma 3.1 then guarantees that the aggregate shared-adapter update in phase P_t is descent-compatible for all frameworks in that phase.

For a fixed phase P_t, applying Theorem 3.1 gives, with high probability,

$$\sum_{j \in P_t} \alpha_j^t \mathcal{L}(\mathcal{D}_j; \Theta, W^t) \leq \sum_{j \in P_t} \alpha_j^t \widehat{\mathcal{L}}(\mathcal{D}_j; \Theta, W^t) + \Gamma(P_t) + \kappa \sum_{\substack{i<j \\ i,j \in P_t}} h(\mathcal{D}_i, \mathcal{D}_j) + \epsilon_h + O\left(\sqrt{\frac{\ln(1/\delta)}{n_t}} \right). \tag{56}$$

The norm-based capacity term $\mathrm{Cap}(P_t)$ upper bounds the complexity contribution and the phase-wise adapter budget. Moreover, by the compatibility calibration condition, the descent benefit associated with compatible shared updates is lower bounded by

$$\lambda \sum_{\substack{i<j \\ i,j \in P_t}} S(\mathcal{D}_i, \mathcal{D}_j). \tag{57}$$

Therefore the phase risk is upper bounded by

$$\sum_{j \in P_t} \alpha_j^t \mathcal{L}(\mathcal{D}_j; \Theta, W^t) \leq C_t + \sum_{\substack{i<j \\ i,j \in P_t}} h(\mathcal{D}_i, \mathcal{D}_j) - \lambda \sum_{\substack{i<j \\ i,j \in P_t}} S(\mathcal{D}_i, \mathcal{D}_j) + \mathrm{Cap}(P_t) + O\left(\sqrt{\frac{\ln(1/\delta)}{n_t}} \right), \tag{58}$$

where C_t absorbs the empirical loss bound, $\Gamma(P_t)$, and constants independent of the partition, $\mathrm{Cap}(P_t) = \mu_s \|w^{s,t}\|_2^2 + \mu_f \sum_{j \in P_t} \|w_j^t\|_2^2$.

Taking the maximum over phases and using $\max_t a_t \leq \sum_t a_t$, we obtain

$$R_{\max}(\Pi) \leq 1 - G(\Pi) + O\left(\sqrt{\frac{\ln(1/\delta)}{N}} \right), \tag{59}$$

where $N = \sum_{j=1}^{k} |\mathcal{D}_j|$.

Finally, since $\widetilde{G}$ is a monotone normalization of G over the feasible partition set, maximizing G and maximizing $\widetilde{G}$ yield the same partition. Hence the optimizer

$$\Pi^\star = \arg \max_{\Pi : \bar{m}(P_t) > C_{P_t} \epsilon_g, \forall t} G(\Pi) \tag{60}$$

minimizes the partition-dependent part of the upper bound. This gives

$$R_{\max}(\Pi^\star) \leq \left[1 - \widetilde{G}(\Pi^\star) \right]_+ + O\left(\sqrt{\frac{\ln(1/\delta)}{N}} \right), \tag{61}$$

which proves the theorem. □

E The details of Offline relationship-guided gating.

In addition to determining the phase partition, the offline heterogeneity and compatibility scores are also used to assign a framework-level shared-update gate. For each framework m, we compute its average heterogeneity and compatibility with the other frameworks:

$$\bar{\Delta}_m = \frac{1}{k-1} \sum_{i \neq m} h(\mathcal{D}_i, D_m), \qquad \bar{S}_m = \frac{1}{k-1} \sum_{i \neq m} S(\mathcal{D}_i, D_m). \tag{62}$$

The gate value $a_m \in [0, 1]$ is obtained from these two offline statistics through a fixed calibration rule:

$$a_m = \mathrm{Calib}(\bar{S}_m, \bar{\Delta}_m), \tag{63}$$

where $\mathrm{Calib}(\cdot, \cdot)$ is monotonic with respect to the two signals, increasing with $\bar{S}_m$ and decreasing with $\bar{\Delta}_m$. Thus, frameworks that are more compatible with the others and less heterogeneous receive larger shared-update weights, while more isolated frameworks rely more strongly on their framework-specific adapters. In our implementation, Calib is instantiated by first min-max normalizing $\bar{S}_m$ and $\bar{\Delta}_m$ across frameworks and then linearly combining them with equal weights, $a_m = \frac{1}{2}\,\hat{S}_m + \frac{1}{2}\,(1 - \hat{\Delta}_m)$, followed by clipping to $[0, 1]$, where $\hat{S}_m$ and $\hat{\Delta}_m$ denote the normalized signals. Because Angular is simultaneously the most heterogeneous ($\hat{\Delta}_m = 1$) and the least compatible ($\hat{S}_m = 0$) framework, this rule assigns it $a_m = 0$: its shared updates are fully disabled and it relies entirely on its framework-specific adapter.

During training, the framework identifier only performs a lookup over the precomputed gate table. For a training sample (x_i, y_i, f_i), let $m = \iota(f_i)$ be its framework index. The framework-specific adapter w_m is selected by this index, and the same index retrieves the scalar gate a_m for the shared adapter update:

$$w^s \leftarrow w^s - \eta\, a_m \nabla_{w^s} \ell(\mathcal{F}(x_i; \Theta, W), y_i), \tag{64}$$

while the framework-specific adapter is updated normally:

$$w_m \leftarrow w_m - \eta \nabla_{w_m} \ell(\mathcal{F}(x_i; \Theta, W), y_i). \tag{65}$$

This gate is computed offline and is not an input-dependent learned routing module.

F Additional Ablation Studies

Gradient Layer Position. Table 8 reports the effect of choosing different layers for offline synergy gradient computation. Our default layer 15 achieves the best result (0.7950 ± 0.0027) with the lowest synergy variance. Earlier layers (8–12) show higher variance (0.0063–0.0082), as low-level visual features carry little framework-specific signal, whereas layers within the Adapter insertion range (18–22) perform worse (CLIP ≤ 0.7823) and also exhibit elevated estimate variance (0.0048–0.0062), suggesting that compatibility estimated from deep, decoding-stage-dominated representations is less stable; layer 15 balances visual-semantic abstraction and task specificity.

Table 8: Effect of gradient computation layer (mean±std over 3 seeds).

Layer	CLIP (Avg.)	CSR (Avg.)	Synergy Std.
Layer 8	0.7763±0.0079	0.8714±0.0061	0.0082
Layer 10	0.7798±0.0065	0.8741±0.0051	0.0071
Layer 12	0.7878±0.0049	0.8802±0.0043	0.0063
Layer 14	0.7924±0.0033	0.8917±0.0030	0.0041
Layer 15 (ours)	**0.7950±0.0027**	**0.9007±0.0024**	0.0031
Layer 18	0.7823±0.0038	0.8771±0.0035	0.0048
Layer 20	0.7774±0.0044	0.8728±0.0040	0.0053
Layer 22	0.7736±0.0056	0.8698±0.0051	0.0062

Discrepancy Distance Function. Table 9 compares three choices of distance function $d(\cdot, \cdot)$ for computing Δ. JS Divergence (our default) achieves the best mean (0.7950 ± 0.0027). Sinkhorn-Wasserstein yields a comparable mean (0.7931 ± 0.0030) but incurs $3.7\times$ the offline computation time. MMD shows slightly lower performance (0.7912 ± 0.0033), attributed to sensitivity to the bandwidth hyperparameter on finite code token vocabularies.

Table 9: Discrepancy distance function ablation (mean±std over 3 seeds).

Distance Function	CLIP (Avg.)	CSR (Avg.)	Offline Comp.
MMD	0.7912 ± 0.0033	0.8901 ± 0.0029	$1.4\times$
Sinkhorn-Wasserstein	0.7931 ± 0.0030	0.8921 ± 0.0028	$3.7\times$
Ours (JS Divergence)	$\mathbf{0.7950 \pm 0.0027}$	$\mathbf{0.9007 \pm 0.0024}$	$1.0\times$

Code Normalization Strength. Table 10 ablates the strength of the normalization operator $\mathcal{N}_f$. Weak normalization (whitespace and formatting only) leads to noisier discrepancy estimates, degrading CLIP by 0.014 points. Strong normalization (additionally removing framework-specific syntax tokens) over-smooths the distribution, degrading CLIP by 0.018 points. Our default moderate normalization achieves the best balance.

Table 10: Code normalization strength ablation.

Normalization	CLIP (Avg.)	CSR (Avg.)
Weak (format only)	0.7812	0.8747
Strong (remove framework syntax)	0.7769	0.8694
Ours (moderate)	**0.7950**	**0.9007**

G Editing and Repair Task Results

This appendix reports results on the DesignBench Editing and Repair tasks. Both tasks use the MLLM-as-Judge scoring protocol from DesignBench [11], where a separate frontier MLLM (GPT-4o) evaluates the output on a 0–10 scale covering visual fidelity, structural correctness, and framework-specific syntax adherence.

Editing Task. The Editing task provides the model with an existing code snippet and a screenshot of the desired modification target. The model must produce code that matches the visual target while preserving the structural parts of the original code that are not subject to change. Table 11 reports MLLM-as-Judge scores for each framework.

Table 11: MLLM-as-Judge scores on the DesignBench Editing task (0–10 scale; mean±std over 3 seeds for fine-tuned methods).

Method	Vanilla	React	Vue	Angular	Avg.
Claude-3.7	7.42	7.83	7.61	5.94	7.20
GPT-4o	7.23	7.41	7.39	5.61	6.91
Qwen2.5-VL-7B	6.81	5.12	5.34	1.03	4.58
ScreenCoder	7.34 ± 0.08	7.52 ± 0.11	7.48 ± 0.09	6.12 ± 0.14	7.12 ± 0.10
EfficientUICoder	7.12 ± 0.09	7.23 ± 0.13	7.19 ± 0.11	5.87 ± 0.16	6.85 ± 0.12
Ours	$\mathbf{7.68 \pm 0.07}$	$\mathbf{7.94 \pm 0.09}$	$\mathbf{7.87 \pm 0.08}$	$\mathbf{6.89 \pm 0.12}$	$\mathbf{7.60 \pm 0.09}$

Our method achieves the best macro-average score (7.60), outperforming ScreenCoder by $+0.48$ points and the zero-shot Claude-3.7 baseline by $+0.40$ points. The most pronounced improvement is on Angular ($+0.77$ over ScreenCoder), where the framework-specific Adapter captures Angular's reactive binding syntax necessary for structurally faithful code edits. The gap between our method and Claude-3.7 is narrower on Editing than on Generation, likely because editing tasks require preserving existing code structure—a capability in which large frontier models retain an advantage through in-context pattern matching.

Repair Task. The Repair task provides a code snippet containing compilation errors and requires the model to produce a corrected, compilable version. Table 12 reports MLLM-as-Judge scores.

Table 12: MLLM-as-Judge scores on the DesignBench Repair task (0–10 scale; mean±std over 3 seeds for fine-tuned methods).

Method	**Vanilla**	**React**	**Vue**	**Angular**	**Avg.**
Claude-3.7	7.91	8.12	7.98	7.03	7.76
GPT-4o	7.78	7.93	7.82	6.87	7.60
Qwen2.5-VL-7B	7.34	6.23	6.47	2.14	5.55
ScreenCoder	7.62±0.08	7.81±0.10	7.74±0.09	6.58±0.13	7.44±0.10
EfficientUICoder	7.43±0.10	7.61±0.12	7.52±0.11	6.34±0.15	7.23±0.12
Ours	**8.03±0.06**	**8.24±0.08**	**8.17±0.07**	**7.52±0.11**	**7.99±0.08**

On the Repair task, our method achieves macro-average 7.99, which surpasses Claude-3.7 (7.76) by $+0.23$ points. The improvement is strongest on Angular ($+0.94$ over ScreenCoder), where compilation errors frequently involve template binding syntax errors that the framework-specific Adapter learns to correct reliably. The Repair task rewards compilation-aware generation—a property consistent with our compilation-oriented evaluation goal, where the gating mechanism reduces shared Adapter updates for frameworks with high discrepancy, preventing cross-framework syntax confusion that commonly leads to compilation failures.

H Discussions

Limitations. The proposed phase partition and gating strategy rely on offline heterogeneity and compatibility estimates, so their reliability may decrease when the training data are sparse, highly imbalanced, or substantially different from the target deployment distribution.

Broader Impacts. This work can reduce the engineering effort required to translate UI designs into executable frontend code, especially when developers need to support multiple frontend frameworks. It may also make rapid prototyping and framework migration more accessible to small teams and non-expert users.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction sections reflect the paper's contributions and scope in the paper.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Limitations are discussed in H.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: Full assumptions and proofs are provided in Appendix C and Appendix D.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Implementation details are given in Section 4.1, Appendix F, and Algorithm 1.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We will release our code after this paper is accepted.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: See Section 4.1 and Appendix F.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The reported results have run three times.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.

- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: See the Efficiency paragraph in Appendix F.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conforms to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Broader impacts are discussed in Appendix H.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: No high-risk assets are released.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All used public benchmarks and models are properly cited in the paper.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: New assets introduced in the paper are well documented and the documentation is provided alongside the assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve human-subject research.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: LLMs are used only as backbones, baselines, or automatic judges, not as a non-standard core methodological component.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.