Vision-Guided Iterative Refinement for Frontend Code Generation

Hannah Sansford, Derek H.C. Law, Wei Liu, Abhishek Tripathi, Niresh Agarwal, Gerrit J.J. van den Burg
Amazon AGI

amazon | science

Summary

The challenge: Human-in-the-loop refinement for frontend code generation is costly and difficult to scale. Code outputs require compilation and rendering for proper visual assessment, making automated improvement challenging.

Our Solution: We present a fully automated Critic-in-the-Loop (CITL) system where a vision-language model serves as a visual critic, providing **structured feedback** on rendered webpages to guide iterative code refinement. We:

- Use a vision-language model to analyze rendered webpages
- Separate visual assessment from code generation
- Observe progressive improvements over iteration cycles
- Apply LoRA finetuning to internalize improvements

Impact:
✓ 17.8% performance improvement over three refinement cycles
✓ Critic guidance demonstrated in the self-refinement and student-teacher setting
✓ LoRA finetuning captures 25% of CITL gains without iteration overhead
✓ Validated VLM-as-a-Judge with 69.5% agreement with human preferences
✓ Consistent improvement across all tested models

Automated Frontend Code Generation

Using LLMs to generate frontend code is an increasingly common task. However, improving frontend code requires an understanding of how code affects the **visual design of a webpage**. This understanding is typically provided through feedback from human developers.

But There's a Problem: the traditional human-in-the-loop approach to iteratively improving webpage designs is time-consuming and expensive, and remains bounded by human expertise

Why This Matters:
- Visual assessment is critical to understanding layout, styling, and aesthetics, which cannot be evaluated from code alone.
- Code-only evaluation misses visual task completion and aesthetic qualities.
- Automated feedback requires a multimodal approach that combines vision and code analysis

Our Solution: Critic-in-the-Loop

Main idea: Use a visual critic and a code critic to improve both aspects simultaneously

Components:
- **Generator (G):** Base LLM creates initial HTML code
- **Renderer (R):** Browser automation captures webpage screenshot
- **Visual Critic (V):** Vision-language model analyzes user request and rendered output
- **Code Critic (C):** Analyzes code and translates visual feedback into actionable code changes
- **Improver (I):** Generate refined code based on a consolidated critique
- **Evaluator (E):** Multi-dimensional scoring using VLM-as-a-Judge across 4 dimensions

Algorithm 1: Critic-in-the-Loop (CITL)

Input: Task x, iteration budget T
Output: Best generated code y_{best}

$y_0 \leftarrow G(x)$;
$s_0 \leftarrow E(x, y_0, R(y_0))$;
$y_{\text{best}} \leftarrow y_0$;
$s_{\text{best}} \leftarrow s_0$;
for $t = 0$ **to** $T - 1$ **do**
  $v_t \leftarrow V(x, R(y_t))$;
  $c_t \leftarrow C(x, y_t, v_t)$;
  $y_{t+1} \leftarrow I(x, y_t, c_t)$;
  $s_{t+1} \leftarrow E(x, y_{t+1}, R(y_{t+1}))$;
  if $s_{t+1} \geq s_{best}$ **then**
    $y_{\text{best}} \leftarrow y_{t+1}, \quad s_{\text{best}} \leftarrow s_{t+1}$;
return y_{best};

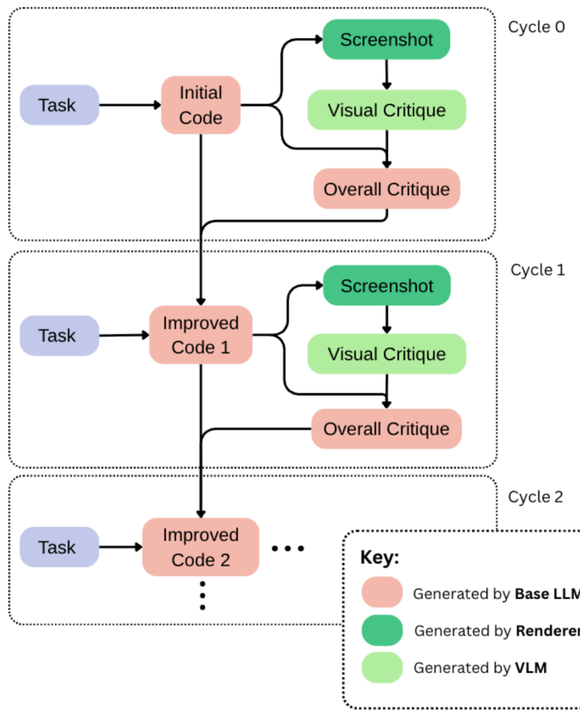
*Our CITL pipeline separates visual and code concerns, leveraging the strengths of the VLM and text-only models for distinct critiques of the generated solution, y_t. This allows for a student-teacher setting when the code generating LLM is a smaller model and the critic model is larger, or for a **self-refinement** setting when the same model is used as the generator, critic, and improver.*

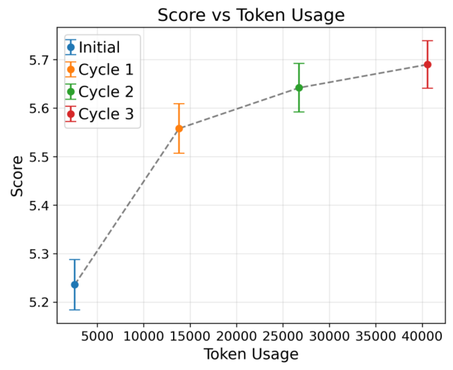
Evaluation: Multidimensional VLMaaJ

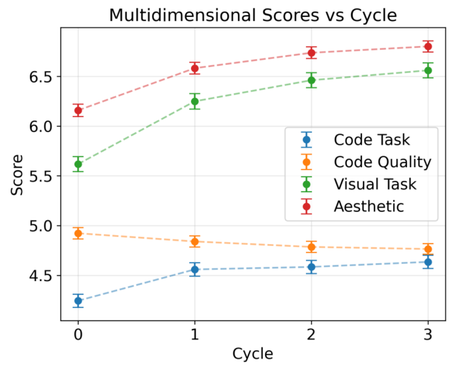
We develop a multidimensional VLM-as-a-Judge that considers task completion, aesthetics, and code quality. We validate this judge against human preferences on the WebDev Arena dataset and find that it has **43% higher agreement and 70% fewer ties** than a single-dimensional judge.

Outcome	Single-dimensional Judge	Multi-dimensional Judge
Agreement	48.5%	69.5%
Draw	28.5%	8.5%
Disagreement	23.0%	22.0%

System Overview







Overview of our **critic-in-the-loop approach** to iterative refinement for frontend code generation. Starting from the user task, we first prompt the **Base LLM** to generate the initial HTML/CSS code. This code is **rendered** into a screenshot and passed to a VLM model for **visual critique**. The visual critique is provided to a code critic that produces **consolidated feedback**. This feedback is given to the Base LLM alongside the user task and initial solution, leading to an **improved solution**. This process is repeated, leading to iterative refinement of the code. Figures on the right show continual improvement using the Distill-Qwen-14B model.

Experiments

We evaluate our system on 600 real-world user requests from WebDev Arena. We use Claude 4.5 Sonnet as the VLM in all cases and use Deepseek-R1-Distill-Qwen-14B, Claude 4.5 Haiku, and Claude 4.5 Sonnet as code generator models. The latter case is the self-improvement setting.

Generator	Cycle 0	Best Cycle	Improvement	Tokens
Distill-Qwen-14B	5.236	6.165	**17.8%**	23,662
Claude Haiku 4.5	7.436	8.160	**9.8%**	65,005
Claude Sonnet 4.5	7.584	8.402	**10.8%**	73,215

We find significant improvements for all generator models through our CITL pipeline. With Claude Sonnet 4.5 **in the self-improvement setting, we see that 86% of tasks benefit** from critic guidance.

Knowledge Distillation

To address the token costs of the CITL pipeline we consider using LoRA on the Distill-Qwen-14B model to internalize the improvements from the visual and code critiques.

Results:
- LoRA finetuning **captures 24.7% of the best CITL performance**
- Significantly reduces token overhead due to not requiring iterations
- Maintain efficient generation while improving quality
- Critic-in-the-loop provides lasting benefits beyond distillation

Key Takeaways

Our critic-in-the-loop approach leads to **consistent improvements in frontend code quality** by combining visual and code critiques. We find that:

- Vision-guided critique is essential for frontend code generation and provides a scalable approach to improving output quality
- Iterative refinement consistently improves quality in student-teacher and self-improvement settings
- Partially internalizing improvements is possible through low-rank adaptation

Future directions can include:

- Extending to other visual code generation tasks
- Exploring more token-efficient critique mechanisms
- Understanding diminishing returns from critique feedback

Scan to learn more!

Corresponding author: gvdburg@amazon.co.uk