
Visually Grounded Reinforcement Learning for LLM Programmatic Animation Generation in Interactive Execution Environments

Ravidu Suien Rammuni Silva*
Department of Computer Science
Nottingham Trent University
United Kingdom
ravidu.rammunisilva2024@my.ntu.ac.uk

Ahmad Lotfi
Department of Computer Science
Nottingham Trent University
United Kingdom
ahmad.lotfi@ntu.ac.uk

Isibor Kennedy Ihianle
Department of Computer Science
Nottingham Trent University
United Kingdom
isibor.ihianle@ntu.ac.uk

Golnaz Shahtahmassebi
Department of Physics and Mathematics
Nottingham Trent University
United Kingdom
golnaz.shahtahmassebi@ntu.ac.uk

Jordan J. Bird
Department of Computer Science
Nottingham Trent University
United Kingdom
jordan.bird@ntu.ac.uk

Abstract

Generating programmatic animations with libraries such as Manim poses unique challenges for large language models (LLMs), requiring spatial reasoning, temporal sequencing, and adaptation to interactive execution environments. However, how different post-training objectives influence an LLM’s capacity to leverage dynamic feedback during generation remains under-researched. This study introduces ManimTrainer, a framework that combines Supervised Fine-tuning (SFT) with visually grounded Reinforcement Learning via Group Relative Policy Optimisation (GRPO) using a unified code-and-visual reward signal, alongside ManimAgent, an inference pipeline featuring Renderer-in-the-loop (RITL) and documentation-augmented execution feedback. Evaluating 17 open-source sub-30B LLMs across nine combinations of training and inference strategies on ManimBench, this work presents a unified study investigating how training types interact with test-time execution feedback. The results show that SFT and GRPO improve visual quality to a comparable degree under vanilla inference, while GRPO also preserves the policy’s responsiveness to extrinsic error signals during self-correction. Under renderer feedback, GRPO outperforms the base and SFT models by median margins of 2.7 pp and 1.3 pp across the 17 models, and the same trend also holds under RITL-DOC. Additionally, the analysis shows that the correlation between code and visual metrics strengthens with SFT and GRPO but weakens under inference-time execution loops, highlighting the complementary roles of training objectives and interactive environments in programmatic generation.

*Corresponding Author

1 Introduction

Programmatically generated animations show promise as reliable mediators for the efficient, automated generation of educational videos using Large Language Models (LLMs). Leveraging LLMs’ code-generation abilities with simple yet effective animation libraries, such as Manim [1], provides greater accuracy and efficiency than diffusion-based video generation models. However, generating correct Manim code remains difficult, as it requires not only syntactic fluency in Python but also reasoning about three-dimensional coordinate systems, sequencing of animations over time, and knowledge of the Manim API, which is underrepresented in general code-based pre-training datasets [2].

Fine-tuning adapts an LLM to a specific task and is typically evaluated based on its performance in a single forward pass or chat turn. However, in practice, a deployed model operates in an interactive execution environment, receiving feedback from the tools it calls, such as compiler errors or renderer logs, and revising its output accordingly. The fine-tuning strategy that scores highest on its own is not necessarily the one that leaves a model best able to use that later feedback, and the two are rarely measured together. Manim is a particularly useful setting for measuring them together because the renderer returns a clear pass/fail signal and error log with no labelling requirement, and the video that is ultimately judged is produced by, but distinct from, the code the model writes.

Recent advancements in Parameter-efficient Fine-tuning, notably LoRA [3] and the quantised variant QLoRA [4], have enabled the specialisation of sub-30B LLMs on a single consumer-grade GPU. Supervised Fine-tuning (SFT) can establish a lexical foundation for the Manim API, while Group Relative Policy Optimisation (GRPO) [2, 5] can further improve the model through execution-based reward signals without requiring a separate critic model. How each of these contributes to the code quality of the generated Manim code, as well as to the visual quality of the video it renders, has not been systematically examined.

At inference time, iterative self-correction with extrinsic feedback from compilers has been shown to effectively improve the quality of LLM-generated code [6], and Retrieval-Augmented Generation (RAG) decreases API hallucinations by incorporating relevant documentation into the prompt context [7, 8]. Both are forms of adaptation that occur after post-training has finished, using signals the model has never encountered before. Whether a model optimised against an execution-grounded reward, such as a vision-based reward, is also better at consuming such signals in context remains unexplored, except for intrinsic self-correction and compiler feedback in general coding tasks [9, 10].

Assessing this requires metrics that span both the code and the visual domain. Code metrics such as CodeBLEU [11] and CodeBERT similarity [12] focus solely on the generated code, while visual metrics such as SSIM [13] and CLIP-based similarity [14] overlook it. The extent to which these two families agree across post-training and inference strategies is itself an open question, and one that bears on whether an evaluation protocol remains valid as the deployment setting changes.

This study hypothesises that the fine-tuning strategy determines not only how well a model performs on its own, but also how much it gains from extrinsic feedback supplied at inference time. To test this, ManimTrainer, a training pipeline combining SFT and GRPO under a unified code-and-visual reward, and ManimAgent, an inference pipeline featuring Renderer-in-the-loop (RITL) and API documentation-augmented RITL (RITL-DOC), are applied to 17 open-source sub-30B LLMs, producing nine combinations of training and inference strategies. Based on the identified gaps and the formulated hypothesis above, this work makes four key contributions:

1. A controlled study of how the fine-tuning strategy governs a model’s ability to exploit environment feedback at inference time, covering 17 models across nine training and inference types on the full text-to-code-to-video process.
2. A per-model comparison of the interaction with the base model. SFT and GRPO improve visual quality comparably under vanilla inference, but once feedback is available, GRPO-trained models consistently lead in the median.
3. Observations on how model size limits both the training and the inference-time gains. Two limitations related to model size were identified. When models have 4B parameters or fewer, GRPO offers no additional benefits over SFT, and retrieved API documentation is ineffective in this range. This is consistent with context saturation.

4. An analysis of the correlation between code and visual metrics across all strategy combinations shows that inference-time strategies weaken this correlation while improving the resulting visual quality, so a code-only metric is least reliable for the best-performing strategy.

2 Background and Related Work

Post-training for code generation. LLM-based code generation has progressed rapidly from Codex [15] to compact models such as Qwen 2.5 Coder [16] and SeedCoder [17], which rival much larger systems on standard benchmarks [18], although they often underperform on domain-specific libraries with custom APIs [19, 2]. SFT establishes vocabulary and syntactic foundations but does not incorporate execution-based feedback [20], whereas earlier RL approaches, such as PPO [21] and DPO [22], require either a separate critic network or preference pairs. GRPO [5] offers a critic-free alternative suited to non-differentiable execution-based signals, and DeepSeek-R1 demonstrated that it can significantly improve functional correctness [20]. RL has been shown to improve self-correction and the use of compiler feedback in general code generation [10, 9]. However, how SFT and RL interact with renderer feedback in visually grounded settings remains underexplored.

Inference-time enhancements and self-correction. Iterative self-correction approaches such as Self-Debugging [6], Reflexion [23] and Self-Refine [24] allow LLMs to improve code via in-context learning, though intrinsic correction yields only marginal gains [25] and extrinsic validation is significantly more effective [26]. CodeRL [27], InterCode [28] and SWE-Agent [29] demonstrate that more informative error messages lead to faster convergence, and ReLook [30] extends this to the visual domain with a multimodal critic. RAG can additionally address hallucinated APIs by injecting relevant documentation into the prompt context [7], although models can struggle with dense technical APIs and longer contexts [8].

Programmatic animation and its evaluation. While other works generate visuals via HTML or SVG code [31–33], Manim enables complex animations in a few lines of Python. ManimBench [2] provides 417 human-reviewed, paired natural-language descriptions and executable Manim code snippets, and automated pipelines such as Animator [34], TheoremExplainAgent [35] and Code2Video [36] generate educational content with mostly closed-source LLMs, the last of these highlighting the gap between code execution success and the quality of the visual output. On the measurement side, code-level metrics [11, 12] may penalise Manim snippets that reach a similar visual output by a different coding pathway [2], while SSIM [37, 13] and CLIP-based similarity [14] capture visual fidelity that code-level metrics overlook but may reward inefficient code.

An extended discussion on the related work is available in Appendix A.

3 Method

The pipeline has three stages. SFT first provides the key understanding of the task and adds Manim-specific knowledge to the LLM at a parametric level. A unified reward $\mathcal{R}$ is then formulated by fusing a code-based text reward $\mathcal{R}_{\mathcal{T}}$ with a visual reward $\mathcal{R}_{\mathcal{V}}$ and is used to fine-tune the LLM via GRPO, which visually grounds the SFT-trained model. Finally, the inference-time enhancements RITL and RITL-DOC are applied to the resulting models. Figure 1 gives an overview of ManimTrainer; the ManimAgent inference pipeline is summarised in Figure 2.

3.1 Reward Function

Let a training instance be $(x, \hat{y})$, where x is the formulated prompt, which includes the textual description and $\hat{y}$ is the reference Manim code snippet. During the RL training cycle, policy π_{θ} samples completion $c \sim \pi_{\theta}(\cdot|x)$, from which executable Manim code is deterministically extracted using $\mathcal{H}$:

$$y = \mathcal{H}(c), \tag{1}$$

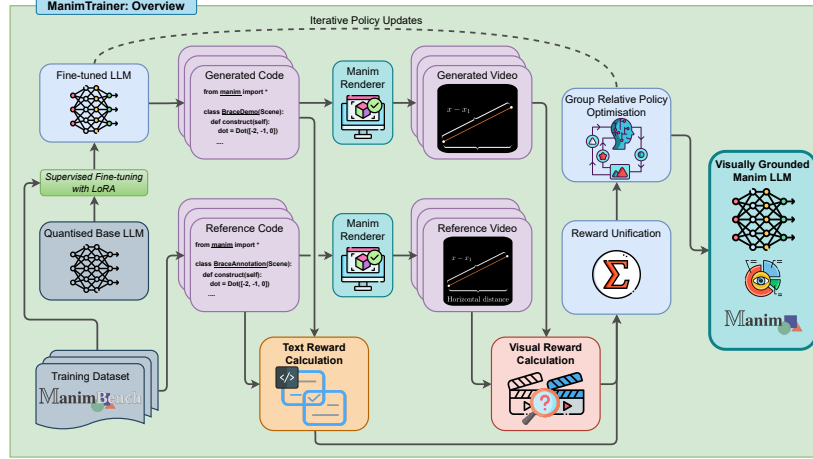


Figure 1: Overview of the ManimTrainer pipeline. The pipeline uses a quantised base LLM and visually grounds it for Manim code generation, using both visual and text reward signals.

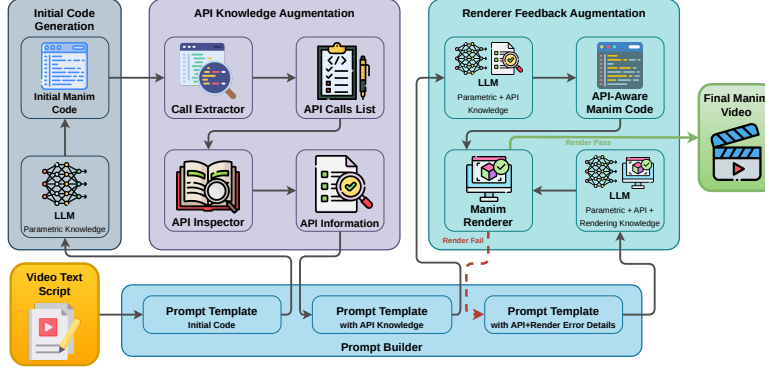


Figure 2: Inference pipeline of the ManimAgent.

where $\mathcal{H}$ is a regex-based function that extracts the generated code snippet y , with $\mathcal{H}(c) = \emptyset$ when no code can be extracted.

The text reward is based on two code-oriented similarity metrics, CodeBLEU (C-BLEU) [11] and CodeBERT similarity (C-BERT) [12], the latter being the cosine similarity of the mean-pooled final hidden layers of the reference and the generated code. The text reward $\mathcal{R}_{\mathcal{T}}$ is their geometric mean:

$$\mathcal{R}_{\mathcal{T}}(y, \hat{y}) = \sqrt{\text{C-BLEU}(y, \hat{y}) \cdot \text{C-BERT}(y, \hat{y})}. \quad (2)$$

For the visual reward, the comparison between the rendered videos v and $\hat{v}$ is performed at each frame, with frames sampled at a fixed rate of 5 frames per second. The sample rate was set empirically, based on many Manim-generated GIFs that run at 5 FPS. From each video pair, a pair of sets of grayscale frames $(G, \hat{G})$ is obtained for the $\mathcal{S}_{ssim}$ calculation and a pair of sets of RGB frames $(I, \hat{I})$ for the $\mathcal{S}_{sem}$ calculation.

The SSIM matrix between all elements of G and $\hat{G}$ is computed as $S' = \text{SSIM}(G, \hat{G})$, with $S' \in \mathbb{R}_{T \times \hat{T}}$, where T and $\hat{T}$ are the frame counts sampled from v and $\hat{v}$ respectively. The matrix is then summarised into two sequences, yielding per-frame best matches for the reference and the generated video, $S'_{ref} = \max_T S'$ and $S'_{gen} = \max_{\hat{T}} S'$, which are aligned using $\text{DTW}_{\text{euclidean}}$, normalised and mapped to $[0, 1]$:

$$\mathcal{S}_{sim}(v, \hat{v}) = \exp\left(-k \cdot \frac{\text{DTW}_{\text{euclidean}}(S'_{gen}, S'_{ref})}{\max(T, \hat{T})}\right), \quad (3)$$

where $k = 5$, making the final metric stricter. The fastDTW [38] library is used for calculation efficiency, and the Euclidean distance is used between elements of the two sequences.

For $\mathcal{S}_{sem}$, each of the two RGB frame sets is encoded with CLIP embeddings and ℓ_2 normalised, giving a semantic encoding for the generated video $\mathcal{E}$ and the reference video $\hat{\mathcal{E}}$:

$$\mathcal{E} = \frac{f_{\text{CLIP}}(I)}{\|f_{\text{CLIP}}(I)\|_2}, \quad \hat{\mathcal{E}} = \frac{f_{\text{CLIP}}(\hat{I})}{\|f_{\text{CLIP}}(\hat{I})\|_2}, \quad (4)$$

where f_{CLIP} is the CLIP encoder from the ViT-L/14 [14] model. DTW is then computed between $\mathcal{E}$ and $\hat{\mathcal{E}}$ under a custom distance $d(e, \hat{e}) = 1 - \langle e, \hat{e} \rangle$ with $e \in \mathcal{E}$ and $\hat{e} \in \hat{\mathcal{E}}$, normalised and mapped to $[0, 1]$:

$$\mathcal{S}_{sem}(v, \hat{v}) = \exp\left(-\frac{\text{DTW}_{d(e, \hat{e})}(\mathcal{E}, \hat{\mathcal{E}})}{\max(T, \hat{T})}\right). \quad (5)$$

The regex function used by $\mathcal{H}$ in Equation 1 attempts to find the code wrapped between `<CODE>` and `</CODE>` tags and/or ````python` markdown-based fences. The training prompt was modelled on the ManimBench study and employed the ‘‘System’’ and ‘‘User’’ roles, as supported by the model’s chat template. GRPOTrainer from TRL (licensed under Apache 2.0) [39] was used for the optimisation in the code implementation. The final visual reward is then calculated as a geometric mean:

$$\mathcal{R}_V(v, \hat{v}) = \sqrt{\mathcal{S}_{sim}(v, \hat{v}) \cdot \mathcal{S}_{sem}(v, \hat{v})}. \quad (6)$$

Unifying the two, and substituting Equation 1, gives

$$\mathcal{R}(x, c, \hat{y}) = \lambda_{\mathcal{T}} \mathcal{R}_{\mathcal{T}}[\mathcal{H}(c), \hat{y}] + \lambda_V \mathcal{R}_V[\mathcal{M}(\mathcal{H}(c)), \mathcal{M}(\hat{y})], \quad (7)$$

where $\mathcal{M}(\cdot)$ denotes the Manim renderer, $\lambda_{\mathcal{T}} = 0.2$, and $\lambda_V = 0.8$. The goal of the RL training cycle is to ground the model’s knowledge in the visual result, but that result is produced only when the completion contains valid Manim code, which yields a high-variance reward signal. A portion of the text similarity had to be introduced to mitigate this, since $\mathcal{R}_{\mathcal{T}}$ produces a signal for both valid and invalid Manim code. The weights were set empirically over $\lambda_{\mathcal{T}} \in \{0.0, 0.2, 0.5\}$, and the (0.2, 0.8) pair yielded the most stable results.

3.2 GRPO-based Optimisation

Using $\mathcal{R}(x, c, \hat{y})$, the loss follows Dr. GRPO [40], $\mathcal{L}(\theta) = -\frac{1}{LG} \sum_{i=1}^G \sum_{t=1}^{|c_i|} l_{i,t}$, where

$$l_{i,t} = \min \left[r_{i,t}(\theta) \hat{A}_{i,t}, \text{clip}(r_{i,t}(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_{i,t} \right] - \beta \mathbb{D}_{\text{KL}}[\pi_{\theta} \| \pi_{\text{ref}}], \quad (8)$$

with $r_{i,t}(\theta) = \pi_{\theta}(c_{i,t} \mid x, c_{i,<t}) / \pi_{\theta_{\text{old}}}(c_{i,t} \mid x, c_{i,<t})$ and the group-relative advantage

$$\hat{A}_{i,t} = \frac{\mathcal{R}(x_i, c_i, \hat{y}_i) - \text{mean}[\mathcal{R}(x, c, \hat{y})]}{\text{std}[\mathcal{R}(x, c, \hat{y})]}. \quad (9)$$

The mean and standard deviation in Equation 9 are computed over $G = 8$ completions sampled for each x_i , with $L \approx 2048 \times 0.8$, $\beta = 0.005$ and $\epsilon = 0.2$. G was selected based on the available resources and the stability of the training cycle; L was selected after analysing the expected token lengths of completed examples, and β and ϵ follow the recommended values [40]. GRPO is performed on top of the SFT model to mitigate reward hacking.

3.3 Inference-time Enhancements

Let d represent the textual description of the animation, $\mathcal{G}$ the generating LLM, and $\text{ManimRender}(\cdot)$ the ManimCE 0.19.0 (licensed under the MIT License) rendering engine that returns a status $s \in \{\text{success}, \text{fail}\}$ together with a renderer error output e . Given initial code $c_1 \leftarrow \mathcal{G}(d)$, RITL iteratively loops for a maximum of K rounds, transforming (d, c_k, e_k) into improved code c_{k+1} and re-rendering it. As shown in Algorithm 1 of Appendix B, the loop terminates either upon a successful render or when reaching $k = K$. To reduce noise in the formulated prompt, the last ten lines of the render error log are used, which are sufficient to highlight the actual issue in the code.

RITL-DOC follows a functionally similar approach, but additionally incorporates API documentation $\mathcal{K}_k$ retrieved via the extracted API call set $\mathcal{A}_k$. The API calls were extracted from the function header comments in Manim’s Python library files, retaining only the parameter details and omitting the examples usually found in those comments, so the prompt does not become excessively large. RITL-DOC thus bypasses the need for an LLM-based or vector-based retriever by extracting API information directly from the initial code. Both place the model in an interactive execution environment at inference time, where the renderer returns its signal in context rather than as a training reward.

4 Experimental Setup

The ManimBench [2] dataset was used, as it includes human-written Manim code snippets alongside natural-language descriptions of the resulting videos and covers all Manim APIs via code examples. Of its 417 code samples, the official test set of 100 was kept for evaluation, and the remainder was used for training. Seventeen models of 30B parameters or fewer were selected from the Unsloth (licensed under Apache 2.0) repository [41] across the Qwen 3 [42], Qwen 2.5 [16], SeedCoder [17], LLaMA 3.1 and 3.2 [43], Mistral 3.2 and Ministral 3 [44] families, based on their BigCodeBench [18] performance and availability there. Qwen 3 Coder Next 80B and GPT-4.1 are included as untrained reference points.

LoRA adapters used rank $r = 8$, with scaling $\alpha = 2r$, targeting the Query, Key, Value, and Output Projection modules, trained at 16-bit precision with the 8-bit AdamW optimiser, while the base model remained frozen at 4-bit Unsloth quantisation. Both training cycles used a batch size of 16. SFT ran for 2 epochs at a learning rate of $2e-6$ with a linear schedule, a weight decay of 0.01 and a warmup ratio of 0.1; GRPO ran on top of the SFT model at a reduced learning rate of $5e-7$. All experiments were conducted on an Ubuntu 24.04 LTS server with 64GB of RAM, an Nvidia RTX 5090 with 32GB of VRAM and 512GB of SSD storage, using ManimCE 0.19.0 and Python 3.12.12.²

Results are reported with three composite metrics. Visual Similarity (VS) and CodeBERTBLEU (CBB) are computed identically to $\mathcal{R}_V$ and $\mathcal{R}_T$ in Section 3.1, and Render Success Rate (RSR) is the fraction of generated code that successfully renders. The pairwise agreement of the seven underlying metrics, which also include N-gram Match, Syntax Match and AST Distance, is examined in Appendix E. Unless stated otherwise, RITL and RITL-DOC use a single correction round, which keeps the emphasis on generating Manim code rather than on repairing faulty code.

5 Results and Discussion

5.1 Overall effect of SFT and GRPO under vanilla inference

The study evaluated 17 open-source sub-30B models using three key metrics that collectively capture their ability to generate Manim code. Table 1 shows that SFT and GRPO consistently improved LLM performance in terms of RSR and VS, demonstrating the benefit of both fine-tuning strategies. Across the 17 models, SFT increased VS by a median of 3.3 pp and GRPO by 2.5 pp, while the mean RSR rose from 38.2% to 41.2% and 40.4%. However, CBB remained relatively similar across all model families and sizes. This behaviour suggests that, with the given dataset, the models learned to fix failing code rather than write new code under both SFT and GRPO.

Paired by model, the VS gain from SFT exceeds the CBB gain by a median of 1.8 pp. As a result, the code metric captures less than half of the improvement shown by the visual metric. Figure 4 in

²GitHub Repository: <https://github.com/SuiensS/manim-trainer>

Table 1: Mean Visual Similarity (VS), CodeBERTBLEU (CBB) and Render Success Rate (RSR) over the 17 fine-tuned models, for every combination of post-training and inference strategy. All values are percentages. Per-model results are given in Appendix C.

Inference	Visual Similarity			CodeBERTBLEU			Render Success Rate		
	Base	SFT	GRPO	Base	SFT	GRPO	Base	SFT	GRPO
Vanilla	34.3	37.6	36.8	52.1	53.4	53.2	38.2	41.2	40.4
RITL	41.7	42.8	44.2	52.1	54.2	54.1	46.6	47.3	48.8
RITL-DOC	43.4	45.0	45.5	53.4	54.3	54.2	48.6	49.6	50.2

Appendix E shows the same separation. The SFT stage moves mainly along the visual axis, while the GRPO stage moves models in both directions. During vanilla inference, 7 models performed best with GRPO, and 9 did so at the SFT stage. Further statistical tests are presented in Appendix D.

5.2 Effect of post-training on the value of renderer feedback

At each training stage, the importance of renderer execution feedback was examined using RITL and RITL-DOC. RITL consistently improved both VS and RSR across most models, outperforming vanilla inference in 47 of the 51 model-and-strategy combinations by a wide margin. Adding retrieved API documentation on top of RITL produced a slight positive shift of 0.7 to 1.9 pp. The mean values in Table 1 suggest a monotonic progression from vanilla inference through RITL to RITL-DOC under all three training strategies. Overall, inference-level strategies deliver greater performance gains than fine-tuning alone, and RITL-DOC with GRPO emerges as the most effective combination throughout the table.

More notably, the ordering of the two fine-tuning strategies changes once feedback is available. Figure 3 reveals a pattern where, overall, SFT slightly outperforms GRPO under vanilla inference, but GRPO improves and surpasses SFT under RITL and RITL-DOC.

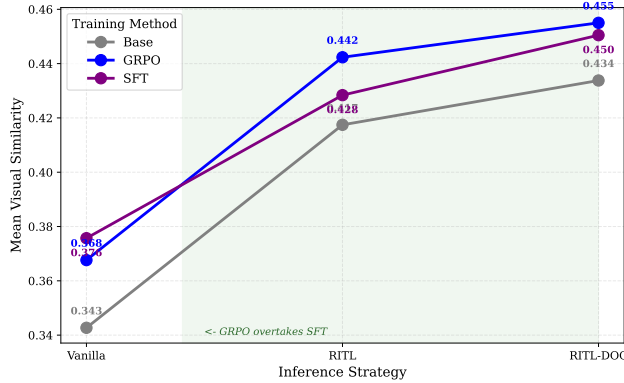


Figure 3: Variation of mean Visual Similarity of the fine-tuned models across the inference strategies. The SFT and GRPO lines cross once a feedback loop is introduced.

This pattern suggests that GRPO-trained models are more receptive to iterative self-correction, consistent with GRPO training, in which the model is trained to satisfy an external validation that closely matches how the feedback loop works in RITL. Table 2 presents differences across inference and training types. Overall, RITL and RITL-DOC have a stronger effect on base and GRPO-trained models, but the effect is less pronounced for SFT models relative to their respective baselines, as Figures 7 and 8 of Appendix E show from either side.

5.3 Limits on fine-tuning and adaptation from parameter count and context capacity

The findings in Tables 1 and 4 indicate that VS improved consistently with SFT, GRPO, or both, across all model sizes. However, the percentage improvement declined as parameter count increased, and CBB remained essentially unchanged across size bands, while VS and RSR expanded, as Figures

Table 2: Median paired pp difference in Visual Similarity between post-training strategies, within each inference type, and the interaction against vanilla inference. Each interaction entry is the median, across models, of the per-model change in that difference. Further statistics are given in Appendix D.

Contrast	Within inference type			Interaction vs. vanilla	
	Vanilla	RITL	RITL-DOC	RITL	RITL-DOC
SFT – Base	+3.3	+1.5	+2.9	−1.7 ($p = 0.021$)	−1.1 ($p = 0.28$)
GRPO – Base	+2.5	+2.7	+2.8	−0.1 ($p = 0.90$)	−0.9 ($p = 0.83$)
GRPO – SFT	−0.8	+1.3	+0.8	+2.1 ($p = 0.028$)	+0.9 ($p = 0.27$)

5 and 6 in Appendix E illustrate. For example, the Qwen 2.5 Coder 7B, Ministral 3 8B, and Qwen 3 8B models demonstrated notable improvements with SFT or GRPO, respectively, increasing their Base VS from 42.0, 34.5, and 29.7 to 50.8, 40.8, and 36.4. However, performance gains for models larger than 14B were comparatively smaller. This is consistent with scaling arguments, such as those in [45], which note that larger models require proportionally larger training datasets to improve. Across families, the 7B-8B range emerges as an efficiency sweet spot. SeedCoder 8B, with only 8B parameters, delivers the best vanilla performance among all 17, with 64.8% VS and 72% RSR under GRPO, surpassing even the newer Qwen 3 Coder Next 80B MoE model.

At the other end of the range, these trends suggest a capacity threshold for the RL training cycle. For the seven models with 4B parameters or fewer, SFT leads GRPO under vanilla inference by a median of 3.1 pp. GRPO appears to benefit models that already possess sufficient task competence after SFT, whereas smaller models ($\leq 4B$) may struggle to generate informative samples during RL optimisation.

Retrieved API documentation stops benefiting in the same size range. Moving from RITL to RITL-DOC, the ten models at 7B and above gain a median of 3.4 pp of VS, while the seven models at 4B and below lose a median of 1.2 pp. A likely explanation is context saturation, since RITL-DOC can consume most of the available context length, with dense API information overwhelming the smaller models. Retrieved documentation is therefore evidently useful when the model has enough context capacity to absorb it, which is roughly the same capacity that GRPO appears to need.

The best configuration reached in these experiments is Qwen 3 Coder 30B with GRPO under RITL-DOC, which achieved 83.0% VS and 92% RSR after a single correction round, using a model with only 3B active parameters out of 30B total. With three correction attempts, it reached 85.7% VS and 94% RSR, compared with 82.8% VS and 94% RSR for GPT-4.1 under the same budget, as shown in Figure 9 of Appendix E. The additional attempts consistently improved VS and RSR across all three training strategies, with larger gains under GRPO. However, for the Base and SFT variants, the additional correction loops tended to lower CodeBERTBLEU, suggesting that they encourage the model to apply fixes to the initial code, which may not be the ideal way to generate the specific code for the given prompt.

5.4 Agreement between code and visual metrics across regimes

Investigating the relationship between VS and CBB, Table 3 presents the rank correlations between the two composite metrics for every combination of training and inference strategies. The strongest correlation appears with GRPO under vanilla inference, with Spearman’s ρ at 0.919 and Kendall’s τ at 0.809. Under vanilla inference, agreement strengthens monotonically along the training pipeline, from 0.689 at Base to 0.833 after SFT and 0.919 after GRPO, suggesting that grounding the policy in execution brings the code the model writes and the video it renders closer together. However, a notable trend emerges when inference-time strategies are introduced, resulting in consistently lower correlation scores, with RITL-DOC dropping to 0.694 for SFT and 0.701 for GRPO. The same pattern appears when grouping by inference strategy, where vanilla inference yields the highest correlation regardless of training strategy.

These results highlight a notable practical phenomenon: while the generated code aligns better with visual quality under SFT and GRPO, the inference-level enhancements prevent that correlation from holding. Instead, they produce code that differs from the expected but still renders videos close to the anticipated output, which can be seen as the models fixing broken code with innovative but

Table 3: Rank correlations between Visual Similarity and CodeBERTBLEU for every combination of post-training and inference strategy, alongside the grouped analyses

Grouping	Category	Spearman’s ρ	Kendall’s τ	n
Post-training $\times$ Inference	Base	0.689	0.559	17
	SFT	0.833	0.676	17
	GRPO	0.919	0.809	17
	Base+RITL	0.696	0.544	17
	SFT+RITL	0.799	0.603	17
	GRPO+RITL	0.713	0.559	17
	Base+RITL-DOC	0.743	0.559	17
	SFT+RITL-DOC	0.694	0.544	17
	GRPO+RITL-DOC	0.701	0.574	17
Post-training strategy	Base	0.703	0.527	51
	SFT	0.791	0.605	51
	GRPO	0.772	0.619	51
Inference strategy	Vanilla	0.817	0.648	51
	RITL	0.729	0.553	51
	RITL-DOC	0.706	0.534	51

suboptimal methods. Figure 10 in Appendix E also shows weak, negative correlations (as expected for a distance) between the AST distance and other metrics, suggesting that it is unreliable for the Manim code generation evaluation. One plausible reason could be Manim’s object-oriented scene structure, which remains relatively consistent across all Manim scripts. A qualitative analysis of the models is presented in Appendix F.

6 Limitations and Broader Impact

Internal validity: Hyperparameter selection was conducted empirically and kept consistent across all models, and tuning tailored to individual models may produce different outcomes. All models used LoRA adapters over 4-bit quantised bases, which could introduce quantisation effects varying across models and families. Each configuration was decoded once, so the per-model scores carry no run-to-run error bars, and the claims from Section 5.2 onward are consequently stated as paired tests across the 17 models rather than as differences between individual runs.

External validity: The experiments use ManimBench (licensed under CC-BY-NC-SA 4.0). While it covers the entire Manim API, it does not include samples that generate longer, script-based videos, so the conclusions pertain to the fundamental capabilities of LLMs for Manim code generation. The findings are specific to ManimCE 0.19.0, and full-precision models might achieve better performance at the cost of increased compute.

Construct validity: The scores are based on a reference code that is expert-written and drawn from Manim’s official documentation, which may penalise generated videos that are high-quality and visually relevant to the description but differ from the reference. The error logs used in RITL were truncated to the last ten lines to reduce noise in the model context, which may occasionally cut relevant diagnostic information.

Broader impact: The pipeline lowers the cost of producing programmatic animations by using compact open models on a single consumer GPU rather than large proprietary systems. The same property makes it easy to mass-produce plausible-looking explanatory videos whose content is incorrect, since a video that renders successfully and resembles a reference is not necessarily accurate; therefore, any deployment that presents generated animations to parties such as students should keep a human in the loop for factual review. No personal or human-subject data was used in this study.

7 Conclusion and Future Work

This paper presents ManimTrainer and ManimAgent, which together offer an end-to-end framework for training and deploying sub-30B LLMs to generate Manim programmatic animations. It offers a comprehensive analysis of the interaction between training strategies and inference strategies for this task, using 17 open-source LLMs ranging from 0.5B to 30B parameters. It further examines how the choice of fine-tuning strategy affects the model’s ability to use extrinsic feedback provided

at inference time. The results showed that both SFT and GRPO fine-tuning consistently improved the LLMs’ Manim code generation performance. However, CBB remained relatively unchanged. LLMs with around 8B parameters demonstrated well-balanced performance in terms of accuracy and efficiency, with the SeedCoder 8B model, trained with GRPO, outperforming the much larger Qwen 3 Coder Next 80B base model.

Inference-time enhancement strategies achieved even greater performance gains than mere fine-tuning. Notably, GRPO models led SFT and base models in the median under both RITL and RITL-DOC with render feedback, a consistent direction that does not reach significance at this sample size. The top overall results of 85.7% VS and 94% RSR were achieved by the Qwen 3 Coder 30B model with GRPO under RITL-DOC, utilising three RITL loops, surpassing the baseline GPT-4.1 model by 2.9 pp in VS at an equal RSR. Further analysis indicated that under vanilla inference, the code-visual metric correlation increased as the training pipeline progressed from base to SFT, then from SFT to GRPO, but consistently decreased when inference-level enhancements were introduced.

As future work, this work can be expanded in several directions. First, integrating a multimodal critic that provides natural-language feedback on rendered videos could further address visual defects and inconsistencies. The rule-based API retrieval in RITL-DOC can also be improved by using a learned retriever model, which would benefit the smaller models that currently struggle with longer, noisier contexts. The same pipeline could then be evaluated across successive Manim releases, using multiple seeds, to test whether the quantities examined here remain stable as the environment evolves. Finally, the ManimTrainer and ManimAgent frameworks could be adapted to other programmatic animation-generation domains with minimal development effort, such as HTML/CSS, SVG-based animation, and TikZ and XML-based diagram generation.

Acknowledgments and Disclosure of Funding

This work was supported solely by the primary academic institutions of the authors. No external funding or commercial relationships were involved. The authors declare no competing interests.

References

- [1] The Manim Community Developers. Manim – Mathematical Animation Framework, 2026. URL <https://www.manim.community/>.
- [2] Ravidu Suien Rammuni Silva, Ahmad Lotfi, Isibor Kennedy Ihianle, Golnaz Shahtahmassebi, and Jordan J. Bird. Large Language Model Approaches to Educational Video Generation Using Manim. In Emma Hart, Tomas Horvath, Zhiyuan Tan, and Sarah Thomson, editors, *Advances in Computational Intelligence Systems*, volume 1468, pages 306–317. Springer Nature Switzerland, Cham, 2026. ISBN 978-3-032-07937-4 978-3-032-07938-1. doi: 10.1007/978-3-032-07938-1_26. Series Title: Advances in Intelligent Systems and Computing.
- [3] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models, 2021. URL <https://arxiv.org/abs/2106.09685>.
- [4] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: efficient finetuning of quantized llms. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS ’23, Red Hook, NY, USA, 2023. Curran Associates Inc.
- [5] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models, 2024. URL <https://arxiv.org/abs/2402.03300>.
- [6] Xinyun Chen, Maxwell Lin, Nathanael Schaerli, and Denny Zhou. Teaching Large Language Models to Self-Debug. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun, editors, *International Conference on Learning Representations*, volume 2024, pages 8746–8825, 2024.

- [7] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems, NIPS '20*, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 978-1-7138-2954-6.
- [8] Zora Zhiruo Wang, Akari Asai, Xinyan Velocity Yu, Frank F. Xu, Yiqing Xie, Graham Neubig, and Daniel Fried. CodeRAG-Bench: Can Retrieval Augment Code Generation?, 2024. URL <https://arxiv.org/abs/2406.14497>.
- [9] Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, JD Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal Behbahani, and Aleksandra Faust. Training Language Models to Self-Correct via Reinforcement Learning. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu, editors, *International Conference on Learning Representations*, volume 2025, pages 54523–54549, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/871ac99fdc5282d0301934d23945ebaa-Paper-Conference.pdf.
- [10] Shihan Dou, Yan Liu, Haoxiang Jia, Enyu Zhou, Limao Xiong, Junjie Shan, Caishuang Huang, Xiao Wang, Xiaoran Fan, Zhiheng Xi, Yuhao Zhou, Tao Ji, Rui Zheng, Qi Zhang, Tao Gui, and Xuanjing Huang. StepCoder: Improving Code Generation with Reinforcement Learning from Compiler Feedback. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4571–4585, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.251. URL <https://aclanthology.org/2024.acl-long.251/>.
- [11] Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. CodeBLEU: a Method for Automatic Evaluation of Code Synthesis, 2020. URL <https://arxiv.org/abs/2009.10297>.
- [12] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. CodeBERT: A Pre-Trained Model for Programming and Natural Languages, 2020. URL <https://arxiv.org/abs/2002.08155>.
- [13] Peter Ndajah, Hisakazu Kikuchi, Masahiro Yukawa, Hidenori Watanabe, and Shogo Muramatsu. SSIM image quality metric for denoised images. In *Proceedings of the 3rd WSEAS international conference on Visualization, imaging and simulation*, pages 53–57, 2010.
- [14] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning Transferable Visual Models From Natural Language Supervision, 2021. URL <https://arxiv.org/abs/2103.00020>.
- [15] Mark Chen, Tworek, et al. Evaluating Large Language Models Trained on Code, 2021. URL <https://arxiv.org/abs/2107.03374>.
- [16] Binyuan Hui, Jian Yang, Zeyu Cui, Jiayi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, Yunlong Feng, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. Qwen2.5-Coder Technical Report, 2024. URL <https://arxiv.org/abs/2409.12186>.
- [17] ByteDance Seed, Yuyu Zhang, Jing Su, Yifan Sun, Chenguang Xi, Xia Xiao, Shen Zheng, Anxiang Zhang, Kaibo Liu, Daoguang Zan, Tao Sun, Jinhua Zhu, Shulin Xin, Dong Huang, Yetao Bai, Lixin Dong, Chao Li, Jianchong Chen, Hanzhi Zhou, Yifan Huang, Guanghan Ning, Xierui Song, Jiaze Chen, Siyao Liu, Kai Shen, Liang Xiang, and Yonghui Wu. Seed-Coder: Let the Code Model Curate Data for Itself, 2025. URL <https://arxiv.org/abs/2506.03524>.
- [18] Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widayarsi, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen Gong, Thong Hoang, Armel Randy Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan

- Zhang, Prateek Yadav, Naman Jain, Alex Gu, Zhoujun Cheng, Jiawei Liu, Qian Liu, Zijian Wang, Binyuan Hui, Niklas Muennighoff, David Lo, Daniel Fried, Xiaoning Du, Harm de Vries, and Leandro Von Werra. BigCodeBench: Benchmarking Code Generation with Diverse Function Calls and Complex Instructions, 2024. URL <https://arxiv.org/abs/2406.15877>.
- [19] Qiushi Sun, Jingyang Gong, Yang Liu, Qiaosheng Chen, Lei Li, Kai Chen, Qipeng Guo, Ben Kao, and Fei Yuan. JanusCoder: Towards a Foundational Visual-Programmatic Interface for Code Intelligence, 2025. URL <https://arxiv.org/abs/2510.23538>.
 - [20] Daya Guo, Dejian Yang, and Zhang and others. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, September 2025. ISSN 0028-0836, 1476-4687. doi: 10.1038/s41586-025-09422-z. URL <https://www.nature.com/articles/s41586-025-09422-z>.
 - [21] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744, 2022.
 - [22] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 53728–53741, 2023.
 - [23] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 8634–8652, 2023.
 - [24] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegrefe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative Refinement with Self-Feedback. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 46534–46594, 2023.
 - [25] Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. When Can LLMs *Actually* Correct Their Own Mistakes? A Critical Survey of Self-Correction of LLMs. *Transactions of the Association for Computational Linguistics*, 12:1417–1440, November 2024. ISSN 2307-387X. doi: 10.1162/tac1_a_00713. URL https://direct.mit.edu/tac1/article/doi/10.1162/tac1_a_00713/125177/When-Can-LLMs-Actually-Correct-Their-Own-Mistakes.
 - [26] Guiyao Tie, Zenghui Yuan, Zeli Zhao, Chaoran Hu, Tianhe Gu, Ruihang Zhang, Sizhe Zhang, Junran Wu, Xiaoyue Tu, Ming Jin, Qingsong Wen, Lixing Chen, Pan Zhou, and Lichao Sun. Can LLMs Correct Themselves? A Benchmark of Self-Correction in LLMs, 2025. URL <https://arxiv.org/abs/2510.16062>.
 - [27] Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven C.H. Hoi. CodeRL: mastering code generation through pretrained models and deep reinforcement learning. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS ’22, Red Hook, NY, USA, 2022. ISBN 978-1-7138-7108-8.
 - [28] John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. InterCode: standardizing and benchmarking interactive coding with execution feedback. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS ’23, Red Hook, NY, USA, 2023. Curran Associates Inc.
 - [29] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: agent-computer interfaces enable automated software engineering. In *Proceedings of the 38th International Conference on Neural Information*

- Processing Systems*, NIPS '24, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 979-8-3313-1438-5.
- [30] Yuhang Li, Chencheng Zhang, Ruilin Lv, Ao Liu, Ken Deng, Yuanxing Zhang, Jiaheng Liu, Wiggan Zhou, and Bo Zhou. ReLook: Vision-Grounded RL with a Multimodal LLM Critic for Agentic Web Coding, 2025. URL <https://arxiv.org/abs/2510.11498>.
 - [31] Tiffany Tseng, Ruijia Cheng, and Jeffrey Nichols. Keyframer: Empowering Animation Design using Large Language Models, 2024. URL <https://arxiv.org/abs/2402.06071>.
 - [32] Xuanle Zhao, Deyang Jiang, Zhixiong Zeng, Lei Chen, Haibo Qiu, Jing Huang, Yufeng Zhong, Liming Zheng, Yilin Cao, and Lin Ma. VinciCoder: Unifying Multimodal Code Generation via Coarse-to-fine Visual Reinforcement Learning, 2025. URL <https://arxiv.org/abs/2511.00391>.
 - [33] Vivian Liu, Rubaiat Habib Kazi, Li-Yi Wei, Matthew Fisher, Timothy Langlois, Seth Walker, and Lydia Chilton. LogoMotion: Visually-Grounded Code Synthesis for Creating and Editing Animation. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, CHI '25, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 979-8-4007-1394-1. doi: 10.1145/3706598.3714155. URL <https://doi.org/10.1145/3706598.3714155>.
 - [34] Samarth P, Vyoman Jain, Shiva Golugula, and Motamarri Sai Sathvik. Manimator: Transforming Research Papers into Visual Explanations, 2025. URL <https://arxiv.org/abs/2507.14306>.
 - [35] Max Ku, Cheuk Hei Chong, Jonathan Leung, Krish Shah, Alvin Yu, and Wenhui Chen. TheoremExplainAgent: Towards video-based multimodal explanations for llm theorem understanding. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6663–6684, 2025.
 - [36] Yanzhe Chen, Kevin Qinghong Lin, and Mike Zheng Shou. Code2Video: A Code-centric Paradigm for Educational Video Generation, 2025. URL <https://arxiv.org/abs/2510.01174>.
 - [37] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004. doi: 10.1109/TIP.2003.819861.
 - [38] Stan Salvador and Philip Chan. Toward accurate dynamic time warping in linear time and space. *Intell. Data Anal.*, 11(5):561–580, October 2007. ISSN 1088-467X.
 - [39] Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. TRL: Transformers Reinforcement Learning, 2020. URL <https://github.com/huggingface/trl>.
 - [40] Zichen Liu, Changyu Chen, and Li and others. Understanding R1-Zero-Like Training: A Critical Perspective, 2025. URL <https://arxiv.org/abs/2503.20783>.
 - [41] Michael Han Daniel Han and Unsloth team. Unsloth, 2023. URL <https://github.com/unslothai/unsloth>.
 - [42] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 Technical Report, 2025. URL <https://arxiv.org/abs/2505.09388>.
 - [43] Aaron Grattafiori and Dubey and others. The Llama 3 Herd of Models, 2024. URL <https://arxiv.org/abs/2407.21783>.

- [44] Alexander H. Liu and Khandelwal and others. Ministral 3, 2026. URL <https://arxiv.org/abs/2601.08584>.
- [45] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack W. Rae, and Laurent Sifre. Training compute-optimal large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

A Extended Related Work

SFT and RL for LLM-based code generation. LLM-based code generation has progressed rapidly from Codex [15] to recent compact models such as Qwen 2.5 Coder [16] and SeedCoder [17], which rival much larger systems on standard benchmarks [18]. However, these models often underperform on domain-specific libraries with custom APIs [19, 2]. While SFT establishes vocabulary and syntactic foundations for code generation, it does not incorporate execution-based feedback [20]. Reinforcement learning addresses this gap, but earlier approaches such as PPO [21] and DPO [22] require a separate critic network or preference pairs. Group Relative Policy Optimisation [5] addresses this by offering a critic-free alternative suited to code generation, where outputs are evaluated using non-differentiable execution-based signals. While DeepSeek-R1 demonstrated that GRPO can significantly improve functional correctness [20], the interaction between SFT- and GRPO-trained models with inference-time enhancement strategies remains underexplored.

Inference-time enhancements and self-correction. Iterative self-correction approaches such as Self-Debugging [6], Reflexion [23], and Self-Refine [24] allow LLMs to improve code via in-context learning, though systematic evaluations show intrinsic correction yields only marginal gains [25] and extrinsic validation is significantly more effective [26]. Extrinsic feedback from compilers, interpreters and renderers can significantly improve the self-correction capabilities of LLMs. CodeRL [27], InterCode [28], and SWE-Agent [29] demonstrate that more informative error messages lead to faster convergence [6]. ReLook [30] extends this to the visual domain, using a multimodal LLM-based critic to detect defects that execution logs cannot surface. Supplementing self-correction loops, Retrieval-Augmented Generation can address text-based errors, such as hallucinated APIs and their parameters, by injecting relevant documentation into the prompt context [7]. While RAG consistently reduces hallucinations, models can struggle with dense technical APIs and longer contexts [8]. The interaction between API information augmentation, iterative execution, and renderer-based refinement loops, however, remains largely underexplored.

Programmatic animation generation via LLMs. When combined with the code-generation capabilities of LLMs, libraries such as Manim [1] offer a more efficient and precise alternative for generating animations. While other works generate visuals via HTML or SVG code [31–33], Manim enables complex animations in a few lines of Python. However, it requires spatial reasoning about 3D coordinates, temporal sequencing of animations, and knowledge of a specialised API that is not well represented in general pre-training corpora. ManimBench [2] addresses the need for a domain-specific evaluation benchmark for Manim by providing a dataset of 417 human-reviewed, paired natural-language descriptions and executable Manim code snippets. Automated pipelines such as Animator [34] and TheoremExplainAgent [35] employ multi-stage processes to generate code for visualising educational and academic content, primarily using commercial, API-based, closed-source LLMs. Code2Video [36] further explores this multi-stage process, highlighting the gap between code execution success and the quality of the visual outputs.

Evaluating LLM-generated code and its visual output. Code-level metrics such as CodeBLEU [11] and CodeBERT similarity [12] evaluate the structure of generated code, but may penalise Manim snippets that produce similar visual outputs via different coding pathways [2]. Visual metrics such as the Structural Similarity Index [37, 13] and CLIP-based [14] semantic similarity can evaluate the quality of rendered outputs against reference frames, capturing visual fidelity that code-level metrics overlook. Conversely, relying solely on visual metrics may reward inefficient code. This necessitates combining both metric families and extending evaluation to the temporal dimension of video.

B Renderer-in-the-loop Procedure

Algorithm 1 states the inference-time loop of Section 3.3. RITL and RITL-DOC differ only in whether retrieved API documentation is added to the correction prompt.

Algorithm 1 Renderer-in-the-loop generation, with optional retrieved API documentation

Require: Textual description d , maximum RITL rounds K , generating LLM $\mathcal{G}$, Manim API knowledge base $\mathcal{D}$

Ensure: Generated Manim code c , render status s

```

1:  $c_1 \leftarrow \mathcal{G}(d)$  ▷ Initial code generation
2:  $s_1, e_1 \leftarrow \text{ManimRender}(c_1)$  ▷ Render and capture status/errors
3: for  $k = 1$  to  $K$  do
4:   if  $s_k = \text{success}$  then
5:     break
6:   end if
7:   if RITL-DOC then
8:      $\mathcal{A}_k \leftarrow \text{ExtractAPICalls}(c_k)$  ▷ Identify Manim API calls in code
9:      $\mathcal{K}_k \leftarrow \text{RetrieveDocs}(\mathcal{A}_k, \mathcal{D})$  ▷ Relevant API documentation
10:     $p \leftarrow \text{BuildPrompt}_{\text{RITL-DOC}}(d, c_k, e_k, \mathcal{K}_k)$ 
11:   else
12:     $p \leftarrow \text{BuildPrompt}_{\text{RITL}}(d, c_k, e_k)$  ▷ Prompt with renderer output
13:   end if
14:    $c_{k+1} \leftarrow \mathcal{G}(p)$  ▷ Regenerate code
15:    $s_{k+1}, e_{k+1} \leftarrow \text{ManimRender}(c_{k+1})$ 
16: end for
17: return  $c, s$ 

```

C Per-model Results

Tables 4, 5 and 6 provide the complete per-model evaluation under vanilla, RITL, and RITL-DOC inference, respectively. Table 3 presents the full correlation breakdown discussed in Section 5.4.

Table 4: Evaluation summary of all the fine-tuned models under vanilla inferencing. The scores present the percentage visual and code similarity values under the VS and CBB columns, and the percentage of generated samples successfully rendered under Render Success Rate.

Model	Visual Similarity			CodeBERTBLEU			Render Success Rate		
	Base	SFT	GRPO	Base	SFT	GRPO	Base	SFT	GRPO
Qwen 2.5 0.5B	6.3	10.9	11.6	36.4	40.3	41.3	7.0	12.0	13.0
LLaMA 3.2 1B	6.4	11.8	8.7	40.8	44.6	39.9	7.0	13.0	10.0
LLaMA 3.2 3B	12.8	16.1	10.7	52.5	51.0	51.0	14.0	18.0	12.0
Ministral 3 3B	15.9	19.4	20.9	52.9	54.3	53.6	18.0	22.0	23.0
Qwen 2.5 Coder 3B	30.8	35.8	32.4	54.5	54.8	54.4	34.0	39.0	36.0
Qwen 3 4B	26.3	28.4	28.4	54.3	54.0	53.9	29.0	31.0	31.0
Qwen 2.5 Coder 1.5B	23.8	24.1	20.6	50.6	52.5	52.7	27.0	27.0	23.0
Qwen 2.5 Coder 7B	42.0	50.8	48.4	55.2	54.3	55.5	46.0	55.0	53.0
Meta LLaMA 3.1 8B	29.7	30.0	30.1	54.4	54.4	54.3	33.0	33.0	33.0
Ministral 3 8B	34.5	40.8	40.0	51.4	56.8	56.7	40.0	44.0	43.0
Qwen 3 8B	29.7	34.4	36.4	56.4	55.2	54.4	33.0	38.0	40.0
SeedCoder 8B	60.3	62.3	64.8	57.7	58.0	57.8	67.0	69.0	72.0
Ministral 3 14B	39.9	42.7	44.6	47.7	54.4	54.5	45.0	46.0	48.0
Qwen 2.5 Coder 14B	52.0	54.8	51.8	56.9	57.1	57.1	57.0	60.0	57.0
Qwen 3 14B	57.1	58.3	54.6	56.2	55.6	56.6	64.0	65.0	60.0
Mistral Small 3.2 24B	55.9	54.9	59.4	55.2	56.8	56.7	63.0	59.0	64.0
Qwen 3 Coder 30B (A3B)	59.2	63.2	61.7	53.2	54.5	54.6	66.0	70.0	68.0
Qwen 3 Coder Next 80B	63.7	-	-	54.6	-	-	72.0	-	-
OpenAI GPT-4.1	68.6	-	-	55.9	-	-	77.0	-	-

Table 5: Evaluation summary of all the fine-tuned models under RITL inference.

Model (with RITL)	Visual Similarity			CodeBERTBLEU			Render Success Rate		
	Base	SFT	GRPO	Base	SFT	GRPO	Base	SFT	GRPO
Qwen 2.5 0.5B	8.1	11.3	12.4	46.2	45.1	44.5	9.0	13.0	14.0
LLaMA 3.2 1B	9.7	10.8	12.4	49.3	47.6	47.7	11.0	12.0	14.0
LLaMA 3.2 3B	14.3	15.8	17.2	53.2	53.6	53.5	16.0	18.0	19.0
Ministral 3 3B	21.2	19.4	24.4	28.4	53.4	55.2	24.0	22.0	27.0
Qwen 2.5 Coder 3B	34.0	36.8	35.5	54.7	53.9	54.8	38.0	40.0	39.0
Qwen 3 4B	33.5	35.8	29.8	54.6	54.8	53.7	37.0	40.0	33.0
Qwen 2.5 Coder 1.5B	24.0	22.8	25.2	51.4	51.3	52.2	27.0	26.0	29.0
Qwen 2.5 Coder 7B	61.0	58.0	58.8	54.6	55.6	54.9	66.0	63.0	64.0
Meta LLaMA 3.1 8B	34.1	31.3	39.8	54.5	55.0	54.7	39.0	37.0	45.0
Ministral 3 8B	45.9	47.3	48.0	52.0	56.9	56.1	53.0	51.0	52.0
Qwen 3 8B	37.0	40.4	47.3	56.2	55.9	56.6	41.0	44.0	53.0
SeedCoder 8B	66.4	68.4	69.2	57.5	57.9	57.8	74.0	76.0	77.0
Ministral 3 14B	49.9	55.9	57.8	52.2	55.8	55.9	56.0	61.0	63.0
Qwen 2.5 Coder 14B	65.2	69.0	66.1	57.0	57.1	56.8	72.0	77.0	72.0
Qwen 3 14B	67.8	62.7	64.0	56.3	55.8	55.4	76.0	69.0	71.0
Mistral Small 3.2 24B	63.1	60.4	63.4	55.1	57.2	56.4	71.0	65.0	68.0
Qwen 3 Coder 30B (A3B)	74.5	82.2	80.6	53.3	54.8	54.1	83.0	90.0	89.0
Qwen 3 Coder Next 80B	72.6	-	-	53.9	-	-	82.0	-	-
OpenAI GPT-4.1	76.9	-	-	56.2	-	-	86.0	-	-

Table 6: Evaluation summary of all the fine-tuned models under RITL-DOC inference.

Model (RITL-DOC)	Visual Similarity			CodeBERTBLEU			Render Success Rate		
	Base	SFT	GRPO	Base	SFT	GRPO	Base	SFT	GRPO
Qwen 2.5 0.5B	8.8	7.2	8.8	49.5	48.4	50.0	10.0	8.0	10.0
LLaMA 3.2 1B	10.0	9.2	10.0	48.7	47.7	45.8	11.0	10.0	11.0
LLaMA 3.2 3B	14.3	12.6	20.5	51.5	52.3	51.2	16.0	14.0	23.0
Ministral 3 3B	20.7	31.4	30.5	47.6	56.0	55.4	24.0	35.0	34.0
Qwen 2.5 Coder 3B	36.2	40.1	30.6	54.4	54.5	54.7	40.0	44.0	34.0
Qwen 3 4B	25.2	31.9	30.2	55.0	54.3	54.1	28.0	35.0	33.0
Qwen 2.5 Coder 1.5B	23.0	24.7	20.4	51.6	50.8	51.7	26.0	28.0	23.0
Qwen 2.5 Coder 7B	60.6	63.5	64.9	54.8	55.0	55.1	66.0	69.0	71.0
Meta LLaMA 3.1 8B	38.9	40.2	41.2	54.5	54.3	54.3	44.0	45.0	46.0
Ministral 3 8B	50.2	55.4	54.6	52.3	57.2	56.9	58.0	60.0	59.0
Qwen 3 8B	51.4	41.5	47.4	56.5	55.4	55.7	58.0	47.0	52.0
SeedCoder 8B	65.6	69.2	69.2	57.3	57.5	57.4	73.0	77.0	77.0
Ministral 3 14B	56.8	59.7	59.6	52.5	55.9	56.2	64.0	65.0	65.0
Qwen 2.5 Coder 14B	66.0	66.4	67.1	56.6	56.9	56.9	73.0	73.0	74.0
Qwen 3 14B	63.6	68.7	71.0	56.5	55.7	55.9	71.0	76.0	79.0
Mistral Small 3.2 24B	68.4	63.2	64.5	55.3	57.3	56.6	77.0	69.0	70.0
Qwen 3 Coder 30B (A3B)	77.7	80.8	83.0	53.5	54.4	53.7	87.0	89.0	92.0
Qwen 3 Coder 30B (A3B) (@3)	79.2	82.6	85.7	53.1	53.7	54.2	89.0	92.0	94.0
Qwen 3 Coder Next 80B	73.9	-	-	54.1	-	-	85.0	-	-
OpenAI GPT-4.1	81.9	-	-	56.6	-	-	92.0	-	-
OpenAI GPT-4.1 (@3)	82.8	-	-	56.1	-	-	94.0	-	-

D Statistical Tests

Table 8 shows further statistical significance tests conducted on the experiment results.

E Additional Figures

Table 7: Mean Visual Similarity of the 17 fine-tuned models across all combinations of post-training and inference strategies.

	Base	SFT	GRPO	Base RITL	SFT RITL	GRPO RITL	Base RITL-D.	SFT RITL-D.	GRPO RITL-D.
[Mean]	34.3	37.6	36.8	41.7	42.8	44.2	43.4	45.0	45.5

Table 8: Exact two-sided Wilcoxon signed-rank tests, paired across 17 models. n counts the non-zero differences used in the test, the median is over all 17 paired differences in percentage points, and $+/-$ indicates the sign split of the non-zero differences. Holm adjustment is within the family indicated by the block. In the interaction rows, Δ_r denotes the stated difference measured within inference regime r .

Family	Comparison	n	Median	+/-	p	Holm
Training stages, Vanilla	VS: SFT vs. Base	17	+3.3	16/1	< 0.0001	0.0007
	VS: GRPO vs. SFT	16	-0.8	7/9	0.2363	0.6780
	VS: GRPO vs. Base	17	+2.5	13/4	0.0075	0.0526
	CBB: SFT vs. Base	16	+0.3	11/5	0.0575	0.2875
	CBB: GRPO vs. SFT	15	-0.1	6/9	0.7291	0.7291
	CBB: GRPO vs. Base	17	+0.3	11/6	0.1290	0.5159
	RSR: SFT vs. Base	15	+4.0	14/1	0.0013	0.0107
	RSR: GRPO vs. SFT	15	-1.0	6/9	0.2260	0.6780
	RSR: GRPO vs. Base	15	+2.0	12/3	0.0308	0.1846
Renderer Execution feedback (VS)	RITL vs. Vanilla, Base	17	+7.2	17/0	< 0.0001	0.0001
	RITL-DOC vs. Vanilla, Base	17	+6.5	15/2	< 0.0001	0.0005
	RITL-DOC vs. RITL, Base	16	+0.7	10/6	0.1973	0.3971
	RITL vs. Vanilla, SFT	16	+5.5	13/3	0.0013	0.0051
	RITL-DOC vs. Vanilla, SFT	17	+8.3	14/3	0.0007	0.0035
	RITL-DOC vs. RITL, SFT	17	+1.9	11/6	0.1324	0.3971
	RITL vs. Vanilla, GRPO	17	+6.5	17/0	< 0.0001	0.0001
	RITL-DOC vs. Vanilla, GRPO	17	+9.8	14/3	0.0005	0.0032
	RITL-DOC vs. RITL, GRPO	16	+1.1	12/4	0.1631	0.3971
GRPO/SFT ordering (VS)	Vanilla	16	-0.8	7/9	0.2363	0.4726
	RITL	17	+1.3	13/4	0.0957	0.2871
	RITL-DOC	16	+0.8	10/6	0.3289	0.4726
Interactions (VS)	$\Delta_{\text{RITL}} - \Delta_{\text{Vanilla}}: \text{SFT} - \text{Base}$	16	-1.7	4/12	0.0214	0.1284
	$\Delta_{\text{R-DOC}} - \Delta_{\text{Vanilla}}: \text{SFT} - \text{Base}$	17	-1.1	7/10	0.2786	1.0000
	$\Delta_{\text{RITL}} - \Delta_{\text{Vanilla}}: \text{GRPO} - \text{Base}$	17	-0.1	8/9	0.8990	1.0000
	$\Delta_{\text{R-DOC}} - \Delta_{\text{Vanilla}}: \text{GRPO} - \text{Base}$	17	-0.9	8/9	0.8267	1.0000
	$\Delta_{\text{RITL}} - \Delta_{\text{Vanilla}}: \text{GRPO} - \text{SFT}$	16	+2.1	12/4	0.0278	0.1392
	$\Delta_{\text{R-DOC}} - \Delta_{\text{Vanilla}}: \text{GRPO} - \text{SFT}$	16	+0.9	9/7	0.2687	1.0000

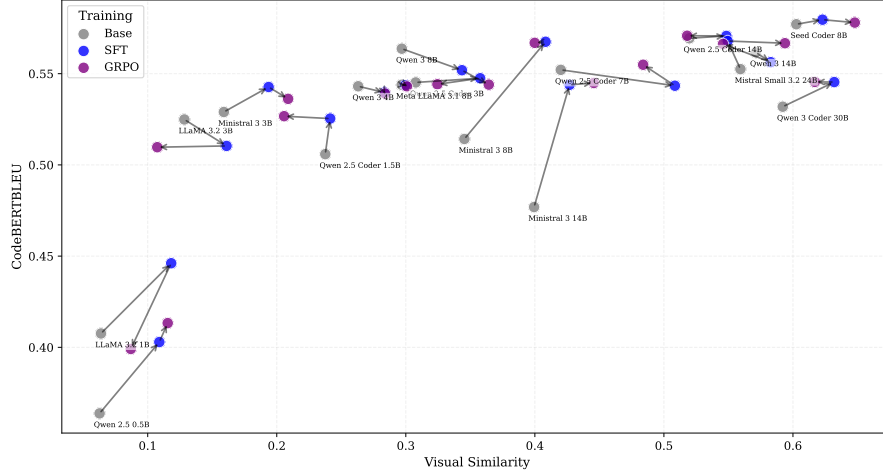


Figure 4: Behaviour of the CodeBERTBLEU score against Visual Similarity during the training cycles. Movement over the SFT stage is predominantly horizontal, along the VS axis, whereas the GRPO stage moves models in either direction and shifts neither metric detectably in aggregate. The figure also highlights the unpredictability of smaller models during training.

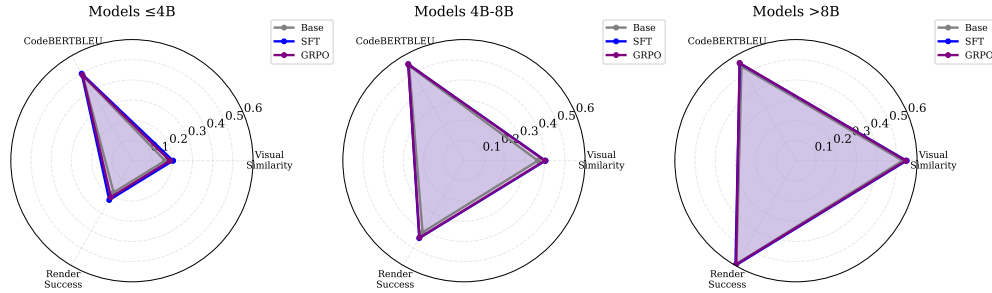


Figure 5: CodeBERTBLEU, Visual Similarity, and Render Success Rate of the models grouped by size category. CBB remains relatively unchanged across the groups, while the VS and RSR regions expand.

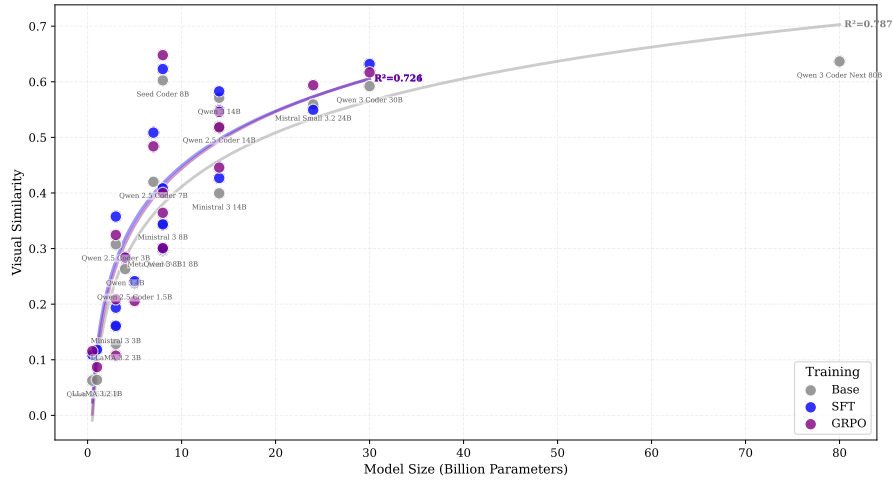


Figure 6: Scaling behaviour of Visual Similarity in the fine-tuned LLMs. The improvement obtained from post-training decreases as the parameter count increases, with the 7B-8B range emerging as an efficiency sweet spot.

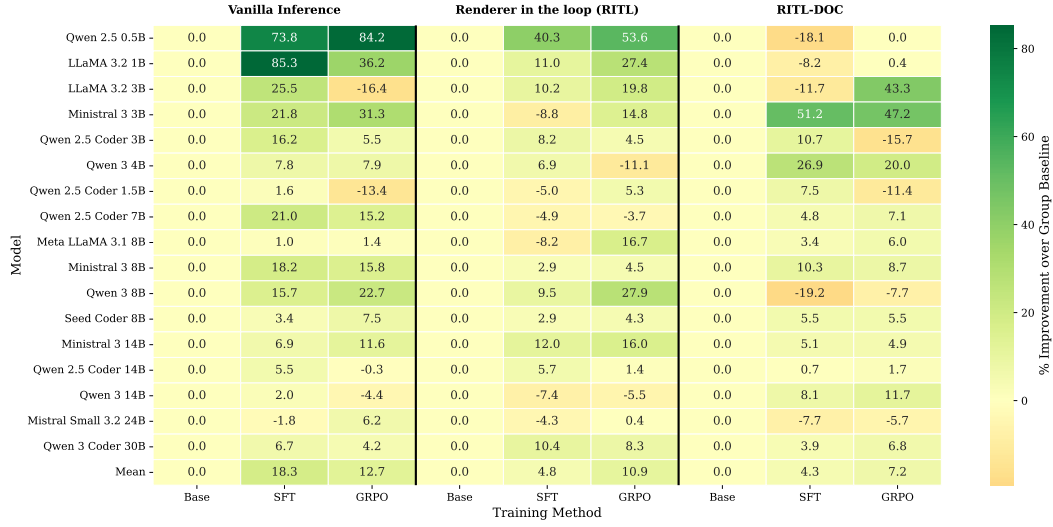


Figure 7: Effect of the post-training strategy on visual similarity, grouped by inference strategy, relative to the base model in each group. The impact of the post-training strategy is particularly significant for smaller models across all inference methods, and more pronounced under vanilla inference than under RITL and RITL-DOC.

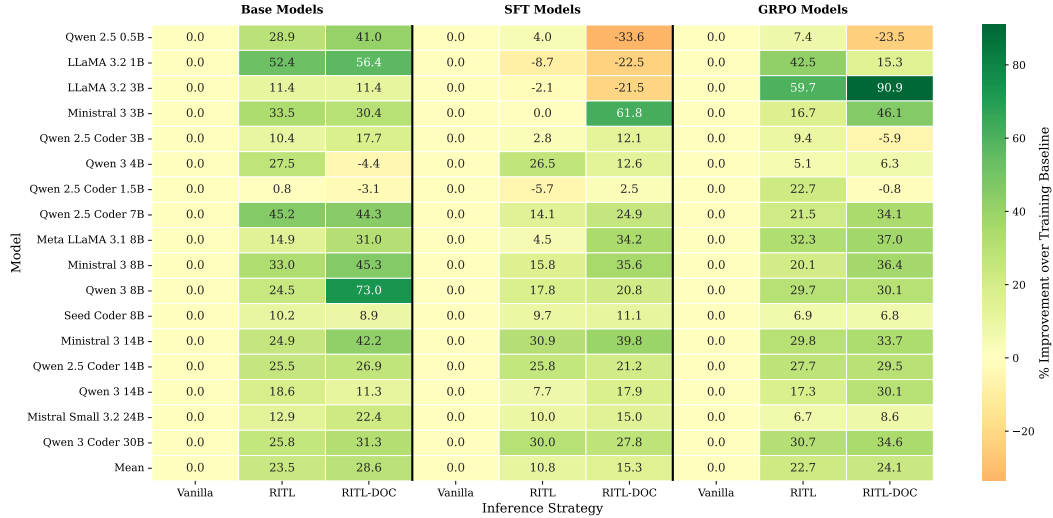


Figure 8: Effect of the inference strategy on visual similarity, grouped by post-training strategy, relative to vanilla inference in each group. RITL and RITL-DOC have a stronger impact on models trained with Base and GRPO, and are less pronounced for SFT models. Smaller models decline under RITL and RITL-DOC when trained with SFT, but improve under GRPO.

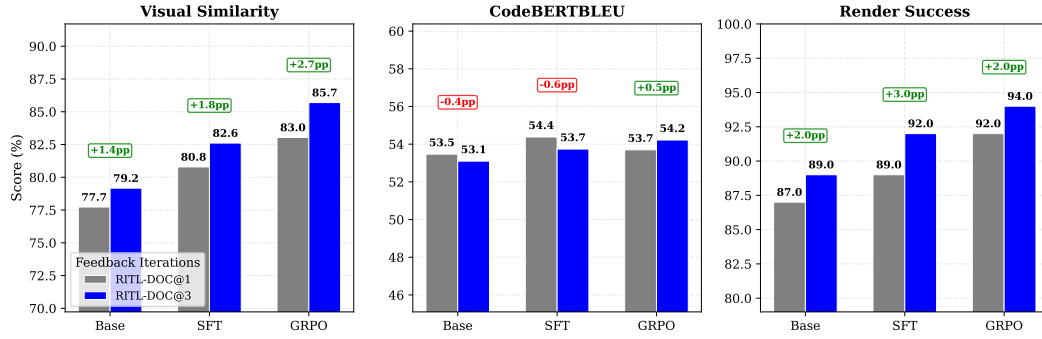


Figure 9: Comparison of performance between RITL-DOC@1 and RITL-DOC@3 for Qwen 3 Coder 30B.

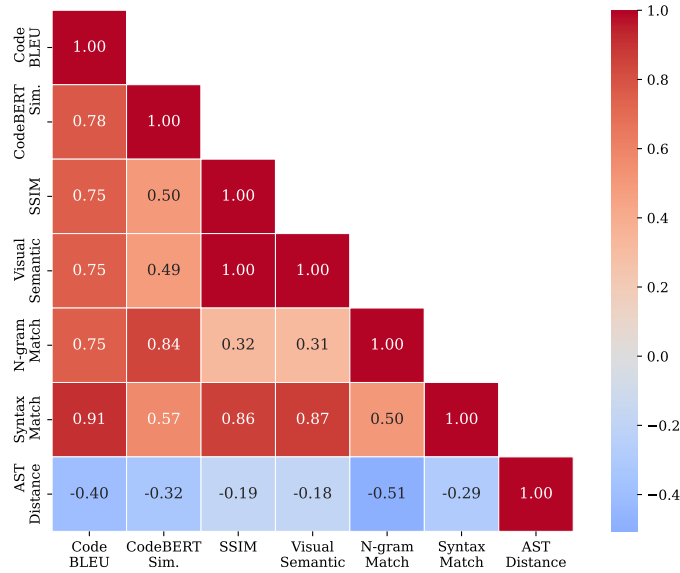


Figure 10: Spearman rank correlation between evaluation metrics, calculated for each model under all combinations of inference and post-training strategy. Within each metric category, the correlation is stronger, as anticipated, while the cross-family correlations between code and visual metrics are relatively weaker. AST Distance weaker correlation with any other metric.

F Qualitative Case Studies

Best cases. Figure 11 shows two examples where GRPO fine-tuning improved the generated Manim code. The Qwen 3 Coder 30B example illustrates that the base model hallucinated API parameters, whereas GRPO guided the model to keep the default parameter value, as in the expected code. Similarly, in the Ministral 3 8B example, the base model hallucinates a built-in function when it is not even needed to create the scene described by the prompt. The GRPO model recognises this and produces code very similar to the reference code.

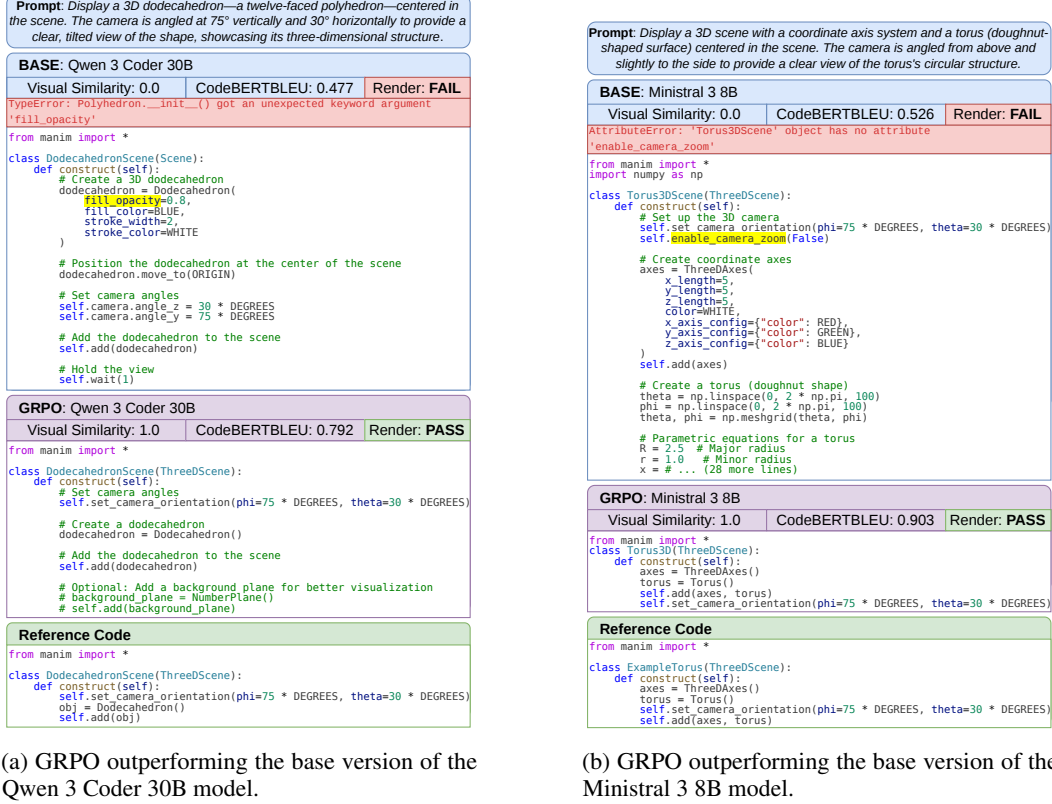


Figure 11: Best case: the fine-tuned model surpasses the base model.

Worst cases. Figure 12 illustrates failure cases where GRPO has caused the degradation of the performance of a base model. One similarity between these two examples is that both are from much smaller models with fewer than 8B parameters. The examples show that the increased spatial awareness of GRPO models has led them to hallucinate functions and API parameters that position visual elements in the required positions.

Rescue cases. The examples in Figure 13 demonstrate cases where inference-level strategies corrected erroneous code. The left figure illustrates an instance of RITL-DOC fixing a hallucinated API parameter with the correct one using Qwen 3 Coder 30B. The right figure depicts the same model learning to correctly utilise the Manim API by employing the visual shapes available out of the box rather than attempting to recreate them.

Outlier cases. Figure 14 shows an outlier where a smaller Ministral 3 8B SFT model successfully generated error-free Manim code when the much larger Mistral Small 3.2 24B model failed. However, as seen in the figure, this is because the model tried to be precise about the visual object's parameters with hallucinated API parameters.

Visual examples. Figure 15 shows a few selected visual examples where the training iterations have increased the visual quality and the accuracy of the generated videos.

Prompt: Display the text "RED COLOR" in red at the center of the screen.		
BASE: LLaMA 3.2 3B		
Visual Similarity: 1.0	CodeBERTBLEU: 0.810	Render: PASS
<pre>from manim import * class RedText(Scene): def construct(self): text = Text("RED COLOR", color=RED) self.add(text.center())</pre>		
GRPO: LLaMA 3.2 3B		
Visual Similarity: 0.0	CodeBERTBLEU: 0.810	Render: FAIL
<pre>AttributeError: Text object has no attribute 'centered' from manim import * class RedText(Scene): def construct(self): text = Text("RED COLOR", color=RED) self.add(text.centered())</pre>		
Reference Code		
<pre>from manim import * class SimpleColor(Scene): def construct(self): col = Text("RED COLOR", color=RED) self.add(col)</pre>		

(a) GRPO underperforming the base version of the LLaMA 3.2 3B model.

Prompt: Display the text "RED COLOR" in red at the center of the screen.		
BASE: Qwen 2.5 Coder 7B		
Visual Similarity: 1.0	CodeBERTBLEU: 0.812	Render: PASS
<pre>from manim import * class RedTextScene(Scene): def construct(self): text = Text("RED COLOR", color=RED) self.add(text) self.wait(2)</pre>		
GRPO: Qwen 2.5 Coder 7B		
Visual Similarity: 0.0	CodeBERTBLEU: 0.797	Render: FAIL
<pre>AttributeError: Text object has no attribute 'to_center' from manim import * class DisplayRedText(Scene): def construct(self): text = Text("RED COLOR", color=RED) text.to_center() self.add(text) self.wait(2)</pre>		
Reference Code		
<pre>from manim import * class SimpleColor(Scene): def construct(self): col = Text("RED COLOR", color=RED) self.add(col)</pre>		

(b) GRPO underperforming the base version of the Qwen 2.5 Coder 7B model.

Figure 12: Worst case: the base model surpasses the fine-tuned model.

Prompt: Display the text "Hello world" in large font (size 144) centered on the screen.		
Vanilla SFT: Qwen Coder 30B		
Visual Similarity: 0.0	CodeBERTBLEU: 0.964	Render: FAIL
<pre>TypeError: Mobject.__init__() got an unexpected keyword argument 'size' from manim import * class HelloWorld(Scene): def construct(self): text = Text("Hello world", size=144) self.add(text)</pre>		
RITL-DOC: Qwen Coder 30B		
Visual Similarity: 1.0	CodeBERTBLEU: 1.000	Render: PASS
<pre>from manim import * class HelloWorld(Scene): def construct(self): text = Text("Hello world", font_size=144) self.add(text)</pre>		
Reference Code		
<pre>from manim import * class HelloWorld(Scene): def construct(self): text = Text("Hello world", font_size=144) self.add(text)</pre>		

(a) RITL-DOC inferencing outperforming vanilla inferencing on the SFT Qwen 3 Coder 30B model.

Prompt: Display a 3D dodecahedron—a twelve-faced polyhedron—centered in the scene. The camera is angled at 75° vertically and 30° horizontally to provide a clear, tilted view of the shape, showcasing its three-dimensional structure.		
Vanilla SFT: Qwen 3 Coder 30B		
Visual Similarity: 0.0	CodeBERTBLEU: 0.639	Render: FAIL
<pre>TypeError: Polyhedron.__init__() got an unexpected keyword argument 'vertices' from manim import * class DodecahedronScene(ThreeDScene): def construct(self): self.set_camera_orientation(phi=75 * DEGREES, theta=30 * DEGREES) dodecahedron = Polyhedron(vertices=[[1, 1, 1], [1, 1, -1], [1, -1, 1], [1, -1, -1], [-1, 1, 1], [-1, 1, -1], [-1, -1, 1], [-1, -1, -1], [0, 1.618, 0.618], [0, 1.618, -0.618], [0, -1.618, 0.618], [0, -1.618, -0.618]], faces=[[0, 1, 3, 2], [4, 5, 7, 6], [0, 4, 6, 2], [1, 5, 7, 3], [0, 4, 5, 1], [2, 3, 7, 6]], face_colors=[BLUE]) self.add(dodecahedron) self.begin_ambient_camera_rotation(rate=0.1)</pre>		
RITL-DOC: Qwen 3 Coder 30B		
Visual Similarity: 1.0	CodeBERTBLEU: 0.903	Render: PASS
<pre>from manim import * class DodecahedronScene(ThreeDScene): def construct(self): self.set_camera_orientation(phi=75 * DEGREES, theta=30 * DEGREES) dodecahedron = Dodecahedron() self.add(dodecahedron) self.begin_ambient_camera_rotation(rate=0.1)</pre>		
Reference Code		
<pre>from manim import * class DodecahedronScene(ThreeDScene): def construct(self): self.set_camera_orientation(phi=75 * DEGREES, theta=30 * DEGREES) obj = Dodecahedron() self.add(obj)</pre>		

(b) RITL-DOC inferencing outperforming vanilla inferencing on the SFT Qwen 3 Coder 30B model.

Figure 13: Rescue case: the inference strategy outperforms vanilla inference.

Prompt: Display a 3D scene with a coordinate axis system and a torus (doughnut-shaped surface) centered in the scene. The camera is angled from above and slightly to the side to provide a clear view of the torus's circular structure		
SFT: Mistral 3 8B		
Visual Similarity: 1.0	CodeBERTBLEU: 0.903	Render: PASS
<pre> from manim import * class Torus3D(ThreeDScene): def construct(self): axes = ThreeDAxes() torus = Torus() self.add(axes, torus) self.set_camera_orientation(phi=75 * DEGREES, theta=30 * DEGREES) </pre>		
SFT: Mistral Small 3.2 24B		
Visual Similarity: 0.0	CodeBERTBLEU: 0.857	Render: FAIL
TypeError: Mobject.__init__() got an unexpected keyword argument 'r1'		
<pre> from manim import * class Torus3D(ThreeDScene): def construct(self): axes = ThreeDAxes() torus = Torus(r1=1, r2=0.3) self.set_camera_orientation(phi=75 * DEGREES, theta=30 * DEGREES) self.add(axes, torus) self.wait() </pre>		
Reference Code		
<pre> from manim import * class ExampleTorus(ThreeDScene): def construct(self): axes = ThreeDAxes() torus = Torus() self.set_camera_orientation(phi=75 * DEGREES, theta=30 * DEGREES) self.add(axes, torus) </pre>		

Figure 14: Outlier case: a smaller Mistral 3 8B SFT model outperforming the Mistral Small 3.2 24B SFT model.



Figure 15: Examples from the top-performing models. The top example shows that GRPO has fixed the issue of text being out of the viewing area, while the middle example shows GRPO fixing overlapping visual elements. The bottom example shows a case where the outputs become increasingly similar to the reference video across training stages.

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 Section 1 state four contributions, and each maps to a subsection of Section 5 where it is presented and explained in detail with adequate mathematical explanations and experimental results.

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: Section 6 discusses the internal, external and construct validity of this work.

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: [N/A]

Justification: The paper does not include purely theoretical results. However, Section 3.1 contains the reward function used for the RL training phase of the model, including the derivation of the visual and text rewards.

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: Section 4 describes the dataset and split, the model list, the LoRA and optimiser settings, and the hardware and software versions. Section 3.1 presents the full reward derivation, and Appendix D presents the statistical testing procedure. The code and configurations are available at the link in Section 4.

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: The dataset used, ManimBench, is publicly available, and a public GitHub repository containing the training and evaluation code, configurations, and trained LoRA adapters is linked in Section 4.

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: Section 4 specifies the data split, batch size, learning rates, schedule, weight decay, warmup ratio, LoRA rank, target modules and quantisation. The Section 3.2 specifies the GRPO group size, completion length, β and ϵ together with how each was chosen.

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: Every paired comparison supporting a main claim has a corresponding exact two-sided Wilcoxon signed-rank test, paired across the 17 models in Table 8 in Appendix D, and all 27 tests are reported in Table 8, together with Holm adjustment within each of the four pre-specified families.

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: Section 4 reports the full setup used for all the experiments reported in the paper.

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. The experiments use a publicly available dataset of Manim code and involve no human subjects or personal data.

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: Section 6 discusses how the pipeline lowers the cost of producing programmatic animations with compact open models. However, it also notes that compact LLMs may make more frequent mistakes than frontier models and argues for keeping a human in the loop for factual review.

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: The released artefacts and code are based on Manim animation code generation, and they do not constitute a general-purpose model. Hence, no high risk of misuse has been identified.

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: The Unsloth model collection, Manim Community Edition (ManimCE), ManimBench, and all LLMs are cited in Sections 2 and 4. All external assets are used in accordance with their licenses and terms of use. ManimCE is licensed under the MIT license, TRL and Unsloth under Apache 2.0, and ManimBench under CC-BY-NC-SA 4.0.

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: All produced artefacts and code are included in the repository linked in Section 4, including the configurations required to reproduce the reported results.

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 study does not involve crowdsourcing or research with 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 study does not involve crowdsourcing or research with human subjects.

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: [Yes]

Justification: LLMs are a core subject of study in this research work. Section 3 describes how the open-source models are further trained with SFT and GRPO. Section 3.3 describes their behaviour under the renderer-in-the-loop inference pipeline. However, the paper itself does not contain any LLM-generated text or artefacts except in Appendix F, which shows example generations from the models experimented with in the study.

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.