FLOORA: A HUMAN-ALIGNED DOMAIN-SPECIFIC LANGUAGE MODEL FOR ARCHITECTURAL DESIGN

Anonymous authors
Paper under double-blind review

ABSTRACT

Foundation models are powerful generators, but many engineering domains require structured representations that general-purpose systems handle poorly. We introduce FLOORA (Floor Layout Optimization with RL Alignment), a family of small domain-specific language (DSL) models for architectural layout generation. With specialized data and alignment, our 0.6B model outperforms much larger frontier models, achieving VLM judge win rates up to **92.0%** on out-of-distribution real-world buildings and **96.0%** on synthetic buildings. Human evaluations further corroborate these results, with FLOORA selected as the best model in **89.3%** of evaluations. FLOORA combines a token-efficient DSL, custom tokenization, domain-specific pretraining, supervised fine-tuning (SFT), and reinforcement learning (RL) with learned human-preference and verifiable rewards. This pipeline improves architectural and geometric validity, supported by extensive empirical evaluation and ablation studies. Although focused on architecture, our results suggest that similar domain-specific recipes may be useful in other engineering domains with structured, verifiable outputs. Datasets, models, and inference code are included in the supplemental.

1 INTRODUCTION

Large language models (LLMs) (Brown et al., 2020) are increasingly evaluated on specialized scientific and engineering tasks (Zhou et al., 2025; Sun et al., 2024; Heesch et al., 2025; Rein et al., 2023), yet it remains unclear whether such domains are best served by continuing to scale general-purpose models (Kaplan et al., 2020; Hoffmann et al., 2022) or by training smaller models around domain-specific data (Beltagy et al., 2019; Lee et al., 2020; Gu et al., 2021; Ouyang et al., 2022).

Architecture, Engineering, and Construction (AEC) provides a useful test case because building designs are structured objects with strict geometric constraints, multiple valid representations, and quality criteria combining verifiable rules with expert judgment. Unlike open-ended text generation, building-design models must produce outputs that can be parsed, validated, and used as geometry. Recent work has benchmarked frontier models on AEC tasks (Mankodiya et al., 2026; Liang et al., 2026; Kondratenko et al., 2026), revealing both their capabilities and limitations. Complementing this work, we ask whether compact DSL models can match or outperform frontier models on an architectural generation task given the right representation, data pipeline, and post-training recipe. We study multifamily residential layout generation from building massings, synthesizing labeled, geometrically parseable floor-plan spaces that satisfy design constraints, and investigate how expert human feedback aligns these models with architectural judgment.

A central obstacle is representation. Standard building formats such as Industry Foundation Classes (IFC) (buildingSMART, 2024) are too verbose and relational for efficient autoregressive language

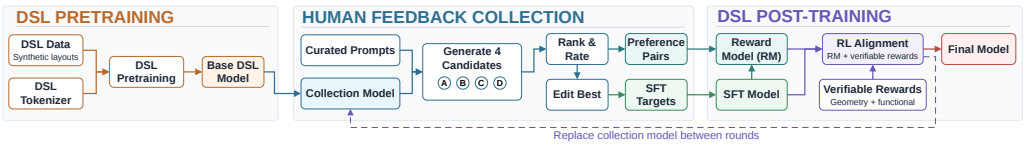


Figure 1: Overview of our DSL pre-training, human feedback collection, and post-training pipeline.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

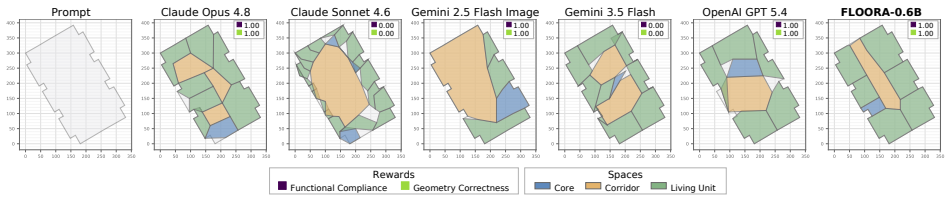


Figure 2: Qualitative comparison of FLOORA-0.6B and frontier models on a real-world test sample. FLOORA outperforms the baselines on geometric and functional checks and architectural quality.

modeling. We introduce a compact DSL that encodes building metadata, massing geometry, and labeled floor-plan spaces as structured text. It can be deterministically parsed, validated, normalized, and converted back to geometry, enabling architectural generation as language modeling while preserving downstream usability. Compared with IFC, it is up to 15× more compact while retaining the geometric and semantic information needed for conceptual layout generation.

We build a pre- and post-training pipeline around this representation, shown in Figure 1. We pretrain compact models on large-scale synthetic architectural data with a DSL tokenizer and evaluate them on out-of-distribution (OOD) real-world OpenStreetMap massings (OpenStreetMap contributors, 2017). Practicing architects then rate, rank, and edit generated layouts, yielding 10k SFT samples and 90k pairwise preferences. Finally, we combine learned and verifiable rewards to optimize architect preferences and explicit design constraints simultaneously.

Notably, this is not a scaling study. Our models are compact, domain-specific, and optimized for structured generation rather than language ability. To test whether the recipe generalizes across backbones, we evaluate Qwen3 (Yang et al., 2025) and Pythia (Biderman et al., 2023) across model sizes. Our results suggest that, for this structured generation task, domain-specific representation and alignment can partially substitute for scale. A 0.6B FLOORA model outperforms much larger frontier models on our layout-generation benchmarks, with Figure 2 showing a qualitative example.

The closest prior work, Yin et al. (2025a), represents room layouts with discrete visual tokens and uses architect feedback for RL. We instead tackle the problem of floor layout by combining an architectural DSL with domain-specific pre-training, enabling direct geometric generation and deterministic verification. In addition, our feedback pipeline uses architect edits for SFT, while our RL pipeline jointly optimizes learned architect preferences and verifiable constraints. To the best of our knowledge, FLOORA is the first architectural post-training framework to jointly optimize a learned reward model with verifiable rewards, requiring careful normalization and balancing of the signals.

Beyond architectural layout generation, our framework suggests a general recipe for specialized domains that combine structured representations, verifiable constraints, and expert-defined quality criteria. More broadly, our results suggest that similar domain-specific approaches may offer an effective alternative to relying primarily on model scale for structured generation tasks. Figure 3 summarizes the full recipe. Our contributions are:

- We show that a token-efficient architectural DSL and custom tokenizer enable effective domain-specific pre-training and post-training, resulting in generalization to OOD real-world massings.
- We develop an architect-guided post-training pipeline that uses direct edits and pairwise preferences to align synthetically pre-trained models with actual design preferences.
- We show that combining learned architect preferences with verifiable rewards generally improves layout generation, with greater reward model benefits at larger scales.
- We demonstrate that compact domain-specific models can outperform substantially larger frontier models across verifiable metrics, VLM judge pairwise comparisons, and human evaluations.
- We release the synthetic and OSM datasets, trained models, and inference code for future research.

2 Related Work

Machine Learning for Architectural Layouts. Optimization and machine learning for architectural layouts have a long history (Weber et al., 2022), typically focusing on room arrangement

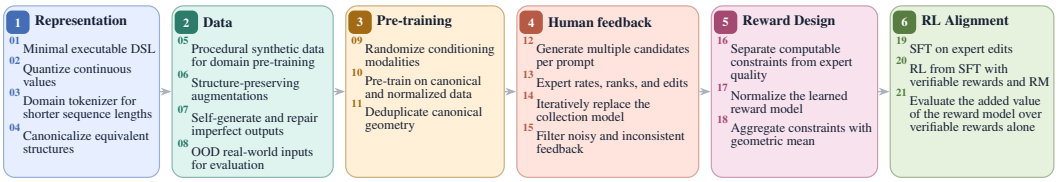


Figure 3: High-level recipe for training and post-training a domain-specific language model.

within homes; we instead model living units, corridors, and cores on multifamily floor plates. Recent generative approaches differ substantially in how they encode geometry. Coordinate-sequence methods use standard model vocabularies: Galanos et al. (2023) represent type labels and corner coordinates, Leng et al. (2023) and Yin et al. (2025b) quantize coordinates, Luo et al. (2024) output full-precision polygon vertices, and de Guevara et al. (2025) combine categorical and continuous room features. Other approaches introduce specialized representations. Armen et al. (2024) generate quantized parametric commands with a custom tokenizer, Klimenko et al. (2026) represent floor plans as space-partition trees from which geometry is recovered procedurally, and Qin et al. (2026) encode massings and rooms using a VQ-VAE. Most closely related, Yin et al. (2025a) represent unit plans with discrete visual tokens and post-train using an architect-preference reward model.

We take a middle ground between unconstrained coordinate sequences and representations that encode geometry implicitly. Our DSL represents layouts directly as compact sequences of labeled closed polygons. Rather than enforcing geometric validity through the representation itself, we use training data and reinforcement learning to learn geometric correctness and architectural validity.

Domain-Specific Language (DSL) Models. A large body of work has explored adapting pretrained language models to specialized domains, motivated by distributional differences in terminology, structure, and domain knowledge. Early approaches primarily focused on domain-specific pre-training or continued pretraining of encoder models (Lee et al., 2020; Beltagy et al., 2019; Huang et al., 2019; Araci, 2019; Chalkidis et al., 2020; Gu et al., 2021; Gupta et al., 2022). More generally, domain-adaptive pretraining has been shown to improve downstream performance by continuing training on data drawn from the target distribution (Gururangan et al., 2020). More recent work extends domain specialization to generative LLMs, typically through domain-focused continued pretraining of general-purpose models (Luo et al., 2022; Taylor et al., 2022; Wu et al., 2023; 2024; Chen et al., 2023; Labrak et al., 2024; Xie et al., 2024; Colombo et al., 2024; Yang et al., 2023), spanning biomedical, scientific, financial, and legal applications.

DSL models remain limited in AEC. Li et al. (2025) combine CAD-specific pretraining with instruction tuning for parametric 3D generation, while Govindarajan et al. (2025) fine-tune code language models on sequential CAD data. In architecture, Lin et al. (2026) fine-tune a general-purpose LLM on BIM-derived question-answering and reasoning data. In contrast, our work provides an end-to-end pipeline for generative architectural design, spanning domain-specific pre-training through expert-guided post-training including supervised fine-tuning and reinforcement learning.

3 Domain-Specific Language (DSL)

To enable scalable LLM training for architectural design, we develop a DSL and parser library that represent building geometry as compact structured text. Unlike Industry Foundation Classes (IFC), whose verbose structure produces long sequences, our DSL captures only the information needed for conceptual design in a semantically structured format optimized for token efficiency and deterministic parsing. The DSL encodes building massing, floor-plan spaces, and metadata as human-readable text suitable for LLMs. The parser provides full roundtrip support, allowing generated DSL to be parsed, validated, normalized, and converted back to geometry. Together, these components bridge free-form LLM outputs and structurally valid building representations for both training data preparation and post-generation validation. See Appendix A for details.

Figure 11 and Listing 1 in Appendix A show example DSL representations and encodings. The DSL in Figure 11a uses 642 characters versus over 10,000 for equivalent IFC, a 15× reduction, while preserving the geometric and semantic information needed for conceptual floor-plan generation.

4 Data Generation and Processing

4.1 Synthetic Data

Data synthesis proceeds in four stages, each broadening the data distribution. First, we derive labeled floor plans from TileGPT (Gaier et al., 2024; Villaggi et al., 2024), a quality-diversity system for high-performing tile-based layouts, and project their geometry into the DSL. Second, we apply scale augmentation to diversify massing and space dimensions beyond the tile grid. Third, we sample massings outside the tile-based distribution, generate layouts with a model checkpoint trained on the preceding synthetic data, and repair near-miss outputs, retaining only those that pass a correctness verifier. This enables generation on free-form massings beyond the tile-based distribution. The verifier follows the verifiable reward criteria from Section 6.2. Fourth, we apply rotation augmentation, sampling 20 angles per layout so that the model sees both near-identity perturbations and large reorientations; it runs last, after the processing steps of Section 4.3, so that no rotated variant of a training layout can reach evaluation. Both scale and rotation augmentations are structure-preserving. The synthetic corpus contains 4.1M samples before rotation augmentation and 82M after. Appendix B.1 provides additional details.

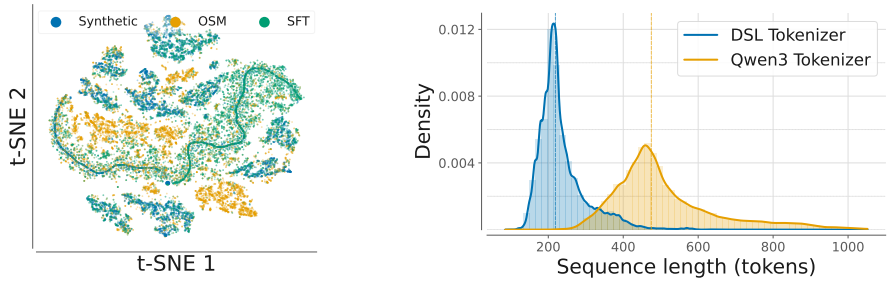
4.2 Real-World Data

We collect 14k real-world multifamily building footprints from OpenStreetMap (OSM) (OpenStreetMap contributors, 2017) across more than 50 major North American cities. The dataset contains building massings and metadata converted to our DSL. Because OSM lacks ground-truth space layouts, it is used for evaluation rather than training. Figure 4a visualizes the massings with t-SNE (Van der Maaten & Hinton, 2008), showing substantial differences between the synthetic and OSM massing distributions, with many OSM samples lying outside the synthetic distribution. This motivates its use for out-of-distribution evaluation. See Appendix B.3 for t-SNE details and Appendix B.4 for data samples.

4.3 Data Processing

Data Quantization. Continuous data is quantized using a bucketing strategy. We limit prompts and completions to a physical extent of $102.4\text{m} \times 102.4\text{m}$ and divide this domain into 1024 buckets of 0.1m, effectively quantizing layouts onto a $10\text{cm} \times 10\text{cm}$ grid. This resolution is sufficient for the expected architectural applications. Continuous values are recovered by mapping each bucket to a fixed value within its range, with error scaling linearly with the grid spacing.

Data Normalization. A floor plan can have different DSL strings due to polygon start vertices, space ordering, and absolute position. We therefore emit each sample in canonical form. Using the parser's winding-order normalization (see Appendix A.1), we translate each sample so the minimum coordinate is the origin, start each polygon's vertex sequence at the vertex nearest the origin, and



(a) t-SNE of massing polygons.

(b) Sequence length distributions.

Figure 4: **(a)** t-SNE visualization of massing polygons across synthetic, OSM, and SFT datasets, indicating the distribution shift between synthetic and real-world massing geometries. **(b)** Tokenized sequence length distributions for the pre-training dataset using the DSL tokenizer and the original Qwen3 tokenizer, showing that the DSL tokenizer reduces sequence lengths by 54%, on average.

order spaces deterministically. The same transformation is applied at inference to match the training distribution. Canonicalization removes design-irrelevant variation and makes deduplication exact.

Deduplication and Splitting. Procedural generation can produce duplicates that inflate evaluation if shared across splits. We therefore hash each sample's canonical form and remove from validation and test any sample whose hash occurs in training. Rotation augmentation is applied after splitting, preventing training footprints from appearing in evaluation as exact or rotated duplicates.

5 DSL Pre-training

DSL pre-training adapts general-purpose models to the DSL for syntactically valid generation.

Prompt Structure and Modality Randomization. Training samples use three DSL control tags: context, generate, and completion, as shown in Figure 5. Context provides conditioning information such as building metadata, massing geometry, and optional partial layouts; generate specifies the target modality; and completion marks the autoregressive target. Models are trained with standard next-token prediction. During pre-training, modalities are randomly split between context and completion, with subsets of spaces provided as context and the remainder withheld for generation. This teaches both full and partial layout synthesis across varied modality combinations.

Custom Tokenizer and Vocabulary. We use a custom tokenizer specialized for the DSL, with a vocabulary of DSL keywords, delimiters, and quantized numeric tokens. As shown in Figure 4b, this substantially shortens sequences relative to general-purpose tokenizers, yielding a pre-training corpus of 17.6B tokens after augmentation. We ablate the impact of DSL-specific tokenization on training and downstream generation in Section 7.4 and Appendix I.2.

Base Models. To test whether our recipe generalizes across model families and scales, we pre-train Qwen3 (Yang et al., 2025) (0.6B, 1.7B) and Pythia (Biderman et al., 2023) (70M, 160M, 410M, 1.4B). Resizing the embedding layers to the smaller DSL vocabulary reduces effective parameter counts, while all remaining weights are initialized from the corresponding base models. The Chinchilla scaling rule of approximately 20 training tokens per parameter (Hoffmann et al., 2022) suggests a compute-optimal scale near 0.9B parameters for our 17.6B-token corpus, though this estimate was developed for general-purpose pre-training. In our experiments, performance largely saturates above $\sim$0.4B parameters on synthetic data and improves only modestly at larger scales on OSM, motivating our focus on compact models up to $\sim$1.4B effective parameters. We use a fixed effective batch size and tune learning rates per model; see Appendix D.1 for details.

6 DSL Post-training

The base model is limited by its reliance on synthetic data. Although DSL pre-training enables parseable outputs, the model inherits simplified design patterns and often produces architecturally weak layouts, with recurring issues in circulation, core placement, unit proportions, geometry, and labeling. We therefore combine human feedback alignment with verifiable rewards: practicing architects provide supervision that corrects synthetic-data biases and reflects professional judgment, while rule-based rewards enforce automatically verifiable geometric and functional constraints.

6.1 Human Feedback Alignment

We collect human feedback through an expert annotation workflow producing ratings, rankings, and corrected layouts (Figure 1). For each massing, sampled from curated synthetic and manually created footprints, the model generates four candidate floor plans.

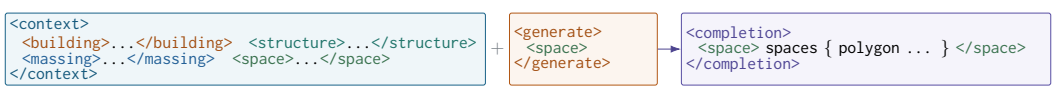


Figure 5: Structured prompt and completion format used for DSL pre-training and post-training.

Architectural labelers rate each candidate on a 5-point Likert scale, rank them with ties permitted, and edit the highest-ranked layout. Edited layouts provide SFT targets, rankings yield pairwise preferences for reward modeling, and ratings filter noisy or inconsistent preferences. Feedback is collected iteratively across multiple rounds, with SFT and RL updates between rounds so that later collection uses progressively improved models. In total, we collect 10k SFT samples and 90k pairwise preference examples. Our setup follows standard human feedback and preference-based alignment practices (Ouyang et al., 2022; Lambert, 2026); see Appendix C for additional details.

6.2 Verifiable Rewards

While human feedback captures holistic design quality and architectural judgment, verifiable rewards automatically enforce precise geometric and functional constraints. By construction, reward terms are bounded to $[0, 1]$ and penalty terms to $[-1, 0]$. Appendix E provides additional details.

Geometric Correctness. This reward measures whether the generated floor plan satisfies fundamental spatial constraints. It comprises three checks: spaces must be contained within the building massing, spaces must not overlap one another, and the massing footprint must be fully occupied by spaces. Each check produces a continuous score, and the scores are combined via geometric mean.

Functional Compliance. This reward evaluates whether the generated floor plan satisfies the requirements of residential buildings. It checks that the numbers of cores and corridors fall within acceptable ranges, that a sufficient fraction of living units meet a minimum area threshold and are properly labeled. The individual check scores are aggregated using a geometric mean.

Soft Overlong Penalty. This reward penalizes overlong completions to discourage unnecessarily verbose or degenerate generations (Yu et al., 2026). It is applied only as a penalty for completions exceeding the target length budget and does not provide positive reward for shorter completions.

6.3 Post-Training Methodology

Supervised Fine-Tuning. The first post-training stage applies SFT on architect-edited layouts. Examples are formatted as prompt-completion pairs, with the conditioning context in the prompt and the target DSL sequence in the completion. The loss is computed only on completion tokens, and the SFT data is augmented using the same rotational transformations as pre-training.

Reward Model Training. The reward model (RM) is initialized from the SFT checkpoint by replacing the language modeling head with a scalar reward head. It is trained on architect-ranked pairwise preferences using the Bradley–Terry objective (Bradley & Terry, 1952; Ouyang et al., 2022): $-\log \sigma \left(r_\theta(x, y_w) - r_\theta(x, y_l)\right)$, where y_w and y_l are the preferred and rejected completions for prompt x. The preference data is also augmented with rotational transformations.

Reinforcement Learning. The final stage performs RL using GRPO (Shao et al., 2024), initialized from the SFT checkpoint, with prompts sampled from the synthetic and SFT data mixture. The reward function combines a frozen RM with verifiable rewards. Let R_{RM} denote the RM score and let $\{R_i\}_{i=1}^{N}$ denote the set of verifiable rewards described in Section 6.2. The overall reward is $R = \lambda_{\mathrm{RM}} \hat{R}_{\mathrm{RM}} + \sum_{i=1}^{N} \lambda_i R_i$, where λ_{RM} and λ_i are weighting coefficients. Since the RM score is unconstrained while verifiable rewards are bounded by construction, we normalize the RM score as:

$$\hat{R}_{\mathrm{RM}} = \tanh\left(\frac{R_{\mathrm{RM}}}{\alpha}\right), \tag{1}$$

where α is a fixed normalization scale. This bounds the RM contribution to $[-1, 1]$ and prevents it from dominating the optimization objective. We note that the normalization also affects the effective preference scale observed by the policy during RL, which can alter the calibration of the RM. We therefore perform ablations over the normalization scale and report the results in Section 7.4.

All post-training stages update the full set of model parameters rather than using parameter-efficient fine-tuning methods. Similarly to the pre-training setup, the effective batch size is held constant across model scales within each post-training stage, while learning rates are tuned separately for each model and stage. Additional training details and hyperparameters are provided in Appendix D.2.

7 RESULTS

7.1 EVALUATION METHODOLOGY

We evaluate on 2k prompts from each of the synthetic and OSM test sets using the verifiable rewards in Section 6.2. We report pass@1/3/5, where a completion succeeds only if all checks pass. We also use a VLM judge to compare rendered layouts from the same input massing based on architectural quality, geometric validity, and functional usability. Each pair is evaluated in both candidate orders, with disagreements counted as ties. We use Gemini 3.5 Flash as the judge because it performs best among the frontier models in Section 7.3. We additionally validate this judge against architect-implied preferences from the human-feedback dataset, finding 77.5% agreement with human feedback across 10k pairs; see Appendix F for details. To the best of our knowledge, there is no publicly available specialized model that can be directly evaluated on the same massing-conditioned floor-plan layout generation task. We therefore compare FLOORA-0.6B (Qwen3) against frontier models using verifiable pass@k, VLM judge win rates on synthetic and OSM test prompts, and human evaluations. See Appendix G for the frontier-model inference protocol.

Because pre-training is costly, we train each base model with one seed. For post-training, we train each setting with 3 seeds. We evaluate all resulting checkpoints with 5 evaluation seeds, reporting aggregate performance with 95% confidence intervals (CIs). Frontier baselines are API-only models, so we evaluate each with a single inference run under a fixed prompting and decoding protocol.

7.2 MAIN RESULTS

Pre-Training. Full results for the base models are presented in Figure 19 and Tables 8 and 9 in Appendix H.1. Models with fewer than 0.4B parameters fail to effectively capture the underlying building patterns, leading to poor performance even on the synthetic dataset. Although larger models achieve near-saturated performance on the synthetic benchmark, their generalization capability continues to improve on OSM, which serves as an out-of-distribution evaluation setting for the base models. Given the inadequate performance of smaller models, such as Pythia-70M and Pythia-160M, we do not perform any post-training experiments on these models.

Supervised Fine-Tuning. Full SFT results are presented in Figure 21 in Appendix H.2. SFT significantly improves performance on the OSM test set across model scales, showing that architect-edited layouts provide an effective alignment signal for real-world building footprints. Since OSM is out-of-distribution relative to the synthetic and SFT data (see Figure 4a), these gains suggest improved generalization beyond the procedural pre-training distribution.

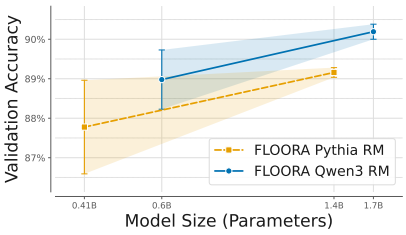
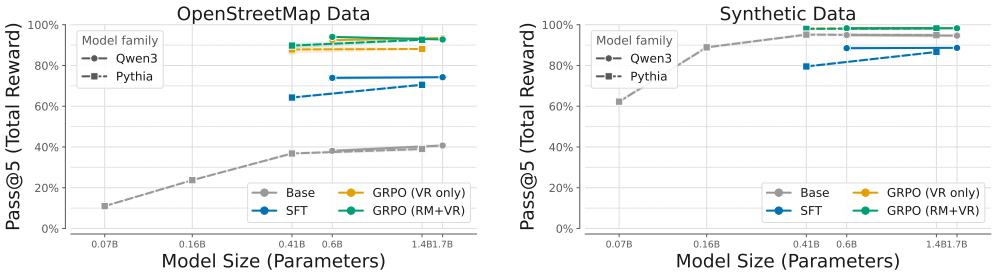


Figure 7: Reward model validation accuracy, reported with 95% CIs.

On the synthetic test set, we observe a decrease in performance after SFT, which we attribute to distribution shift between architect-corrected SFT targets and the procedu-



(a) Pass@5 on the OSM test set.

(b) Pass@5 on the synthetic test set.

Figure 6: Pass@5 for FLOORA models across model sizes and training stages, where success requires all checks to pass. Shading shows 95% CIs across 3 training and 5 evaluation seeds.

ral synthetic layouts used during pre-training (Figure 4a). Overall, these results suggest that SFT functions primarily as a domain-alignment step, trading some in-distribution synthetic performance for better out-of-distribution generalization and closer alignment with expert architectural judgment.

Reward Model Training. We evaluate each RM on held-out pairwise preference data, reporting validation accuracy in Figure 7 as the fraction of pairs where the architect-preferred layout receives a higher score. All RMs achieve high accuracy, capturing architect preference signals. Accuracy also increases with model size, suggesting that larger RMs better capture architectural preferences. Additional results are provided in Appendix H.3.

Reinforcement Learning. Figure 6 reports Pass@5 for the base and post-trained models, with full pass@1/3/5 results in Figure 24 and Tables 10 and 11 in Appendix H.4. Both GRPO variants substantially improve OSM and synthetic performance, resolving many validity and constraint-satisfaction errors left by the base and SFT models. These gains are consistent across datasets and model scales. GRPO also recovers the SFT drop on synthetic data, likely caused by distribution shift, while preserving the OSM gains from SFT.

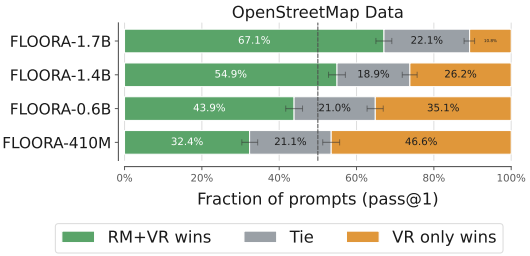
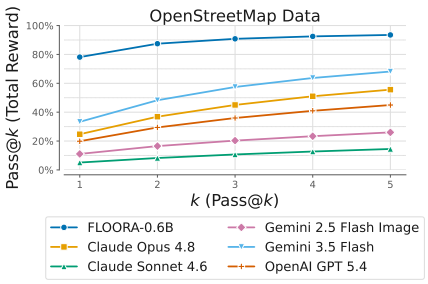


Figure 8: Pairwise VLM judge outcomes for each GRPO (RM+VR) model against its matched GRPO (VR only) counterpart. Bars show the fraction of prompts where RM+VR wins, ties, or loses. As model size increases, the RM+VR variant is increasingly preferred. Error bars indicate 95% Wilson intervals over the evaluation prompts.

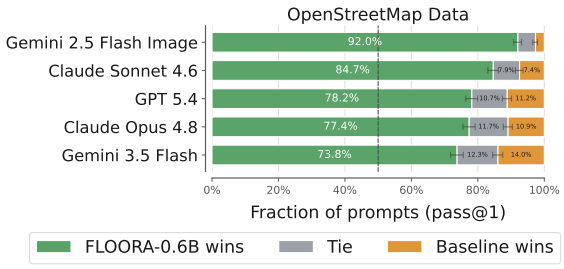
We further isolate the effect of the learned reward model by comparing GRPO (RM+VR) with matched GRPO (VR only) models using the VLM judge. As shown in Figure 8, RM+VR is scale-dependent: it underperforms VR only for Pythia-410M, but becomes increasingly preferred with model capacity, reaching the largest margin for Qwen3-1.7B. This suggests that the reward model can improve qualitative architectural preferences beyond VR-only training, but only when both the reward model and policy have enough capacity to learn and act on this preference signal. Qualitative results in Appendix H.6.2 show the relative benefits of RM+VR in the coherence of space organization, unit proportions, core placement, and circulation quality.

7.3 Comparison with Frontier Models

We compare FLOORA-0.6B against several frontier models using the protocol in Appendix G. FLOORA substantially outperforms all baselines across pass@k (Figure 9a) and is preferred by the pairwise VLM judge at win rates of 73.8–92.0% (Figure 9b). We additionally conduct human



(a) Pass@k on the OSM test set.



(b) Win rates on the OSM test set.

Figure 9: **(a)** Pass@k comparison of FLOORA-0.6B (Qwen3) and frontier baselines, where success requires passing both geometric and functional checks. **(b)** Pairwise VLM judge outcomes for FLOORA-0.6B (Qwen3) against frontier baselines. Bars show the fraction of prompts for which FLOORA wins, ties, or loses. Our model substantially outperforms all evaluated frontier baselines.

evaluations on 100 test inputs, comprising 50 OSM and 50 synthetic samples, with 10 labelers. FLOORA is selected as the best model in 89.3% of evaluations (Figure 10). Additional results and details are provided in Appendix H.5. Thus, FLOORA's advantage extends beyond verifier-based pass@k metrics to visual and architectural quality. Qualitative comparisons in Figure 2 and Appendix H.6.1 show more coherent circulation, regular living-unit subdivisions, and plausible unit proportions. Some frontier outputs still pass geometric and functional checks while exhibiting practical issues such as weak corridor connectivity or irregular unit geometry.

7.4 Ablation Studies

We ablate key design choices and feedback data size using FLOORA-0.6B as the reference.

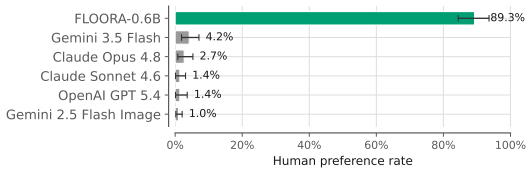


Figure 10: Human evaluation results across 100 test inputs with 95% CIs. FLOORA is selected as the best model in 89.3% of evaluations.

RM Normalization. Appendix I.1 ablates the RM normalization scale α in Equation (1). Without normalization, the RM score dominates the bounded verifiable rewards. Performance is robust for $\alpha \in [1, 4]$, while $\alpha = 10$ degrades verifiable rewards. We select $\alpha = 3$ as a balanced setting that preserves high verifiable rewards while improving the RM signal.

Original vs. DSL Tokenizers. Appendix I.2 ablates tokenizer choice. The DSL tokenizer reduces mean sequence length by 54% (Figure 4b), shrinking the augmented pre-training corpus from 38.5B to 17.6B tokens, wall-clock pre-training time by 31%, and GPU-hours by 66%. It also improves downstream OSM pass@5 by 6.5–7.2 percentage points for the GRPO models.

Human Feedback Scale. Appendix I.3 studies performance as human feedback increases during iterative collection. Because model updates between rounds couple data quantity with collection-model quality, we use cumulative chronological subsets reflecting the feedback available at each stage. Performance improves, with larger early gains and diminishing returns at larger scales. Although this does not isolate data quantity alone, it reflects the practical collection setting and suggests that performance can indicate when additional annotation offers limited marginal benefit.

8 Limitations

Our models are designed for structured architectural generation rather than broad natural language understanding, and should not be interpreted as general-purpose design assistants. We also intentionally do not study agentic workflows, since such systems typically rely on repeated inference from large frontier models and therefore retain the high cost and latency that our compact domain-specific approach seeks to reduce. Finally, our data is deliberately scoped to multifamily residential floor-plan generation conditioned on a building massing. While our model generalizes well across diverse real-world building footprints from OpenStreetMap (pass@5 of 94%), performance may be lower on highly irregular massings that are underrepresented in the training and post-training data.

9 Conclusion

We presented a compact domain-specific modeling recipe for architectural layout generation, combining a token-efficient DSL, domain-specific pre-training, and architect-guided post-training with learned and verifiable rewards. Our results show that a small DSL model can outperform significantly larger frontier models on the evaluated architectural layout generation benchmarks, across automatic metrics, VLM judge comparisons, and human evaluations. Ablations highlight the importance of representation design, reward model normalization, and human feedback, and show that reward model based post-training benefits from larger model capacity. More broadly, our results suggest that similar domain-specific representations, constraints, and expert-guided alignment strategies may be useful for other structured engineering generation tasks.

AI USE STATEMENT

In this work, we used generative AI tools to assist in the implementation of methods and experiments. For synthetic data generation, we used an intermediate checkpoint of our own trained model to generate layouts. We also used a VLM as a judge for portions of the experimental evaluation and used inference APIs of generative AI models as experimental baselines. Additionally, we used generative AI tools to create or edit software code, draft and edit parts of the research paper, improve readability, grammar, and word choice, rewrite and paraphrase text, and assist with the related work section, including summarizing and analyzing existing literature.

We did not use generative AI tools to generate research ideas, propose or refine the central research hypotheses of this work, interpret experimental results, or draw conclusions from the results. The interpretation of the results and the conclusions presented in the paper were developed by the authors.

All AI-assisted code was reviewed and verified by the authors, and all AI-assisted text included in the paper was reviewed and edited by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

The human-feedback component of this work involved professional architects who were engaged and compensated through a BIM consultancy contractor to evaluate, rank, and edit model-generated architectural layouts. The real-world evaluation data are derived from publicly available OpenStreetMap building footprints and do not contain private architectural plans or personally identifiable information. FLOORA is narrowly scoped to structured, conceptual multifamily residential floor-plan generation. We therefore expect the potential for misuse, as well as privacy and security concerns, to be limited compared with general-purpose or agentic systems.

REPRODUCIBILITY STATEMENT

To support reproducibility, we provide detailed descriptions of the DSL and parser in Appendix A, synthetic data generation and processing in Appendix B, the human-feedback collection and processing pipeline in Appendix C, training settings and hyperparameters in Appendix D, verifiable reward definitions in Appendix E, the VLM-judge evaluation protocol in Appendix F, and the frontier-model inference protocol in Appendix G. Additional quantitative results and ablation studies are provided in Appendices H and I.

We release the synthetic and OSM datasets, post-trained model checkpoints, and inference code as supplementary materials to further support future research. The human-feedback data used for supervised fine-tuning and reward model training and the training code are not included in the release. The paper and appendices document the corresponding data collection, processing, training procedures, hyperparameters, reward formulations, and evaluation protocols to facilitate reproduction of the reported experiments.

REFERENCES

Dogu Araci. Finbert: Financial sentiment analysis with pre-trained language models. *arXiv preprint arXiv:1908.10063*, 2019.

Armen et al. Scenescript: Reconstructing scenes with an autoregressive structured language model. Meta Research, 2024.

Iz Beltagy, Kyle Lo, and Arman Cohan. Scibert: A pretrained language model for scientific text. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pp. 3615–3620, 2019.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al.

Pythia: A suite for analyzing large language models across training and scaling. In *International conference on machine learning*, pp. 2397–2430. PMLR, 2023.

Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

buildingSMART. Industry foundation classes (ifc), 2024. URL `https://www.buildingsmart.org/standards/bsi-standards/industry-foundation-classes`.

Ilias Chalkidis, Manos Fergadiotis, Prodromos Malakasiotis, Nikolaos Aletras, and Ion Androutsopoulos. Legal-bert: The muppets straight out of law school. In *Findings of the association for computational linguistics: EMNLP 2020*, pp. 2898–2904, 2020.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

Zeming Chen, Alejandro Hernández Cano, Angelika Romanou, Antoine Bonnet, Kyle Matoba, Francesco Salvi, Matteo Pagliardini, Simin Fan, Andreas Köpf, Amirkeivan Mohtashami, et al. Meditron-70b: Scaling medical pretraining for large language models. *arXiv preprint arXiv:2311.16079*, 2023.

Pierre Colombo, Telmo Pessoa Pires, Malik Boudiaf, Dominic Culver, Rui Melo, Caio Corro, Andre FT Martins, Fabrizio Esposito, Vera Lúcia Raposo, Sofia Morgado, et al. Saullm-7b: A pioneering large language model for law. *arXiv preprint arXiv:2403.03883*, 2024.

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations*, volume 2024, pp. 35549–35562, 2024.

Manuel Ladron de Guevara, Jinmo Rhee, Ardavan Bidgoli, Vaidas Razgaitis, and Michael Bergin. Tokenizing buildings: A transformer for layout synthesis. *arXiv preprint arXiv:2512.04832*, 2025.

Igor Dejanović, Renata Vaderna, Gordana Milosavljević, and Željko Vuković. Textx: a python tool for domain-specific languages implementation. *Knowledge-based systems*, 115:1–4, 2017.

Adam Gaier, James Stoddart, Lorenzo Villaggi, and Shyam Sudhakaran. Generative design through quality-diversity data synthesis and language models. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 823–831, 2024.

Theodoros Galanos, Antonios Liapis, and Georgios N. Yannakakis. Architext: Language-driven generative architecture design, 2023.

Prashant Govindarajan, Davide Baldelli, Jay Pathak, Quentin Fournier, and Sarath Chandar. Cadmium: Fine-tuning code language models for text-driven sequential cad design. *arXiv preprint arXiv:2507.09792*, 2025.

Yu Gu, Robert Tinn, Hao Cheng, Michael Lucas, Naoto Usuyama, Xiaodong Liu, Tristan Naumann, Jianfeng Gao, and Hoifung Poon. Domain-specific language model pretraining for biomedical natural language processing. *ACM Transactions on Computing for Healthcare (HEALTH)*, 3(1): 1–23, 2021.

Tanishq Gupta, Mohd Zaki, NM Anoop Krishnan, and Mausam. Matscibert: A materials domain language model for text mining and information extraction. *npj Computational Materials*, 8(1): 102, 2022.

Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A Smith. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pp. 8342–8360, 2020.

René Heesch, Sebastian Eilermann, Alexander Windmann, Alexander Diedrich, and Oliver Niggemann. Evaluating large language models for real-world engineering tasks. In *Australasian Joint Conference on Artificial Intelligence*, pp. 54–66. Springer, 2025.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, pp. 30016–30030, 2022.

Pin-Lun Hsu, Yun Dai, Vignesh Kothapalli, Qingquan Song, Shao Tang, Siyu Zhu, Steven Shimizu, Shivam Sahni, Haowen Ning, and Yanning Chen. Liger kernel: Efficient triton kernels for llm training. *arXiv preprint arXiv:2410.10989*, 2024.

Kexin Huang, Jaan Altosaar, and Rajesh Ranganath. Clinicalbert: Modeling clinical notes and predicting hospital readmission. *arXiv preprint arXiv:1904.05342*, 2019.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Nikita Klimenko, Hesam Salehipour, Parham Eftekhar, Amir Khasahmadi, and Ramon Elias Weber. Hypergraphformer: Learning hypergraphs from llms for editable floor plan generation, 2026.

Aleksei Kondratenko, Mussie Birhane, Houssame E Hsain, and Guido Maciocci. Aecv-bench: Benchmarking multimodal models on architectural and engineering drawings understanding. *arXiv preprint arXiv:2601.04819*, 2026.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Yanis Labrak, Adrien Bazoge, Emmanuel Morin, Pierre-Antoine Gourraud, Mickael Rouvier, and Richard Dufour. Biomistral: A collection of open-source pretrained large language models for medical domains. In *Findings of the association for computational linguistics: acl 2024*, pp. 5848–5864, 2024.

Nathan Lambert. *Reinforcement Learning from Human Feedback*. Online, 2026. URL `https://rlhfbook.com`.

Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. Biobert: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240, 2020.

Sicong Leng, Yang Zhou, Mohammed Haroon Dupty, Wee Sun Lee, Sam Conrad Joyce, and Wei Lu. Tell2design: A dataset for language-guided floor plan generation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14680–14697, 2023.

Jiahao Li, Weijian Ma, Xueyang Li, Yunzhong Lou, Guichun Zhou, and Xiangdong Zhou. Cad-llama: leveraging large language models for computer-aided design parametric 3d model generation. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 18563–18573. IEEE, 2025.

Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, et al. Pytorch distributed: Experiences on accelerating data parallel training. *Proceedings of the VLDB Endowment*, 13(12), 2020.

Chen Liang, Zhaoqi Huang, Haofen Wang, Fu Chai, Chunying Yu, Huanhuan Wei, Zhengjie Liu, Yanpeng Li, Hongjun Wang, Ruifeng Luo, et al. Aecbench: A hierarchical benchmark for knowledge evaluation of large language models in the aec field. *Advanced Engineering Informatics*, 71: 104314, 2026.

Jia-Rui Lin, Yun-Hong Cai, Xiang-Rui Ni, Shaojie Zhou, and Peng Pan. Qwen-bim: developing large language model for bim-based design with domain-specific benchmark and dataset. *arXiv preprint arXiv:2602.20812*, 2026.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Renqian Luo, Liai Sun, Yingce Xia, Tao Qin, Sheng Zhang, Hoifung Poon, and Tie-Yan Liu. Biogpt: generative pre-trained transformer for biomedical text generation and mining. *Briefings in bioinformatics*, 23(6):bbac409, 2022.

Zhi Hao Luo, Luis Lara, Ge Ya Luo, Florian Golemo, Christopher Beckham, and Christopher Pal. Dstruct2design: Data and benchmarks for data structure driven generative floor plan design, 2024.

Harsh Mankodiya, Chase Gallik, Theodoros Galanos, and Andriy Mulyar. Aec-bench: A multimodal benchmark for agentic systems in architecture, engineering, and construction. *arXiv preprint arXiv:2603.29199*, 2026.

OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org . `https://www.openstreetmap.org`, 2017.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35: 27730–27744, 2022.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Sizhong Qin, Ramon Elias Weber, and Xinzheng Lu. Tokenization allows multimodal large language models to understand, generate and edit architectural floor plans. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10430–10440, 2026.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: international conference for high performance computing, networking, storage and analysis*, pp. 1–16. IEEE, 2020.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 3505–3506, 2020.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Liangtai Sun, Yang Han, Zihan Zhao, Da Ma, Zhennan Shen, Baocai Chen, Lu Chen, and Kai Yu. Scieval: A multi-level large language model evaluation benchmark for scientific research. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19053–19061, 2024.

Ross Taylor, Marcin Kardas, Guillem Cucurull, Thomas Scialom, Anthony Hartshorn, Elvis Saravia, Andrew Poulton, Viktor Kerkez, and Robert Stojnic. Galactica: A large language model for science. *arXiv preprint arXiv:2211.09085*, 2022.

Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

Lorenzo Villaggi, James Stoddart, Adam Gaier, and David Benjamin. Tilegpt: Generative ai for intuitive design exploration and trade-offs navigation. In *Design Modelling Symposium Berlin*, pp. 241–252. Springer, 2024.

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`.

Ramon Elias Weber, Caitlin Mueller, and Christoph Reinhart. Automated floorplan generation in architectural design: A review of methods and applications. *Automation in Construction*, 140: 104385, 2022.

Chaoyi Wu, Weixiong Lin, Xiaoman Zhang, Ya Zhang, Weidi Xie, and Yanfeng Wang. Pmc-llama: toward building open-source language models for medicine. *Journal of the American Medical Informatics Association*, 31(9):1833–1843, 2024.

Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. Bloomberggpt: A large language model for finance. *arXiv preprint arXiv:2303.17564*, 2023.

Yong Xie, Karan Aggarwal, and Aitzaz Ahmad. Efficient continual pre-training for building domain specific large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 10184–10201, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Hongyang Yang, Xiao-Yang Liu, and Christina Dan Wang. Fingpt: Open-source financial large language models. *arXiv preprint arXiv:2306.06031*, 2023.

Jun Yin, Pengyu Zeng, Haoyuan Sun, Yuqin Dai, Han Zheng, Miao Zhang, Yachao Zhang, and Shuai Lu. Floorplan-llama: Aligning architects' feedback and domain knowledge in architectural floor plan generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6640–6662, 2025a.

Jun Yin, Pengyu Zeng, Jing Zhong, Peilin Li, Miao Zhang, Ran Luo, and Shuai Lu. Floorplan-deepseek (fpds): A multimodal approach to floorplan generation using vector-based next room prediction. *arXiv preprint arXiv:2506.21562*, 2025b.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *Advances in Neural Information Processing Systems*, 38:113222–113244, 2026.

Xiyuan Zhou, Xinlei Wang, Yirui He, Yang Wu, Ruixi Zou, Yuheng Cheng, Yulu Xie, Wenxuan Liu, Huan Zhao, Yan Xu, et al. Engibench: A benchmark for evaluating large language models on engineering problem solving. *arXiv preprint arXiv:2509.17677*, 2025.

A Additional Details on the Domain-Specific Language (DSL)

A.1 Overview

To represent building designs as structured text suitable for LLM training and generation, we developed a custom DSL and an accompanying parser library. The DSL encodes a building design as a collection of independent modalities, each covering a distinct architectural or structural aspect. Modalities are compact, keyword-delimited text blocks using integer coordinates (in quantized format) and enumerated vocabulary, designed to be token-efficient, human-readable, and directly consumable by LLMs as both input context and generation target. The library provides full roundtrip support; raw DSL text produced by a model can be parsed, structurally validated, semantically normalized, and re-emitted in canonical form, forming the backbone of both training data preparation and post-generation validation. The library is organized into three layers:

Grammar. Each modality is specified by a formal Parsing Expression Grammar (PEG) grammar written using TextX (Dejanović et al., 2017), a Python framework that automatically derives parsers and in-memory object models from grammar definitions. Grammars share coordinate and size primitives reused across modalities.

Parser. Each grammar has a corresponding parser class that loads the grammar at runtime, invokes TextX to parse input text, and post-processes the result into a typed domain model. Post-processing enforces polygon winding-order normalization and resolves symbolic cross-references (e.g., edge names referencing vertices by ID).

Model. Each modality has a typed Python dataclass that holds validated geometric and semantic data, performs structural integrity checks, supports JSON serialization with backward-compatible field aliasing, and can re-emit canonical DSL text.

A.2 Modality Reference

This section describes each modality, its purpose, and a minimal example.

Building and Structure. The *building* is the global metadata about the building: occupancy type, number of storeys, and a reference floor level with its elevation. This modality provides the high-level design intent and serves as the context header for a generation sequence. The *structure* specifies the primary structural material system. Together with building, this establishes the engineering premise of the design before any geometry is introduced.

Massing. The *massing* describes the 3D volumetric envelope as a polygonal footprint sharing a single floor-to-floor height.

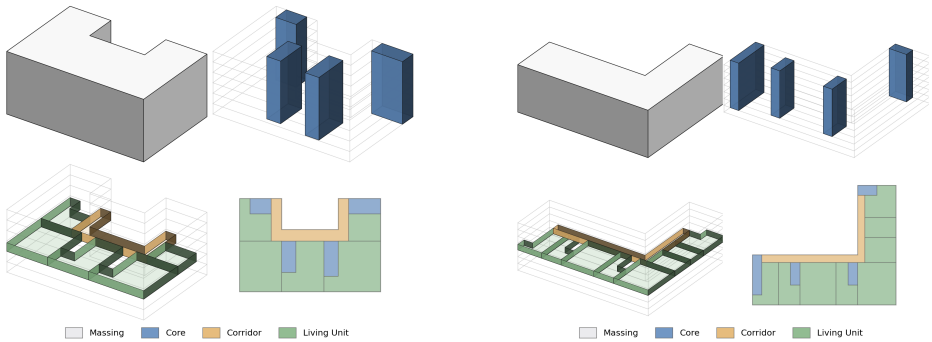
Space. The *space* represents the floor-plan geometry as a flat list of labeled polygons. Each polygon carries a semantic space type and a sequence of counter-clockwise 2D vertices. This is the simplest geometric representation, pure boundary coordinates with no explicit topology, making it the preferred format for floor-plan generation tasks. The supported space types are core, corridor, and living unit.

Space Graph. The *space graph* is the topological counterpart to spaces. In this representation, a floor plan is modeled as a graph: named vertices store 2D coordinates, named edges connect pairs of vertices through symbolic references, and spaces are defined as ordered traversals of edges. TextX cross-reference resolution enforces referential integrity by ensuring that every edge reference in a space boundary points to a declared edge and that every edge endpoint refers to a declared vertex.

Shared walls between adjacent rooms are represented as a single directed edge rather than duplicated overlapping boundaries, making the spacegraph the canonical representation for downstream structural and topological analysis. Since spaces are a simplified abstraction of the spacegraph, spaces are used to train the LLM, while spacegraphs are used for parsing and validation.

A.3 EXAMPLE DSL

Figure 11 and Listing 1 show examples of the DSL representation used for building layout generation. The DSL encodes each design using structured text blocks for building metadata, structural material, massing geometry, and labeled floor-plan spaces. In Listing 1, the `massing` block defines the exterior footprint and height, while the `spaces` block specifies the cores, corridor, and living units as labeled polygons. Figure 11 visualizes the same example geometrically, illustrating how the compact textual encoding maps directly to the massing and 2D floor-plan layout.



(a) (b)

Figure 11: Examples of DSL representation of multifamily residential buildings, illustrating the massing, vertical cores, living units, corridors, and corresponding 2D floor-plan layout.

Listing 1: Examples of DSL encodings of multifamily residential buildings, including building metadata, structural material, massing geometry, and floor-plan spaces.

```
# DSL for Example (a)
building { occupancy_type multifamily_residential storeys 6 level 2 elevation 60 }
structure { material wood_frame }
massing {
    height 30 polygon 0,0 416,0 416,275 291,275 291,183 124,183 124,275 0,275
}
spaces {
    polygon core 322,229 416,229 416,275 322,275
    polygon core 249,45 291,45 291,149 249,149
    polygon core 124,57 166,57 166,149 124,149
    polygon core 31,229 93,229 93,275 31,275
    polygon corridor 93,149 322,149 322,275 291,275 291,183 124,183 124,275 93,275
    polygon living_unit 322,149 416,149 416,229 322,229
    polygon living_unit 249,0 416,0 416,149 291,149 291,45 249,45
    polygon living_unit 124,0 249,0 249,149 166,149 166,57 124,57
    polygon living_unit 0,149 93,149 93,229 31,229 31,275 0,275
    polygon living_unit 0,0 124,0 124,149 0,149
}

# DSL for Example (b)
building { occupancy_type multifamily_residential storeys 7 level 3 elevation 90 }
structure { material reinforced_concrete }
massing {
    height 30 polygon 0,0 610,0 610,508 448,508 448,212 0,212
}
spaces {
    polygon core 0,42 40,42 40,212 0,212
    polygon core 162,84 203,84 203,180 162,180
    polygon core 407,84 448,84 448,180 407,180
    polygon core 448,466 529,466 529,508 448,508
    polygon corridor 40,180 478,180 478,466 448,466 448,212 40,212
    polygon living_unit 0,0 112,0 112,180 40,180 40,42 0,42
    polygon living_unit 112,0 203,0 203,84 162,84 162,180 112,180
    polygon living_unit 203,0 356,0 356,180 203,180
    polygon living_unit 356,0 448,0 448,84 407,84 407,180 356,180
    polygon living_unit 448,0 610,0 610,180 448,180
    polygon living_unit 478,180 610,180 610,286 478,286
    polygon living_unit 478,286 610,286 610,371 478,371
    polygon living_unit 478,371 610,371 610,508 529,508 529,466 478,466
}
```

A.4 Industry Foundation Classes (IFC)

Industry Foundation Classes (IFC) is an open industry standard file format for encoding and exchanging AEC BIM data (buildingSMART, 2024). The IFC specification defines a data model for encoding a complete BIM model, including: high-level object classes (like IfcWall, IfcSlab, and IfcSpace), non-geometric material and property values, and low-level geometry, like vectors, points, lines, polygons, meshes, and BRep solids.

IFC files are ASCII-based and utilize line number references to link components of an element definition, meaning the definition of an XYZ point referenced by the geometry defining a BIM element may involve tens or hundreds of intermediate references and span thousands of lines. IFC file size is also highly dependent on the modality of element geometry representation, where the mesh-modeled elements can add hundreds or thousands of lines just to define the vertex and face geometry compared to a geometrically equivalent but compact BRep representation. The verbosity and highly variable structure of IFC presents challenges to direct encoding with transformer architecture, and the multi-layer referential structure means a generated file might be syntactically correct but unparsable by BIM editing software.

B Additional Details on Data Generation and Processing

B.1 Data Generation

The synthetic corpus is built in four stages. Each stage targets a distributional limitation of the previous one, and every stage emits DSL in the canonical form, described in Appendix B.2. Stages 1 to 3 produce the layouts; stage 4 is applied last after deduplication and split generation.

B.1.1 Stage 1: Layouts Derived from TileGPT

TileGPT (Gaier et al., 2024; Villaggi et al., 2024) is a generative design system for tile-based architectural layouts. It uses MAP-Elites, a quality-diversity search algorithm, to produce a collection of high-performing layouts spanning user-defined design attributes, fine-tunes a language model on that collection, and refines generated conceptual layouts into constraint-compliant detailed layouts using Wave Function Collapse, a constraint-satisfaction procedure. The diversity and validity of the stage-1 layouts therefore derive from prior work.

Our contribution at this stage is the projection into the DSL. Each TileGPT layout is a three-dimensional array of tile identifiers with transformation flags, which we resolve against a tile catalog into a node-and-feature geometric representation: a deduplicated node set with tolerance-based corner merging, and labeled features referencing those nodes. We then consolidate collinear walls, resolve the labeled spaces into closed polygons, and emit the building, structure, massing, and spaces modalities. Because tiles meet on shared edges, this step must merge coincident vertices and eliminate degenerate segments; without it, polygons that are visually correct produce self-intersecting or zero-area geometry in the DSL.

B.1.2 Stage 2: Scale Augmentation

Stage-1 layouts inherit the tile module dimensions, so space sizes are drawn from a small discrete set. We rescale 94% of stage-1 layouts, drawing an independent scale factor for each of the two horizontal axes uniformly from $[0.604, 1.396]$, i.e. 1 ± 0.396. Because the two factors are drawn independently, the operation both changes overall size and perturbs aspect ratio, producing layouts whose massing and space dimensions vary continuously rather than over the discrete tile set. The bound is what keeps the result usable: beyond it, scaling yields living units and corridors whose proportions no longer correspond to buildable space.

B.1.3 Stage 3: Inference and Repair on Free-Form Massings

Stages 1 and 2 cannot produce massings outside the tile-based distribution, yet interactive use requires competence on free-form outlines. Stage 3 closes this gap by generating massings directly, asking an intermediate model to fill them, and repairing the outputs that are close to correct.

Massing Sampling. We sample rectangles, squares, and L-shapes, each in orthogonal and non-orthogonal variants. Rather than drawing shape parameters independently, we enumerate a full-factorial grid over the parameters of each family (width and height, and for L-shapes the notch proportion in each axis and the notch corner) and select a subset of grid points for generation, which gives systematic coverage of the parameter space rather than the clustering that independent uniform sampling produces. L-shape notch proportions are restricted to $[0.4, 0.65]$ of each axis, which keeps both wings thick enough to contain habitable space. Non-orthogonal variants are produced by perturbing each vertex independently with probability 0.7 by an offset of up to 20–25 grid units, with any perturbation rejected if it would duplicate an existing vertex. Rectangles and squares span 100–550 grid units (10–$55\mathrm{m}$) per dimension and L-shapes 150–550 grid units (15–$55\mathrm{m}$).

Candidate Generation. For each sampled massing we query an intermediate checkpoint of our own model for a spaces completion.

Repair Operator. Intermediate-model completions on unfamiliar massings are typically close to valid but leave small uncovered slivers, overshoot the outline, or omit a space. The repair operator converts the completion to a layout graph, adjusts it to fit the massing, and converts it back to DSL. It proceeds in a fixed order: remove spaces lying entirely outside the outline; clip spaces crossing it; divide large interior unused areas proportionally among adjacent spaces; insert new living units where the remaining free area admits them; iteratively extend spaces toward the outline; and finally merge any leftover small unused areas into adjacent spaces. Both iterative phases stop when the layout passes the admission checks below, when an iteration fails to improve coverage, or at a bounded iteration count. Operating on a graph rather than on independent polygons means shared boundaries between adjacent spaces move together; in addition, each merge or extension is tested against neighbouring spaces beforehand and skipped if it would produce a significant overlap.

Admission Verifier. A repaired layout enters the corpus only if it satisfies all of the following, evaluated on the repaired geometry with $\epsilon = 10^{-3}$:

1. *Coverage.* The intersection-over-union of the union of spaces with the massing is within ϵ of 1, and the ratio of total space area to massing area does not exceed $1 + \epsilon$.
2. *Composition.* At least one core, at least one corridor, and at least one living unit are present.
3. *Shape quality.* No living unit or core contains a narrow neck, detected by morphological erosion after simplification. This rejects spaces that are nominally large enough but not usable.
4. *Fidelity to the model output.* Relative to the pre-repair completion, no corridor or core loses more than 20% of its area and no living unit loses more than 50%. This prevents the repair operator from silently replacing the model's design with its own.

Layouts failing any check are discarded rather than corrected further.

This verifier is a formal specification of geometric and functional correctness for multifamily layouts, and it is the same specification that the verifiable rewards in Section 6.2 implement as a reward signal. We regard this as a design decision rather than an incidental overlap: a single definition of correctness is applied consistently when admitting training data and when scoring policy rollouts. It does mean that verifier-based metrics on synthetic data are not independent evidence. Every sample in the corpus passes the coverage filter of Appendix B.2.2, and stage-3 samples additionally pass the composition and shape checks above, so in-distribution base performance is correspondingly high: our 0.6B base model reaches 90.2% pass@1 on the synthetic test set, which measures how well it has internalized a specification it was trained toward rather than performance against an independently chosen metric. The informative measurement is on the real-world footprints of Section 4.2, which are used neither to generate nor to filter training data. There the same model reaches 23.8% pass@1 (Appendix H.1), and the post-training gains reported in Section 7 are measured from that baseline.

B.1.4 Stage 4: Rotation Augmentation

Rotation augmentation is applied last, after the filtering, deduplication, and splitting described in Appendix B.2, so that no rotated variant of a training layout can reach the validation or test split.

Writing the angle distribution of rotation augmentation as a mixture,

$$\{\theta_i\}_{i=1}^{20} \sim 4\ \mathrm{Uniform}(0^\circ, 10^\circ) + 12\ \mathrm{Uniform}(10^\circ, 350^\circ) + 4\ \mathrm{Uniform}(350^\circ, 360^\circ),$$

the two narrow components supply near-identity perturbations, which teach insensitivity to small orientation changes, while the wide component supplies large reorientations. Rotated coordinates are re-quantized, and a rotated variant is discarded if any coordinate falls outside the representable range, so a layout near the extent limit contributes fewer than 20 variants. The same procedure is used for the human-feedback data (Appendix C.3).

B.1.5 Dataset Composition

Table 1: Synthetic corpus composition by generation stage. Counts are before rotation augmentation unless noted.

Stage	Description	Samples	Share of corpus
1,2	TileGPT-derived layouts with scale jitter	3.9M	95%
3	Inference + repair (admitted)	0.2M	5%
Total before rotation augmentation		4.1M	100%
Total after rotation augmentation		82M	

B.2 Data Processing

B.2.1 Canonicalization

A floor plan has many DSL encodings that carry the same design information: the vertex sequence may start anywhere in the cycle, spaces may be listed in any order, and the sample may sit anywhere in the coordinate range, which is immaterial because the DSL does not model site position. Winding order is already normalized by the parser (Appendix A.1), which emits outer boundaries counter-clockwise and hole boundaries clockwise so that orientation alone distinguishes a boundary from an interior void. The remaining freedom is removed as follows.

Start Vertex. The vertex sequence is rotated to begin at the vertex of least Euclidean distance from the origin. Line-like elements are ordered so that the endpoint nearer the origin comes first, with ties broken by smaller x and then smaller y.

Translation. The sample is translated so that the minimum x and minimum y taken jointly over all planar modalities become zero. The shift is a single integer vector applied uniformly, so relative geometry is unchanged and the sample occupies the low corner of the quantized range.

Space Ordering. Space polygons are sorted by label and then by their vertex string, giving a deterministic order independent of the order in which the generator emitted them.

Canonicalization runs on training data and on model inputs at inference time. Two properties follow: textually identical samples are geometrically identical, which makes hash-based deduplication exact; and the model is never asked to spend capacity distinguishing encodings that denote the same building.

B.2.2 Deduplication and Split Construction

Quality Filtering. Samples are first scored and filtered on two criteria: any coordinate outside the representable range, and a coverage score combining the intersection-over-union of the space union with the massing and the ratio of total space area to massing area. Samples scoring at or below the threshold are discarded.

Canonical Hash. For each surviving sample we form the string consisting of its canonicalized spaces text and its massing text, and hash it with BLAKE2b truncated to 128 bits. Collision probability is negligible at corpus scale. The hash covers geometry only: the building and structure

modalities are excluded, so two layouts with identical geometry but different structural material are treated as duplicates and one is retained. This is intentional, since geometry is the property we deduplicate on, but it does mean the corpus contains no pairs that differ solely in structural material.

Within-Split Deduplication. Samples are streamed in chunks and the first occurrence of each hash is retained.

Cross-Split Overlap Removal. Deduplication within a split does not prevent the same layout appearing in two splits. We therefore treat the training split as the reference and remove from validation and test any sample whose hash occurs in training.

Ordering. Rotation augmentation is applied after splitting, so a footprint present in training cannot reach evaluation as a rotated variant, and cross-split overlap removal ensures it cannot reach evaluation as an exact duplicate either.

B.3 t-SNE Visualization of Massing Geometry

To compare the geometric distributions of the synthetic, OSM, and SFT (described in Section 6.1 and Appendix C) datasets, we project their massing footprints into a shared 2D embedding using t-distributed stochastic neighbor embedding (t-SNE) (Van der Maaten & Hinton, 2008). The analysis assesses whether the datasets occupy overlapping or distinct regions of footprint shape space. The procedure consists of two stages: shape descriptor construction and dimensionality reduction. Notably, this visualization is computed on the datasets before rotation augmentation is applied.

Shape Descriptor Construction. Raw polygon coordinates are not directly comparable across samples because footprints vary in absolute location, scale, and vertex count. We therefore convert each footprint into a normalized fixed-length contour descriptor. First, we translate the polygon to center the footprint at the origin. Second, we normalize scale by dividing all coordinates by $\sqrt{A}$, where A is the polygon area after centering, so that the normalized polygon has unit area. Third, we resample the exterior boundary at uniform arc-length intervals. Given a normalized polygon with perimeter length L, we sample N boundary points at distances:

$$d_i = \frac{iL}{N}, \qquad i = 0, 1, \ldots, N-1. \tag{2}$$

The resulting coordinates $\{(x_i, y_i)\}_{i=0}^{N-1}$ are concatenated into a $2N$-dimensional feature vector. This descriptor is invariant to translation and scale. We do not normalize rotation, since orientation and axis alignment are meaningful geometric properties of constructed building footprints. Notably, the visualization is done on the datasets prior to rotation augmentations.

t-SNE Embedding. Feature vectors from all datasets are stacked into a single matrix:

$$X \in \mathbb{R}^{M \times 2N}, \tag{3}$$

where M is the total number of sampled footprints. We then apply t-SNE to obtain a 2D embedding. The embedding is initialized with PCA to improve stability and reduce sensitivity to random initialization. Table 2 summarizes the hyperparameters used in the analysis.

Table 2: Hyperparameters used for the t-SNE analysis of massing footprints.

Parameter	Value
Boundary sample points N	64
Maximum samples per dataset	10,000
t-SNE perplexity	40
t-SNE iterations	1,000
t-SNE initialization	PCA

Interpretation. The resulting embedding, presented in Figure 4a, contains several smooth, approximately 1D structures. These arise naturally from simple low-complexity footprints. For example, after translation and area normalization, a rectangular footprint is largely parameterized by a single geometric degree of freedom, its aspect ratio $r = W/H$. As r varies, the uniformly sampled boundary points change continuously along the four edges of the rectangle, tracing a smooth 1D manifold in the high-dimensional descriptor space. t-SNE preserves this local structure, causing rectangular footprints to appear as visible lines or arcs in the 2D embedding.

Similarly, L-shaped, T-shaped, and other low-vertex footprints form lower-dimensional submanifolds that appear as shorter curves or branches. In contrast, irregular many-vertex footprints, which are more common in the OSM data, do not follow the same low-complexity structure and are distributed more broadly across the embedding. This pattern supports the interpretation that real-world OSM massings occupy regions of shape space that are only partially covered by the procedurally generated synthetic dataset.

B.4 Data Samples

Figures 12 to 14 show representative samples from the OpenStreetMap (OSM), synthetic, and SFT datasets. Synthetic samples include procedurally generated space layouts, consisting of cores, corridors, and living units, in addition to the building massing and metadata. In contrast, OSM samples contain only the building massing and associated metadata, without ground-truth space layouts. The SFT samples, described in detail in Appendix C, are drawn from a curated set of building footprints and include architect-edited layouts. These edited layouts consist of living units and, depending on the massing scale and design requirements, may also include cores and corridors.

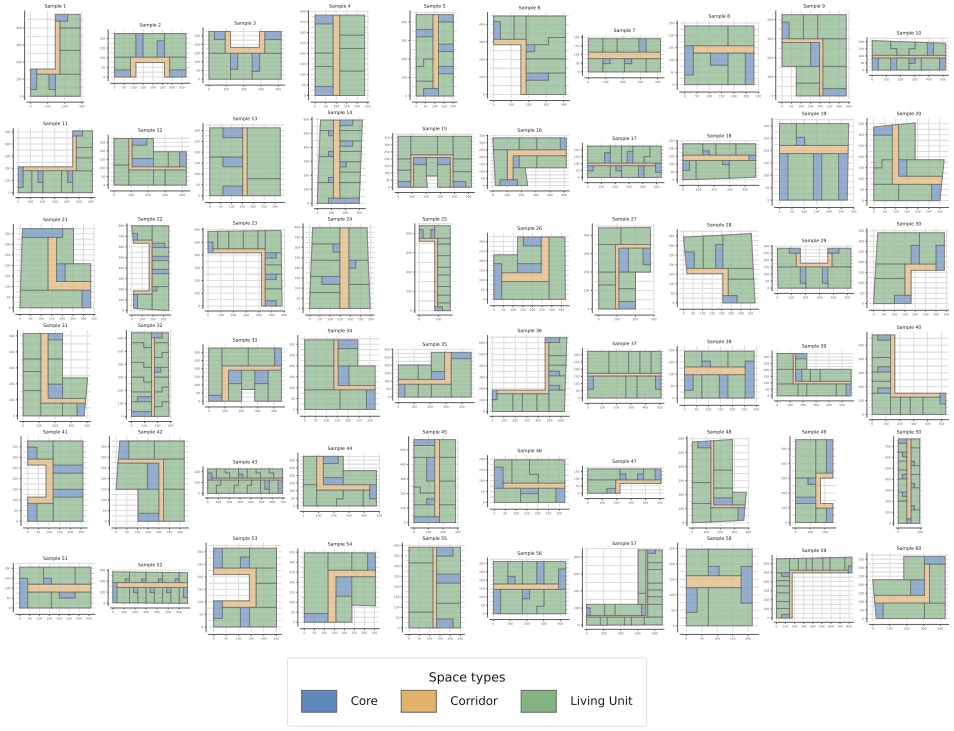


Figure 12: Samples from the **synthetic** dataset. Each sample contains the building massing, metadata, and a procedurally generated space layout consisting of cores, corridors, and living units.

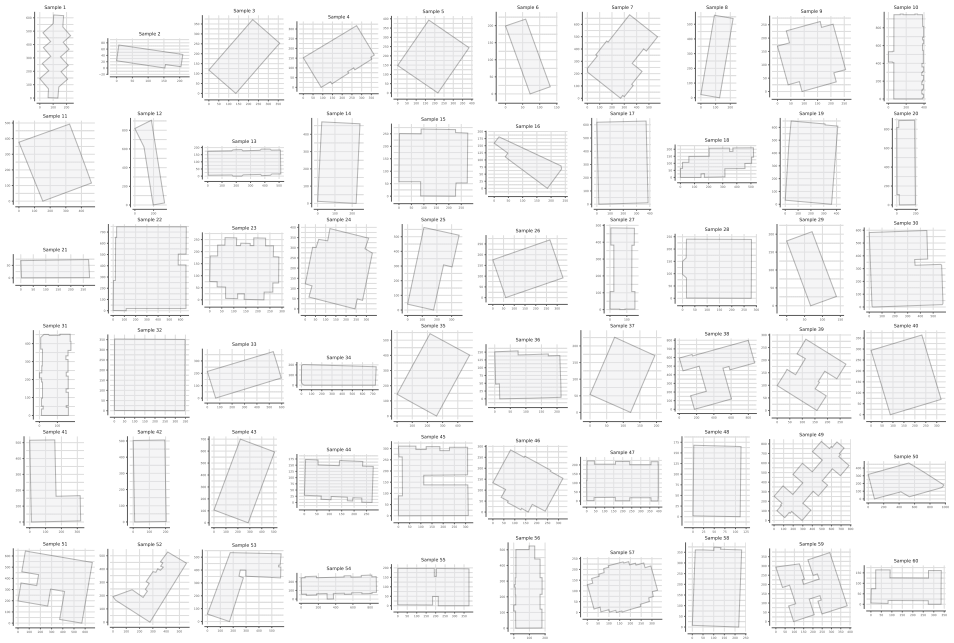


Figure 13: Samples from the **OSM** dataset. Each sample contains only the building massing and metadata, without ground-truth space layouts, and therefore cannot be used for pre-training.



Figure 14: Samples from the **SFT** dataset. Each sample contains the building massing, metadata, and an architect-edited space layout consisting of living units and, when needed, cores and corridors.

C Additional Details on the Human Feedback Data

C.1 Labeler Information

The labelers were 10 practicing architects hired through a BIM consultancy contractor. They were instructed to evaluate and edit model-generated outputs according to architectural conventions commonly observed in North America.

C.2 Labeling Instructions and Interface

The instructions provided to the labelers are summarized below. In short, a massing is first sampled from a curated dataset, after which four model-generated layouts are produced. Each layout is independently evaluated by the labeler according to a predefined set of criteria. The layouts are subsequently ranked relative to one another, with ties allowed. Finally, the highest-ranked layout is edited by the labeler to correct any remaining issues. Figures 15 to 17 show screenshots of our labeling interface.

Instructions Provided to Labelers

Introduction
You are given a randomly generated prompt that includes the following inputs:

- Building Type: Multifamily residential
- Structure: Either wood frame or concrete frame
- Massing: A length and width to define a massing envelope

The LLM will generate four floor plan layouts based on this prompt. It can generate and arrange the following types of spaces to fit within the mass envelope:

- Cores: The model does not currently differentiate between elevators and stairs. Use your judgement to evaluate and edit the cores assuming they contain either stairs alone, elevators alone, or a combination of both depending on the layout.
- Corridors: Corridors should be between 1m (3ft) and 2m (6ft)
- Living Units: Use this as a guide for a "good" layout of living units:
  - Studio/1-bedroom: 30–60 m² (320–650 ft²)
  - 2-bedroom: 60–90 m² (650–970 ft²)
  - 3-bedroom: 90–120 m² (970–1,300 ft²)

Step 1: Rating Each of the Four Layouts
For each of the four generated layouts, provide:

- An overall rating from 5 (highest) to 1 (lowest)
- Responses to a set of yes/no evaluation questions

Step 1a: Rate Each Layout (Scale 1–5)
Rate each layout using the following scale:

- 5: Architecturally appropriate, geometrically perfect, fully labeled, ready to use
- 4: Good quality with minor issues that are easily correctable (e.g., fixable in approximately 1 minute)
- 3: Moderate issues requiring some corrections, but salvageable (e.g., fixable in 2–3 minutes)
- 2: Many issues requiring significant corrections, but salvageable (e.g., fixable in 4–5 minutes)
- 1: Major architectural or geometric flaws; may not be worth correcting (e.g., would require more than 5 minutes to fix)
- Nothing Generated: The layout is completely empty

Criteria for Rating

- Architectural Correctness: The floor plan should support well-proportioned living spaces and sensible circulation.

- Usability and Clarity: Layouts should be practical and clearly labeled.
- Completeness: All required elements should be present (cores, corridors, living spaces).
- Core Placement and Size: Cores should be appropriately placed and sized, without redundancy.
- Corridor Placement: Corridors should provide proper connectivity to all spaces.
- Geometric Correctness: No overlapping spaces or empty areas within the massing. Spaces highlighted in red indicate geometric errors and should negatively impact the rating.

Step 1b: Individual Criteria (Yes/No Questions)
Answer the following questions to the best of your ability:

- All areas inside the massing are filled with spaces.
- Living spaces are well distributed and well proportioned.
- A useful corridor exists and connects to all living spaces.
- Cores are placed appropriately.
- There are an appropriate number of cores.
- One or more spaces are labeled as "Uncategorized".
- Self-intersecting spaces exist.
- Overlapping spaces exist.
- Spaces are partially placed outside of the massing.

Add any additional notes that may help explain your evaluation decisions (optional).

Step 2: Rank Layouts
Rank all layouts from best to worst based on your evaluation. Multiple layouts may be assigned the same rank, and some ranking slots may remain empty.

Ranking Guidelines

- Prioritize geometric and architectural correctness over minor labeling issues.
- A layout with one major but fixable issue may rank higher than one with many minor unfixable issues.
- A layout with correct geometry but incorrect labels is more valuable than one with correct labels but overlapping rooms.
- Consider which layout would be most useful to an architect beginning the design process with the fewest required edits.
- Incorrect layouts can lead to construction errors, wasted materials, or safety concerns; prioritize correctness over creativity.
- When uncertain, ask: "Which layout would I rather receive if I needed to create a working floor plan today?"

Step 3: Edit the Best Layout
You will be presented with the highest-rated layout based on the ratings and rankings you provided. Using the simplified editor, make corrections according to the following guidelines.

Editor Capabilities

- Draw a wall to separate one space into two
- Select and edit a wall
- Delete a wall
- Drag wall endpoints
- Modify space labels

Editing Principles

- Be diligent and accurate; floor plan errors can have real-world consequences.
- Make precise modifications; small dimensional changes may be important.
- Prioritize corrections that affect structural integrity and usability.

High-Priority Editing Objectives (in descending priority order)

1. Fix Geometric Issues (Highest Priority)

- Fix spaces that extend outside the massing.
- Ensure all spaces completely fill the massing.
- Eliminate overlapping polygons.
- Remove or merge tiny polygons.
- Resolve self-intersecting polygons.

2. Fix Critical Architectural Issues

- Correct core placement and sizing.
- Ensure corridor connectivity to all living spaces.
- Adjust living space dimensions if severely incorrect.

3. Fix Room Labels

- Ensure all spaces are correctly labeled.
- Verify labels match the intended space function.

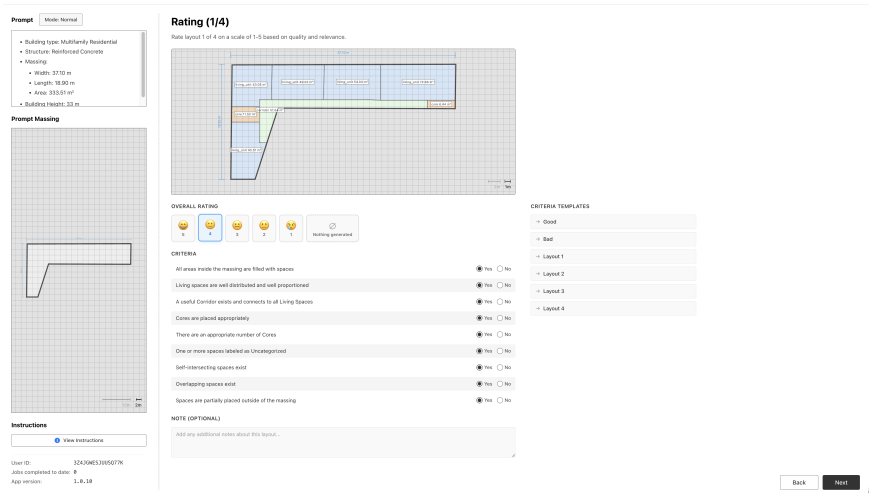


Figure 15: Labelers independently evaluate each model generation according to the criteria.

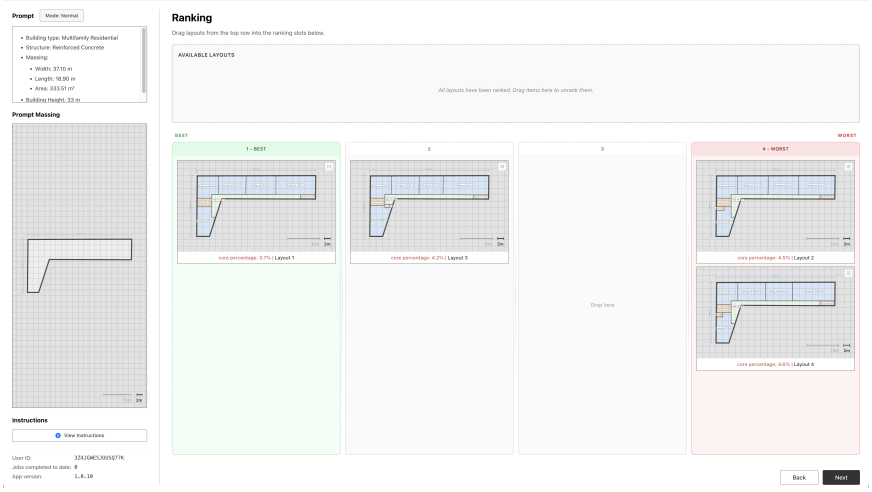


Figure 16: Model-generated outputs are ranked relative to one another, with ties permitted.

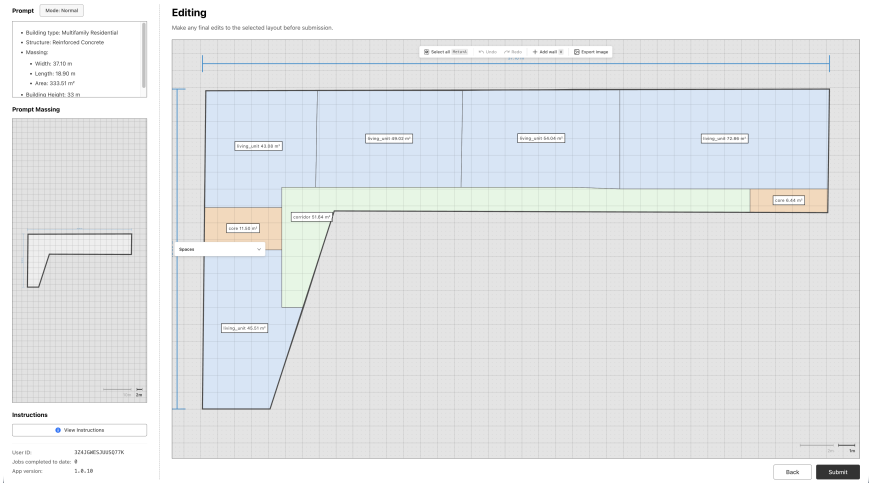


Figure 17: The highest-ranked model-generated output is subsequently edited by the labeler.

C.3 Human Feedback Data Processing

Human feedback data is collected through annotation sessions in which labelers evaluate 4 model-generated building layouts for a given prompt. For each candidate layout k, the labeler provides:

1. An ordinal rank $r_k \in \{1, \ldots, 4\}$, where lower rank is better.
2. A scalar rating $s_k \in [1, 5]$.
3. A binary checklist of design and geometry criteria.
4. A human-edited layout.

Basic Quality Filtering. Before constructing any training dataset, we apply several deterministic filters. First, we remove skipped annotation sessions and any candidate layout with missing or sentinel labels, such as a rank or rating of -1 or `null`. Second, each candidate layout is converted from its structured JSON representation into our compact DSL representation using the DSL encoder. Candidates whose DSL serialization fails or exceeds a certain wall-clock timeout are removed. Any preference pair or training example depending on a failed serialization is also removed.

Finally, we remove duplicate preference pairs. In particular, if the chosen and rejected layouts in a pair produce identical DSL strings after serialization, the pair is discarded, since it provides no useful preference signal.

Rank and Rating Consistency. Each annotation contains both a rank and a rating. These two signals should generally agree, but noisy annotations may contain contradictions. For example, a layout may be ranked above another layout while receiving a lower rating. To reduce such noise, we keep only pairs whose rank order is consistent with their ratings.

For two candidate layouts a and b, with ranks r_a, r_b and ratings s_a, s_b, we define

$$\text{consistent}(a, b) = \begin{cases} s_a \geq s_b, & \text{if } r_a < r_b, \\ s_b \geq s_a, & \text{if } r_a > r_b, \\ \top, & \text{if } r_a = r_b. \end{cases}$$

Thus, if layout a is ranked better than layout b, then its rating must be at least as high as the rating of b. Ties in rank are always treated as consistent. This check is applied to all pairwise combinations within each annotation session. Pairs that fail the check are excluded from preference training.

Geometric Filtering of Edited Completions. Human-edited layouts are used as SFT targets, but they may still contain geometric errors. We therefore filter edited completions using an automated test based on the geometric correctness verifier, described in Section 6.2 and Appendix E.

Rotation Augmentation. Architectural layouts are invariant to planar rotation; i.e., rotating a valid layout usually preserves its design structure and geometric validity. We use this property to augment the data utilizing the same approach used for the pre-training dataset, described in Section 4.

For each training example, we generate rotated variants by sampling an angle θ from three regimes:

$$\{\theta_i\}_{i=1}^{12} \sim 4\ \mathrm{Uniform}(0^\circ, 10^\circ) + 4\ \mathrm{Uniform}(10^\circ, 350^\circ) + 4\ \mathrm{Uniform}(350^\circ, 360^\circ),$$

This produces both small near-identity perturbations and large rotations. If any rotated coordinate falls outside the tokenizer coordinate range of 1024 grid units, the augmented example is discarded. Therefore, each training example produces at most $3N_{\mathrm{aug}}$ rotated variants, corresponding to a nominal training-set expansion factor of $3N_{\mathrm{aug}} + 1$, where N_{aug} was set to 4.

Table 3 summarizes the number of datapoints removed during human-feedback data filtering. After this filtering step, Table 4 summarizes the size of the remaining human-feedback datasets before and after rotation augmentation.

We reserve 10% of the filtered data for validation. To prevent information leakage, the split is performed at the annotation-session level, so that all candidate layouts and pairwise preferences associated with the same prompt are assigned entirely to either the training or validation split. Training and validation examples are augmented separately so that rotated variants of the same example do not appear across splits.

Table 3: Number of human feedback datapoints removed during data filtering and processing.

Category	Reason	Count
Skipped	Rating -1	61
Skipped	Inconsistent pairs	575
Skipped	Failed geometric checks	287
Errors	Failed DSL encodings	227
Total		**1,150**

Table 4: Number of datapoints in the human-feedback dataset after filtering, before and after rotation augmentation.

Dataset	Before Aug.	After Aug.
SFT completions	10.4k	135.4k
Pairwise preferences	90.6k	1.2M

D Additional Details on the Training Setup

D.1 Pre-training Settings and Hyperparameters

Tokenization Efficiency. To improve token efficiency, we use a tokenizer specialized for the DSL representation. Its vocabulary is composed primarily of DSL keywords, delimiters, and quantized numeric tokens, which better match the structure of the generated architectural sequences. As shown in Figure 4b, the DSL tokenizer substantially shortens the sequence-length distribution compared with the original Qwen3 tokenizer. The statistics in Table 5 quantify this effect, showing reductions in the mean, median, 95th-percentile, and total token counts. This reduction is especially important after augmentation, where the total token budget decreases from 38.5B tokens with the Qwen3 tokenizer to 17.6B tokens with the DSL tokenizer.

Effective Parameter Counts. Table 6 reports the effective parameter counts after resizing the embedding layers to match the DSL tokenizer vocabulary. The tokenizer contains 1367 tokens and is resized to 1408, the nearest multiple of 64, to improve embedding efficiency on modern hardware.

Pre-Training. All models are optimized using AdamW (Kingma & Ba, 2014; Loshchilov & Hutter, 2017) with a weight decay of 0.01 and a cosine learning rate schedule with 4000 warmup steps

Table 5: Tokenization statistics by tokenizer. "Before Aug." and "After Aug." denote total token counts before and after augmentation.

Tokenizer	Mean	Median	P95	Max	Before Aug.	After Aug.
DSL Tokenizer	235.9	219.0	373.0	785	880M	17.6B
Qwen3 Tokenizer	516.6	475.0	844.0	1,902	1.9B	38.5B

and a minimum learning rate of 1e-6. We maintain a fixed effective batch size of 1024 sequences across all experiments, while tuning the learning rate separately for each model size, as shown in Table 6. Sequence lengths are dynamically varied during training and padded to the maximum sequence length within each batch. All models are trained for a maximum of 200k optimizer steps, with the best checkpoint selected based on validation loss. Pretraining takes approximately three epochs with this effective batch size.

Training is performed with PyTorch (Li et al., 2020) in bfloat16 mixed precision using FlashAttention-2 (Dao, 2024), gradient checkpointing (Chen et al., 2016), and Liger kernels (Hsu et al., 2024) to improve memory efficiency and throughput. All experiments were conducted on NVIDIA H100 GPUs using either Distributed Data Parallel (DDP) (Li et al., 2020) or DeepSpeed (Rajbhandari et al., 2020; Rasley et al., 2020), depending on model scale, with the number of GPUs adjusted to maintain a consistent effective batch size across experiments.

Table 6: Effective parameter counts after resizing the embedding layers to match the DSL tokenizer vocabulary, and peak learning rates used during pre-training.

Base Model	Effective Params	Learning Rate
Qwen3-0.6B	440M	3e-4
Qwen3-1.7B	1.4B	4e-4
Pythia-70M	20M	2e-4
Pythia-160M	90M	6e-5[1]
Pythia-410M	310M	3e-4
Pythia-1.4B	1.2B	3e-4

D.2 Post-training Settings and Hyperparameters

Our post-training setup is based on TRL (von Werra et al., 2020) and PyTorch (Paszke et al., 2019). Across all post-training stages, we use full-parameter fine-tuning with AdamW optimization (Kingma & Ba, 2014; Loshchilov & Hutter, 2017) with a cosine learning rate schedule, full bfloat16 precision, and gradient checkpointing (Chen et al., 2016). All experiments were conducted on NVIDIA H100 GPUs using Distributed Data Parallel (DDP) (Li et al., 2020). Within each post-training stage, the effective batch size is held constant across model scales, while learning rates are tuned separately for each model and stage. The resulting hyperparameters are reported in Table 7.

Supervised Fine-Tuning. We perform supervised fine-tuning (SFT) by fine-tuning all model parameters from the pretrained base checkpoint. SFT uses a 10% warmup ratio and no weight decay. We train for 2k optimization steps with an effective batch size of 256.

Reward Modeling. We train the reward model (RM) by fine-tuning all parameters of the SFT checkpoint on pairwise preference data. The RM warmup ratio is 10%, and weight decay is set to 0.01. We train for 5k optimization steps with an effective batch size of 256.

GRPO. We fully fine-tune the SFT model using GRPO with verifiable DSL rewards and the RM. GRPO uses 500 warmup steps, no weight decay, and a maximum gradient norm of 1.0. We train for 5k optimization steps with an effective batch size of 256. The coefficients for the reward model and verifiable rewards are set to $\lambda_{\text{RM}} = \lambda_i = 1$, and the reward model normalization scale in Equation (1) is set to $\alpha = 3$.

[1] A lower learning rate was used because training was unstable at higher learning rates.

For each prompt, we sample 8 completions with temperature 1.0, top-p 0.95, and a maximum completion length of 2048 tokens. The KL coefficient is set to 0; in our experiments, we observed better stability without the KL term. This also removes the need to load a reference model, enabling a larger effective batch size. Generation is accelerated with collocated vLLM (Kwon et al., 2023).

Table 7: Peak learning rates used during post-training.

Model	SFT	RM	GRPO
Qwen3-0.6B	2e-4	8e-5	6e-5
Qwen3-1.7B	2e-4	3e-4[2]	8e-5
Pythia-410M	2e-4	8e-5	4e-5
Pythia-1.4B	2e-4	8e-5	4e-5

E Additional Details on the Verifiable Rewards

As outlined in Section 6.2, human feedback alone is insufficient to guarantee adherence to specific design requirements. We therefore employ verifiable rewards to automatically evaluate and enforce geometric and functional constraints. Here, we describe the verifiable rewards in more detail.

Unless otherwise stated, continuous constraint violations are converted to rewards using a smoothstep function. For a normalized error $e \in [0, 1]$ and tolerance τ, we define

$$S_\tau(e) = \begin{cases} 1, & e \le \tau, \\ \exp\left(-\frac{e-\tau}{\tau}\right), & e > \tau. \end{cases} \tag{4}$$

This mapping provides full reward within a small tolerance around the desired constraint and decays exponentially once the tolerance is exceeded. All individual reward components are bounded to $[0, 1]$.

E.1 Geometric Correctness

Geometric correctness evaluates whether the generated space polygons form a valid partition of the input building massing. We compute three spatial consistency signals from the parsed polygon geometries:

1. **Containment.** Generated spaces should lie within the building massing. The verifier computes the normalized fraction of space geometry outside the massing, denoted e_{out}.
2. **Non-overlap.** Generated spaces should not overlap one another. The verifier computes the normalized overlap violation e_{ov} across the generated spaces.
3. **Massing coverage.** The generated spaces should collectively occupy the complete building footprint. Let $c_{\text{mass}} \in [0, 1]$ denote the fraction of the massing covered by the union of the generated spaces. We define the coverage error as $e_{\text{cov}} = 1 - c_{\text{mass}}$. Coverage is rewarded only when the containment constraint is satisfied within tolerance. This prevents a layout from obtaining a high coverage score by extending spaces beyond the massing boundary.

For geometric correctness, we use a tolerance of $\tau_{\text{geom}} = 0.01$. The three component rewards are therefore

$$r_{\text{contain}} = S_{\tau_{\text{geom}}}(e_{\text{out}}), \tag{5}$$

$$r_{\text{overlap}} = S_{\tau_{\text{geom}}}(e_{\text{ov}}), \tag{6}$$

$$r_{\text{coverage}} = \begin{cases} S_{\tau_{\text{geom}}}(e_{\text{cov}}), & e_{\text{out}} \le \tau_{\text{geom}}, \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

The final geometry reward is computed as the geometric mean of the three component scores,

$$R_{\text{geom}} = \text{GM}\left(r_{\text{contain}}, r_{\text{overlap}}, r_{\text{coverage}}\right), \tag{8}$$

[2] A higher learning rate was used because the model was under-trained at lower learning rates.

where

$$\mathrm{GM}(r_1, \ldots, r_n) = \left(\prod_{i=1}^{n} \max(r_i, \varepsilon) \right)^{1/n}. \tag{9}$$

In practice, the geometric mean is computed in log space using $\exp(\frac{1}{n} \sum_i \log(\max(r_i, \varepsilon)))$, with $\varepsilon = 10^{-8}$ for numerical stability. Unlike an arithmetic mean, this aggregation prevents a high score on one constraint from compensating for a severe violation of another.

Outputs that cannot be successfully parsed, geometrically evaluated, or that contain invalid polygons such as self-intersecting or degenerate geometries receive zero reward.

E.2 Functional Compliance

Functional compliance evaluates residential design requirements using the parsed space labels, counts, and polygon areas. We consider four criteria:

1. **Core count.** The number of vertical circulation cores must fall between one and four.
2. **Corridor count.** The number of corridors must fall between one and three.
3. **Unit size.** At least 95% of living units must satisfy the minimum area requirement of $A_{\min} = 30$ square meters.
4. **Unit labeling.** At least 95% of living units must satisfy the expected labeling requirement.

As with geometric correctness, each criterion is mapped to a continuous score in $[0, 1]$. We use a smooth tolerance of $\tau_{\text{func}} = 0.05$, providing full reward when a requirement is satisfied and exponentially decreasing reward as the violation increases.

For count-based constraints with an acceptable interval $[\ell, u]$, we define the normalized violation as

$$e_{\text{range}}(x; \ell, u) = \begin{cases} \dfrac{\ell - x}{\max(\ell, 1)}, & x < \ell, \\ 0, & \ell \le x \le u, \\ \dfrac{x - u}{\max(u, 1)}, & x > u. \end{cases} \tag{10}$$

The corresponding core and corridor rewards are

$$r_{\text{core}} = S_{\tau_{\text{func}}} \left(e_{\text{range}}(n_{\text{core}}; 1, 4) \right), \tag{11}$$

$$r_{\text{corridor}} = S_{\tau_{\text{func}}} \left(e_{\text{range}}(n_{\text{corridor}}; 1, 3) \right). \tag{12}$$

For the unit-size and labeling requirements, let p_{size} denote the fraction of living units satisfying the minimum area requirement and p_{label} the fraction satisfying the labeling requirement. Their normalized violations are

$$e_{\text{size}} = \max\left(0, 0.95 - p_{\text{size}}\right), \tag{13}$$

$$e_{\text{label}} = \max\left(0, 0.95 - p_{\text{label}}\right), \tag{14}$$

with corresponding rewards

$$r_{\text{size}} = S_{\tau_{\text{func}}}(e_{\text{size}}), \tag{15}$$

$$r_{\text{label}} = S_{\tau_{\text{func}}}(e_{\text{label}}). \tag{16}$$

The final functional compliance reward is computed as the geometric mean of the four components:

$$R_{\text{func}} = \mathrm{GM}\left(r_{\text{core}}, r_{\text{corridor}}, r_{\text{size}}, r_{\text{label}}\right). \tag{17}$$

This aggregation ensures that deficiencies in any individual requirement substantially reduce the overall functional reward. Outputs that cannot be successfully parsed, evaluated, or that contain invalid polygon geometries receive zero reward.

E.3 Soft Overlong Penalty

To discourage excessively long completions and improve the stability of training, we apply a length-based penalty adapted from DAPO (Yu et al., 2026) and implemented in TRL (von Werra et al., 2020). Let $|y|$ denote the completion length, $L_{\max}$ the maximum allowed completion length, and L_{cache} a soft penalty window. The reward is defined as:

$$R_{\text{length}}(y) = \begin{cases} 0, & |y| \le L_{\max} - L_{\text{cache}}, \\ \dfrac{(L_{\max} - L_{\text{cache}}) - |y|}{L_{\text{cache}}}, & L_{\max} - L_{\text{cache}} < |y| \le L_{\max}, \\ -1, & |y| > L_{\max}. \end{cases} \tag{18}$$

This reward does not incentivize shorter completions. Instead, it imposes a progressively stronger penalty as the completion length approaches the maximum budget and assigns the maximum penalty once the budget is exceeded.

F Additional Details on the VLM Judge

We evaluate perceptual and architectural layout quality using a pairwise VLM judge protocol. For each evaluation prompt, two candidate completions are generated from the same input: one from model A and one from model B. The judge compares the corresponding rendered floor plans and assigns one of three outcomes: A wins, B wins, or tie. Outcomes are then aggregated across prompts to obtain strict win, loss, and tie rates.

F.1 VLM Judge Implementation

Pair Construction. For each prompt, we select one completion from each model and render both outputs as 784×784 pixel floor-plan images using the same deterministic DSL parser and renderer used during training and evaluation. This ensures that the VLM judge observes only the geometric and semantic content induced by the generated DSL.

Judgment Protocol. The two rendered floor plans are provided to a vision-language model in a single comparison prompt. The images are labeled as *Floorplan A* and *Floorplan B*, giving the judge unambiguous references that are independent of their visual position in the prompt. The judge is instructed to evaluate the layouts using the structured prompt shown below. The prompt asks the judge to first record observations for each floor plan independently, then compare the two layouts along six architectural criteria, and finally synthesize a concise comparative assessment. The final verdict is required to appear inside `<answer></answer>` tags as a JSON object with fields `reasoning` and `winner`, where

$$\texttt{winner} \in \{\texttt{"A"}, \texttt{"B"}, \texttt{"tie"}\}.$$

System Prompt for Pairwise VLM Judge

```
system_prompt: |-
  You are an expert architectural floorplan evaluator. You will be shown
  TWO rendered 2D floorplan images generated for the same design brief:
  Floorplan A (the first image) and Floorplan B (the second image).
  Your task is to decide which floorplan is the better architectural layout.

  The floorplans use the following visual conventions:
  - Light gray polygons with black edges: building massing (outer boundary)
  - Blue polygons: cores (stairs, elevators)
  - Orange polygons: corridors
  - Green polygons: living units (residential)
  - Gray polygons: undefined spaces
  - Labels at polygon centroids indicate space types

  ## Evaluation criteria
```

```
1. Spatial Organization (most important): Are spaces logically arranged?
   Do living units fill the massing efficiently? Are cores and
   corridors placed to provide good access to all units?

2. Space Proportions: Are individual rooms reasonably shaped, not too
   narrow and not excessively elongated? Do they have practical aspect
   ratios?

3. Coverage & Utilization: Does the layout fill the massing boundary well?
   Are there large gaps or undefined areas? Good layouts use most of the
   available floor area.

4. Circulation Quality: Are corridors and cores positioned to allow
   efficient movement through the building? Is there clear access from
   corridors to units?

5. Structural Logic: If columns/gridlines are present, are they placed in a
   regular grid pattern? Do they align with the building structure?

6. Overlap & Validity: Do spaces avoid overlapping each other? Are all
   spaces contained within the massing boundary?

First, summarize what you observe in each floorplan in a few short
bullet points, listing Floorplan A and Floorplan B separately.

Next, for each of the six criteria above, write one sentence comparing the
two floorplans grounded in your observations, and state which is stronger on
that criterion:
- A = Floorplan A is stronger
- B = Floorplan B is stronger
- tie = the two are comparable on this criterion

Based on the above, give a short summary assessment comparing the two
layouts overall. Weight Spatial Organization most heavily, and treat validity
failures, such as overlaps or spaces outside the massing, as strong evidence
against a floorplan.

After your synthesis, output your final verdict inside <answer></answer>
tags as a JSON object with exactly two fields:
- "reasoning": a brief 1-3 sentence comparative summary of the verdict
- "winner": "A" if Floorplan A is the better layout, "B" if Floorplan B is
  the better layout, or "tie" if the two are of essentially equal quality

Judge purely on the layouts shown; do not assume the image order implies
quality.

Example response structure:
Observations:
- Floorplan A: square footprint; 1 core, 1 corridor, 4 living units;
  no overlaps; near-full coverage
- Floorplan B: same footprint; 2 living units, no core or corridor;
  large empty areas
...

Criterion comparison:
1. Spatial Organization [A]: A has a corner core with a corridor serving
   every unit, while B has no circulation infrastructure.
2. Space Proportions [tie]: Both use rectangular units with practical
   proportions.
...

Assessment:
Floorplan A provides efficient circulation and near-complete coverage, while
```

```
  Floorplan B lacks any core/corridor and wastes most of the floor plate.
  Floorplan A is clearly the better layout.

  <answer>
  {
    "reasoning": "Floorplan A offers logical circulation and near-complete
    coverage, whereas B has no circulation infrastructure and large unused
    areas.",
    "winner": "A"
  }
  </answer>

  Always end your response with the <answer></answer> block. Do not put
  anything after the closing </answer> tag.

user_query: |-
  Compare these two floorplans. Floorplan A is the first image, and Floorplan B
  is the second. Follow the three-step protocol from the system instructions:
  (1) Observations for Floorplan A and Floorplan B, (2) Criterion comparison
  across all six criteria, (3) Synthesis, then the <answer> JSON block with
  the winner.
```

Order-bias Mitigation. To reduce sensitivity to presentation order, each pair is evaluated twice: once in the original order (A, B) and once in the swapped order (B, A). The verdict from the swapped comparison is mapped back to the original model labels before aggregation. If both judgments agree after canonicalization, the corresponding win direction is accepted. If the two judgments disagree, the comparison is conservatively recorded as a tie.

Aggregation and Uncertainty. Let n denote the number of evaluated prompts, and let w_A, w_B, and t denote the number of strict wins for model A, strict wins for model B, and ties, respectively. We report

$$p_A = \frac{w_A}{n}, \qquad p_{\text{tie}} = \frac{t}{n}, \qquad p_B = \frac{w_B}{n},$$

so that $p_A + p_{\text{tie}} + p_B = 1$. Ties are therefore reported as a separate outcome rather than being split between the two models.

For each strict win rate, we report a 95% Wilson score confidence interval. For a binomial proportion $\hat{p} = w/n$ and $z = 1.96$, the interval is

$$\text{CI}_{95\%}(\hat{p}) = \left[\frac{\hat{p} + \frac{z^2}{2n} - z\sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}}, \frac{\hat{p} + \frac{z^2}{2n} + z\sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}} \right], \quad z = 1.96.$$

These intervals quantify uncertainty due to finite prompt sampling within a single evaluation run and do not capture cross-seed variation.

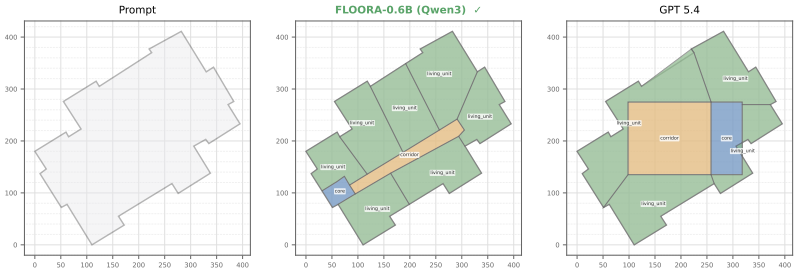
F.2 Validation Against Architect Feedback

To evaluate whether the VLM judge reflects expert architectural preferences, we apply the same pairwise judging protocol to preference pairs from the reward-model dataset. We use a VLM judge rather than the learned reward model for evaluation because FLOORA is directly optimized using that reward model during post-training, which would bias the evaluation in favor of our models.

For each pair, we render both layouts and evaluate them using the same VLM judge protocol described above, including evaluation under both presentation orders. Across approximately 10k model–architect pairs, the VLM judge agrees with the architect-implied preference in **77.5%** of cases. This provides direct evidence that the VLM judge is meaningfully aligned with expert architectural corrections, while also indicating that it does not perfectly reproduce human judgment.

F.3 Examples of the VLM Judge Outcomes

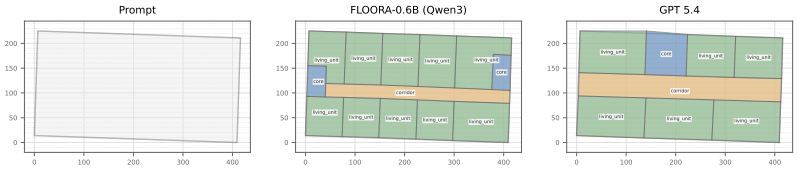
Figure 18 shows representative VLM judge reasoning for a FLOORA win, a tie, and a baseline win. The examples show that decisions reflect architectural criteria, including spatial organization, circulation, coverage, proportions, and validity, rather than visual appearance alone. The tie case also demonstrates how comparable strengths and weaknesses are handled without forcing a preference.



Ordering 1 [image A = FLOORA-0.6B (Qwen3), image B = GPT 5.4] -> judge picked A (FLOORA-0.6B (Qwen3))
Floorplan A features an efficient, clean layout with a logical central corridor and zero overlaps, whereas Floorplan B suffers from severe validity failures with a massive central core and corridor overlapping multiple living units.

Ordering 2 [image A = GPT 5.4, image B = FLOORA-0.6B (Qwen3)] -> judge picked B (FLOORA-0.6B (Qwen3))
Floorplan B aligns its circulation and core with the natural diagonal axis of the building, resulting in well-proportioned units, whereas Floorplan A uses an oversized, blocky central corridor that forces the living units into highly awkward, wrapping shapes.

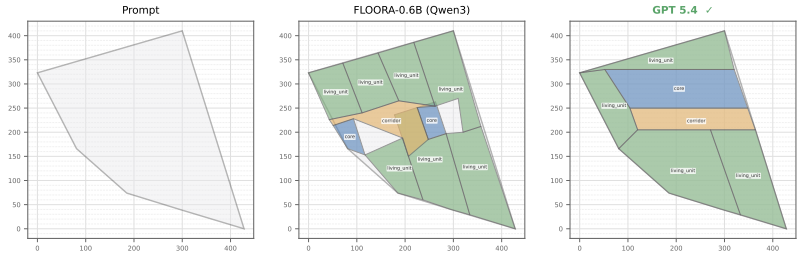
(a) Example VLM judge reasoning for a comparison in which the FLOORA model is preferred.



Ordering 1 [image A = FLOORA-0.6B (Qwen3), image B = GPT 5.4] -> judge picked A (FLOORA-0.6B (Qwen3))
Floorplan A is highly efficient, featuring a balanced double-loaded corridor layout with ten regularly proportioned units and two well-placed cores, whereas Floorplan B has fewer, less-consistent units and an awkwardly positioned central core.

Ordering 2 [image A = GPT 5.4, image B = FLOORA-0.6B (Qwen3)] -> judge picked A (GPT 5.4)
Floorplan A provides a much cleaner spatial organization with practical, well-proportioned living units and a logically integrated central core, whereas Floorplan B features overly narrow units and awkwardly placed cores.

(b) Example VLM judge reasoning for a comparison judged as a tie.



Ordering 1 [image A = FLOORA-0.6B (Qwen3), image B = GPT 5.4] -> judge picked B (GPT 5.4)
Floorplan B is a clean, valid layout with a logical central core and corridor, whereas Floorplan A suffers from severe geometric overlaps between elements and leaves unprogrammed gaps.

Ordering 2 [image A = GPT 5.4, image B = FLOORA-0.6B (Qwen3)] -> judge picked A (GPT 5.4)
Floorplan A is a clean, well-organized layout with excellent coverage and no geometry errors. Floorplan B is architecturally invalid due to severe overlapping polygons, awkward room shapes, and unassigned gaps.

(c) Example VLM judge reasoning for a comparison in which the frontier model is preferred.

Figure 18: Example VLM judge reasoning for representative pairwise outcomes.

G Frontier Model Baselines and Inference Protocol

To establish a strong external baseline, we evaluate a set of frontier LLMs on the same floor-plan generation task used for our domain-specific models. Unlike the models trained in this work, these frontier models are accessed only through inference APIs and are not fine-tuned on the DSL or on the architectural feedback dataset. This evaluation therefore measures the few-shot ability of general-purpose frontier models to produce syntactically valid and architecturally plausible floor-plan layouts from structured DSL context.

All frontier models are evaluated on the same prompt distributions used in the main text, including synthetic test prompts and OpenStreetMap-derived real-world massing prompts. Their completions are scored with the same verifiable reward functions described in Section 6.2, including geometric correctness, functional compliance, and aggregate total reward. As in the main evaluation, a completion passes the total reward criterion only when all required checks pass. This shared protocol enables direct comparison between frontier model completions and completions from our trained DSL models. We also compute the win-rate of our models against frontier baselines using a VLM judge.

G.1 Models Evaluated

We evaluate Claude Sonnet 4.6, Claude Opus 4.8, OpenAI GPT-5.4, Gemini 2.5 Flash Image, and Gemini 3.5 Flash through inference APIs, all with a maximum of 8,192 tokens and provider-default temperature. All models receive identical prompts with one illustrative example of the expected DSL syntax.

G.2 Prompt Format

Each query consists of a system message that defines the floor-plan generation task and a user message containing the full DSL context for the evaluated sample. The user message includes all available input modalities, such as the `building`, `structure`, `massing` blocks. The model is instructed to return only a complete `spaces` block delimited by sentinel tokens.

System Prompt for Frontier Model Baselines

```
You are an expert architectural design assistant generating floor plan layouts.

You will receive a building description in DSL format. The DSL uses named
modality blocks:
- `building { ... }`: building metadata, including occupancy type and floor levels
- `structure { ... }`: structural grid and material information
- `massing { polygon <type> x,y x,y ... }`: the building massing footprint polygon
- `spaces { polygon <type> x,y x,y ... }`: individual spaces, when partially provided

All coordinates are quantized integers in the range 0 to 1023. Each
`polygon <type>` is followed by space-separated `x,y` coordinate pairs defining
a closed polygon. Do NOT repeat the first vertex at the end.

Space types: `core`, `corridor`, `living_unit`.

Cores are spaces intended to contain stairs, elevators, or other building
services. Corridors are spaces intended for circulation. Living units are spaces
intended for residential use.

Your task is to generate a complete `spaces` layout for the given massing.
Produce a DSL `spaces { ... }` block with exactly the following syntax:

spaces {
  polygon <type> x,y x,y x,y ...
  polygon <type> x,y x,y x,y ...
  ...
}
```

```
Rules:
- All coordinate values must be integers in the range 0 to 1023.
- Each polygon must have at least 3 vertices.
- Do NOT repeat the first vertex at the end.
- Spaces must tile the massing polygon completely. Every point inside the
  massing must belong to exactly one space.
- There must be no uncovered area within the massing boundary.
- Every `living_unit` must share at least one edge with the exterior facade,
  defined by the massing boundary.
- Every `living_unit` must share at least one edge with a `corridor`.
- Space sizes must be architecturally plausible. For example, each core must
  be large enough to realistically contain stairs, elevators, or building
  services.

Example

Input:
building { occupancy_type multifamily_residential storeys 4 level 1 elevation 0 }
structure { material reinforced_concrete }
massing { polygon mass 0,0 400,0 400,200 0,200 }

Output:
STARTING_GENERATION
spaces {
  polygon living_unit 0,0 160,0 160,80 0,80
  polygon core 160,0 240,0 240,80 160,80
  polygon living_unit 240,0 400,0 400,80 240,80
  polygon corridor 0,80 400,80 400,120 0,120
  polygon living_unit 0,120 200,120 200,200 0,200
  polygon living_unit 200,120 400,120 400,200 200,200
}
END_GENERATION

Output your response in this exact format:
STARTING_GENERATION
spaces {
  ...
}
END_GENERATION

No markdown, no explanations, and no extra text outside the sentinels.
```

G.3 Output Parsing

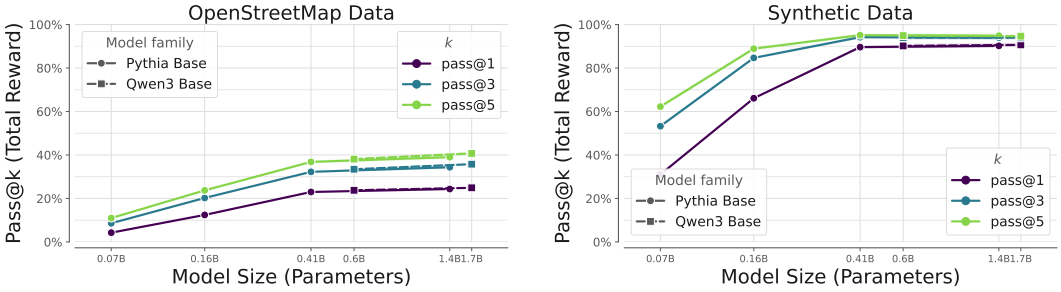
Frontier models are instructed to delimit their generations using the sentinel tokens `STARTING_GENERATION` and `END_GENERATION`. When both sentinels are present, we extract the text between them and strip surrounding whitespace. When one or both sentinels are absent, which can occur when a model emits preamble text or otherwise deviates from the requested format, the full model response is retained and passed to the parser.

The extracted text is validated using the same DSL parser used during data preprocessing and evaluation, described in Appendix A. This parser checks whether the generated text can be interpreted as a valid `spaces { ... }` block and converted into the internal geometric representation. Responses that fail parsing are not discarded. Instead, they are passed to the reward functions, which assign zero reward to syntactically invalid DSL. This preserves comparability with trained model completions, where invalid generations are evaluated under the same scoring pipeline rather than filtered out before evaluation.

H Additional Experimental Results

H.1 Pre-Training Results

Figure 19 and Tables 8 and 9 present pass@1/3/5 evaluation results for the base models across different model scales on the synthetic and OpenStreetMap (OSM) test sets. Pass@k for the total reward is considered achieved only when both the geometric correctness and functional compliance checks pass. Models with fewer than 0.4B parameters struggle to learn the underlying building patterns, resulting in poor performance even on synthetic data. While larger models largely saturate on the synthetic benchmark, their generalization continues to improve on OSM, which represents a held-out evaluation setting.



(a) Pass@k on the OSM test set.

(b) Pass@k on the synthetic test set.

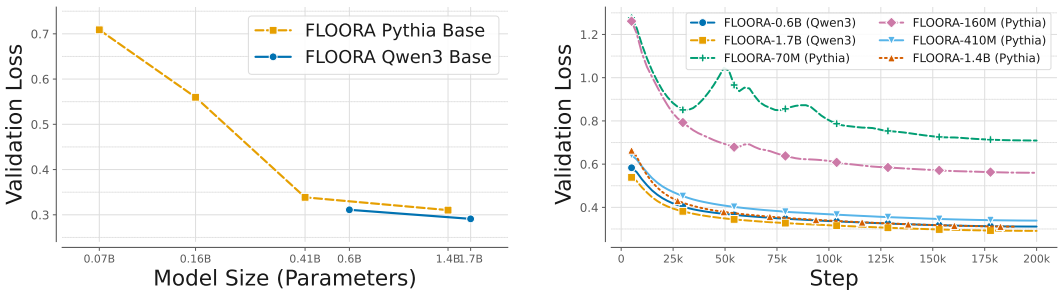
Figure 19: Pass@1/3/5 performance of the base models on OSM and synthetic test sets. Pass@k is considered achieved only when both the geometric correctness and functional compliance checks pass. Shaded regions represent 95% CIs across 5 evaluation seeds.

Figure 20 shows pre-training validation loss, measured as next-token cross-entropy, across model scales throughout training. Consistent with the downstream results, models below 0.4B parameters struggle to capture the underlying building patterns, obtaining higher validation cross-entropy at the end of the pre-training stage.

H.2 Supervised Fine-Tuning

Figure 24 compares the pass@1/3/5 results for the base and SFT checkpoints across model families and scales. SFT improves performance on the OSM test set across the evaluated settings, indicating that architect-edited completions provide an effective alignment signal for real-world building footprints. Since OSM is out-of-distribution relative to the synthetic pre-training data and SFT data (see Figure 4a), these gains suggest improved transfer beyond the procedural data distribution.

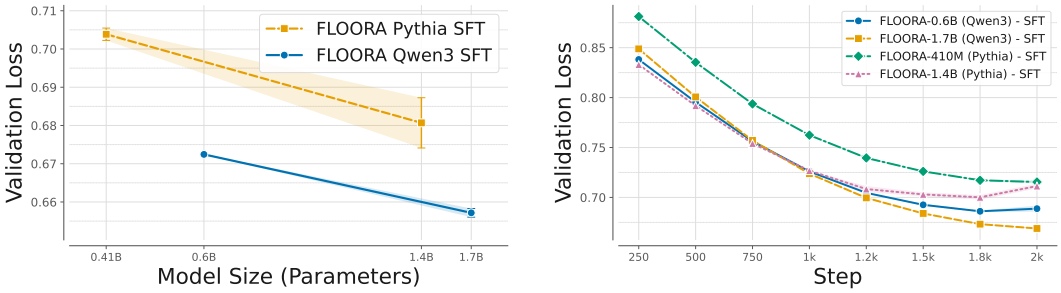
On the synthetic test set, we observe a modest decrease in pass@k performance after SFT. We attribute this to the distribution shift between the procedural synthetic layouts used during pre-training



(a) Final validation loss.

(b) Validation loss curves.

Figure 20: Pre-training validation across model scales. **(a)** Final validation loss achieved by each model. **(b)** Validation loss throughout pre-training.



(a) Final validation loss.

(b) Validation loss curves.

Figure 21: SFT validation loss across model scales. **(a)** Final validation loss achieved by each model. **(b)** Validation loss throughout training. Shaded regions represent 95% CIs across 3 training seeds.

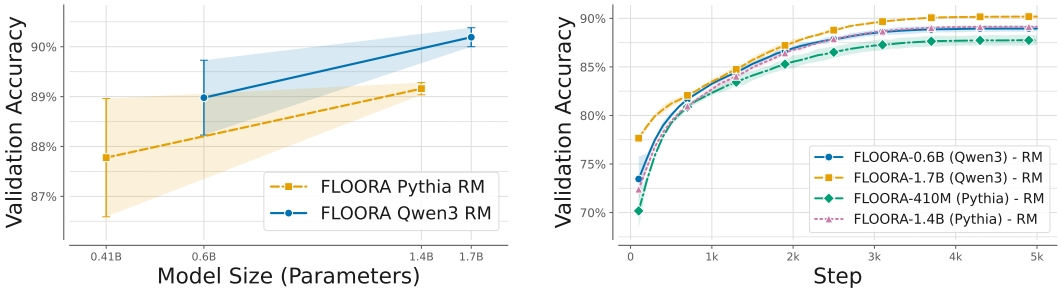
and the architect-corrected completions used for SFT. In this sense, SFT shifts the model away from reproducing the synthetic generator distribution and toward layouts that better reflect expert architectural judgment.

Figure 21 reports SFT validation cross-entropy. These metrics provide a training diagnostic for the supervised objective, but they should be interpreted alongside the pass@k results rather than as direct measures of architectural quality. Overall, the results suggest that SFT functions primarily as a domain-alignment stage, trading some in-distribution synthetic performance for improved generalization to real-world footprints.

H.3 Reward Model Training

Figure 22 presents reward model validation accuracy. Accuracy improves rapidly during the early stages of optimization and then plateaus, indicating that the models learn most of the pairwise preference signal within the first few thousand training steps.

As expected, final validation accuracy increases with model size, suggesting that larger reward models better capture the architectural preferences expressed in the feedback data.



(a) Final validation accuracy.

(b) Validation accuracy curves.

Figure 22: Reward model validation accuracy across model scales. **(a)** Final validation accuracy achieved by each model. **(b)** Validation accuracy throughout reward model training. Shaded regions represent 95% CIs across 3 training seeds.

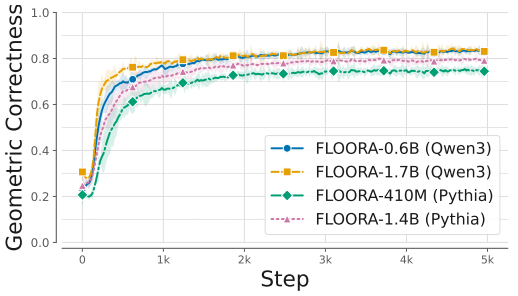
H.4 Reinforcement Learning

Figure 23 shows the GRPO learning curves for the reward components. Across model scales, training is stable; geometric correctness, functional compliance, and the reward-model score improve quickly and then plateau, while the soft overlong penalty remains near zero.

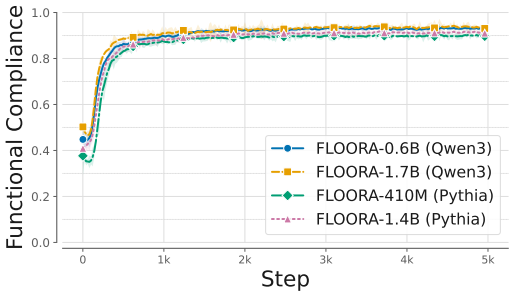
The main post-training results are shown in Figure 24 and Tables 10 and 11. Both GRPO variants substantially improve pass@1/3/5 on OSM and synthetic data, indicating that RL fixes many of the validity and constraint-satisfaction issues left by the base and SFT models. GRPO produces consistent positive improvements across datasets and model scales. The only exception is SFT on the synthetic test set, where performance drops due to the distribution shift discussed in Appendix H.2. Overall, GRPO recovers this loss while preserving the OSM gains from the SFT stage.

Finally, we evaluate the effect of the learned reward model during post-training by comparing models trained with GRPO (RM+VR) against matched GRPO (VR only) models using the VLM judge described in Appendix F. Figure 25 reports pairwise outcomes supplementing Figure 8, where a win indicates that the RM+VR model is preferred over its VR only counterpart. The results show that the benefit of including the reward model grows with model scale. In particular, RM+VR underperforms VR only for the smallest model, FLOORA-410M (Pythia), but becomes increasingly preferred as model capacity increases, reaching the strongest margins for FLOORA-1.7B (Qwen3).

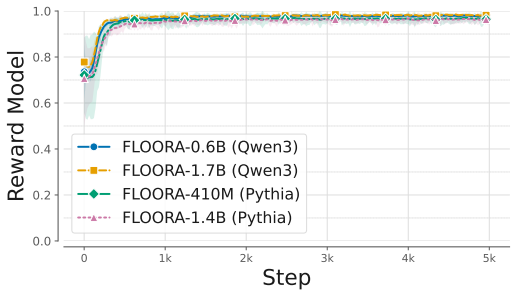
This trend suggests that the learned reward model can improve qualitative architectural preferences beyond VR-only training when both the policy and reward model have sufficient capacity to capture and exploit architectural-quality judgments, but that its benefits are capacity-dependent and may be limited or even harmful for smaller models.



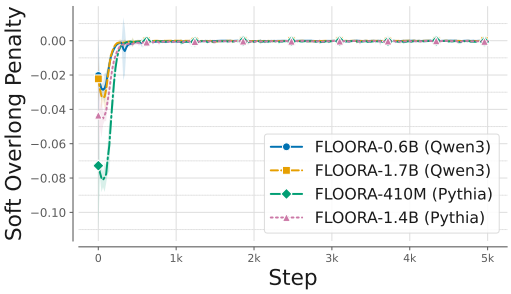
(a) Geometric correctness reward curves.



(b) Functional compliance reward curves.

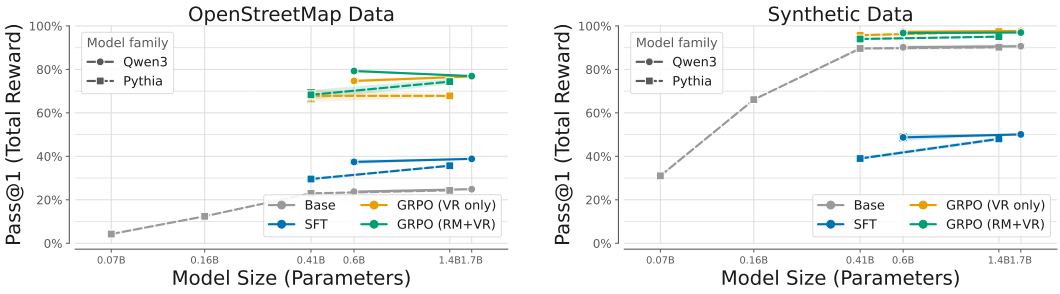


(c) Reward model curves.



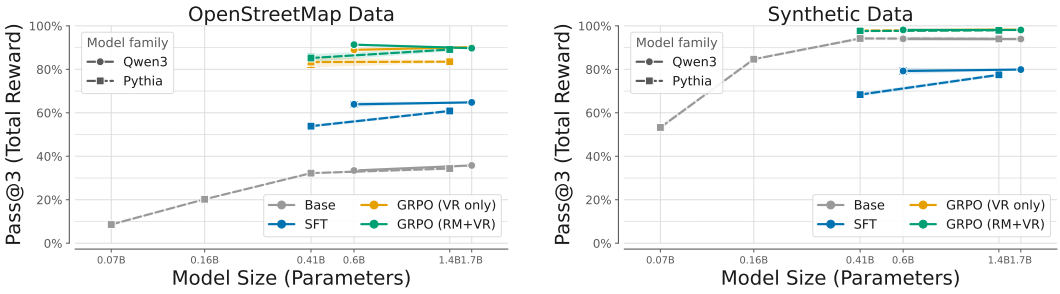
(d) Soft overlong penalty curves.

Figure 23: Learning curves for GRPO reward components across model scales, including geometric correctness, functional compliance, the normalized reward model score, and the soft overlong penalty. Shaded regions show 95% CIs across 3 training seeds.



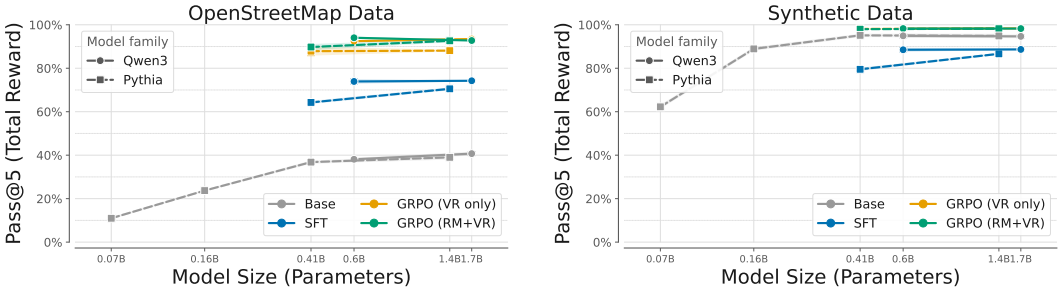
(a) Pass@1 on the OSM test set.

(b) Pass@1 on the synthetic test set.



(c) Pass@3 on the OSM test set.

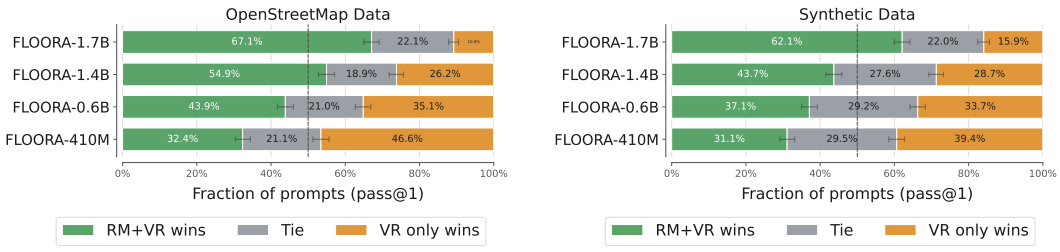
(d) Pass@3 on the synthetic test set.



(e) Pass@5 on the OSM test set.

(f) Pass@5 on the synthetic test set.

Figure 24: Pass@1/3/5 performance of the base and fine-tuned models on OSM and synthetic test sets. Pass@k is considered achieved only when both the geometric correctness and functional compliance checks pass. Shaded regions represent 95% CIs across 3 training and 5 evaluation seeds.



(a) Win rates on the OSM test set.

(b) Win rates on the synthetic test set.

Figure 25: Pairwise VLM judge outcomes for each FLOORA GRPO (RM+VR) model against its matched GRPO (VR only) counterpart. Bars show the fraction of prompts where RM+VR wins, ties, or loses. As model size increases, the RM+VR variant is increasingly preferred. Error bars indicate 95% Wilson intervals over the evaluation prompts.

Table 8: Pass@1/3/5 performance of the base models on the **OSM** test set, reported with 95% CIs across 5 evaluation seeds. Best value within each model family is shown in bold.

	Functional Compliance			Geometric Correctness			Total Reward		
Model	@1	@3	@5	@1	@3	@5	@1	@3	@5
FLOORA-0.6B (Qwen3) – Base	34.2 ± 0.4	51.2 ± 0.8	58.8 ± 1.2	27.8 ± 0.3	38.5 ± 0.6	43.2 ± 0.8	23.8 ± 0.1	33.4 ± 0.4	38.1 ± 0.7
FLOORA-1.7B (Qwen3) – Base	$\mathbf{35.6 \pm 0.4}$	$\mathbf{53.3 \pm 0.5}$	$\mathbf{61.2 \pm 0.4}$	$\mathbf{28.7 \pm 0.1}$	$\mathbf{40.5 \pm 0.4}$	$\mathbf{45.5 \pm 0.6}$	$\mathbf{24.9 \pm 0.3}$	$\mathbf{35.8 \pm 0.6}$	$\mathbf{40.7 \pm 0.9}$
FLOORA-70M (Pythia) – Base	26.6 ± 0.4	47.6 ± 0.6	56.8 ± 0.8	4.3 ± 0.2	8.7 ± 0.5	11.1 ± 0.6	4.2 ± 0.2	8.6 ± 0.4	11.0 ± 0.6
FLOORA-160M (Pythia) – Base	26.3 ± 0.3	43.8 ± 0.5	52.3 ± 0.8	13.1 ± 0.2	21.3 ± 0.2	24.9 ± 0.2	12.4 ± 0.1	20.2 ± 0.3	23.7 ± 0.5
FLOORA-410M (Pythia) – Base	34.6 ± 0.4	52.3 ± 0.6	60.6 ± 0.7	26.7 ± 0.3	36.9 ± 0.5	41.5 ± 0.5	23.0 ± 0.1	32.2 ± 0.4	36.8 ± 0.7
FLOORA-1.4B (Pythia) – Base	$\mathbf{37.6 \pm 0.3}$	$\mathbf{56.8 \pm 0.4}$	$\mathbf{65.6 \pm 0.7}$	$\mathbf{28.0 \pm 0.3}$	$\mathbf{39.1 \pm 0.9}$	$\mathbf{44.0 \pm 1.2}$	$\mathbf{24.3 \pm 0.4}$	$\mathbf{34.4 \pm 0.7}$	$\mathbf{39.0 \pm 1.0}$

Table 9: Pass@1/3/5 performance of the base models on the **synthetic** test set, reported with 95% CIs across 5 evaluation seeds. Best value within each model family is shown in bold.

	Functional Compliance			Geometric Correctness			Total Reward		
Model	@1	@3	@5	@1	@3	@5	@1	@3	@5
FLOORA-0.6B (Qwen3) – Base	90.9 ± 0.9	$\mathbf{94.3 \pm 0.9}$	$\mathbf{95.2 \pm 0.9}$	96.9 ± 0.4	98.4 ± 0.3	$\mathbf{98.5 \pm 0.3}$	90.2 ± 1.0	$\mathbf{94.0 \pm 1.0}$	$\mathbf{94.9 \pm 1.0}$
FLOORA-1.7B (Qwen3) – Base	$\mathbf{91.3 \pm 0.9}$	94.2 ± 0.8	95.0 ± 0.8	$\mathbf{97.2 \pm 0.4}$	$\mathbf{98.4 \pm 0.3}$	$\mathbf{98.5 \pm 0.3}$	$\mathbf{90.7 \pm 0.9}$	93.9 ± 0.8	94.7 ± 0.8
FLOORA-70M (Pythia) – Base	70.2 ± 1.0	90.6 ± 0.7	93.8 ± 0.6	32.0 ± 0.6	54.8 ± 0.8	64.0 ± 0.7	31.0 ± 0.6	53.2 ± 0.9	62.2 ± 0.9
FLOORA-160M (Pythia) – Base	79.2 ± 0.6	91.9 ± 0.6	93.8 ± 0.5	69.6 ± 0.5	88.3 ± 0.6	92.3 ± 0.8	66.1 ± 0.3	84.7 ± 0.5	88.9 ± 0.9
FLOORA-410M (Pythia) – Base	91.1 ± 0.7	$\mathbf{94.7 \pm 0.7}$	$\mathbf{95.6 \pm 0.7}$	96.0 ± 0.3	98.3 ± 0.3	98.4 ± 0.3	89.6 ± 0.7	$\mathbf{94.2 \pm 0.7}$	$\mathbf{95.1 \pm 0.8}$
FLOORA-1.4B (Pythia) – Base	$\mathbf{91.3 \pm 0.9}$	94.4 ± 0.9	95.2 ± 0.9	$\mathbf{96.7 \pm 0.3}$	$\mathbf{98.4 \pm 0.3}$	$\mathbf{98.5 \pm 0.3}$	$\mathbf{90.2 \pm 0.9}$	94.0 ± 0.9	94.9 ± 1.0

Table 10: Pass@1/3/5 performance of the post-trained models on the **OSM** test set, reported with 95% CIs across 3 training and 5 evaluation seeds. The final column reports percentage-point changes in Total Reward relative to the base checkpoint. Best absolute value within each model family is shown in bold.

	Functional Compliance			Geometric Correctness			Total Reward			Δ Total Reward (vs Base)		
Model	@1	@3	@5	@1	@3	@5	@1	@3	@5	@1	@3	@5
FLOORA-0.6B (Qwen3) – SFT	45.5 ± 1.0	73.4 ± 0.9	82.7 ± 0.8	45.1 ± 0.9	71.3 ± 1.0	79.6 ± 0.8	37.4 ± 1.0	63.9 ± 1.2	73.9 ± 1.1	+13.7	+30.5	+35.8
FLOORA-0.6B (Qwen3) – GRPO (RM+VR)	85.6 ± 0.3	95.6 ± 0.2	97.5 ± 0.2	**80.9 ± 0.7**	**92.3 ± 0.5**	**94.8 ± 0.5**	**79.3 ± 0.7**	**91.3 ± 0.5**	**94.0 ± 0.5**	**+55.5**	**+57.9**	**+55.9**
FLOORA-0.6B (Qwen3) – GRPO (VR only)	84.0 ± 0.7	95.3 ± 0.4	97.2 ± 0.3	76.3 ± 0.5	90.1 ± 0.3	93.5 ± 0.3	74.7 ± 0.7	89.0 ± 0.5	92.5 ± 0.5	+50.9	+55.5	+54.3
FLOORA-1.7B (Qwen3) – SFT	46.1 ± 0.2	73.5 ± 0.3	82.5 ± 0.3	46.6 ± 0.2	72.1 ± 0.3	79.9 ± 0.3	38.8 ± 0.2	64.8 ± 0.3	74.2 ± 0.3	+13.9	+29.0	+33.5
FLOORA-1.7B (Qwen3) – GRPO (RM+VR)	**85.8 ± 0.4**	95.7 ± 0.2	97.4 ± 0.2	78.2 ± 0.1	90.5 ± 0.2	93.4 ± 0.2	76.9 ± 0.1	89.7 ± 0.2	92.8 ± 0.3	+52.0	+54.0	+52.0
FLOORA-1.7B (Qwen3) – GRPO (VR only)	85.0 ± 0.3	**95.8 ± 0.1**	**97.7 ± 0.2**	78.2 ± 0.8	90.9 ± 0.5	94.0 ± 0.4	76.7 ± 0.8	90.2 ± 0.4	93.5 ± 0.3	+51.9	+54.4	+52.7
FLOORA-410M (Pythia) – SFT	36.7 ± 0.6	64.0 ± 0.7	74.5 ± 0.8	36.0 ± 0.7	61.4 ± 0.8	70.8 ± 0.7	29.6 ± 0.6	53.8 ± 0.7	64.3 ± 0.7	+6.6	+21.6	+27.5
FLOORA-410M (Pythia) – GRPO (RM+VR)	78.7 ± 1.5	93.3 ± 0.9	96.3 ± 0.5	69.9 ± 2.2	86.1 ± 1.8	90.5 ± 1.4	68.3 ± 2.3	85.2 ± 1.8	89.8 ± 1.4	+45.3	+53.0	+53.0
FLOORA-410M (Pythia) – GRPO (VR only)	79.3 ± 1.5	92.6 ± 1.0	95.5 ± 0.7	69.6 ± 2.7	84.5 ± 2.5	88.8 ± 2.0	67.8 ± 2.7	83.3 ± 2.5	87.9 ± 2.0	+44.9	+51.1	+51.0
FLOORA-1.4B (Pythia) – SFT	43.1 ± 0.5	70.8 ± 0.5	80.4 ± 0.4	43.1 ± 0.5	68.5 ± 0.5	76.8 ± 0.5	35.7 ± 0.6	60.8 ± 0.7	70.5 ± 0.7	+11.3	+26.5	+31.5
FLOORA-1.4B (Pythia) – GRPO (RM+VR)	**82.4 ± 1.0**	**95.0 ± 0.5**	**97.3 ± 0.3**	**76.0 ± 1.4**	**90.1 ± 0.9**	**93.3 ± 0.7**	**74.4 ± 1.3**	**89.1 ± 0.9**	**92.6 ± 0.7**	**+50.0**	**+54.8**	**+53.6**
FLOORA-1.4B (Pythia) – GRPO (VR only)	79.1 ± 0.7	92.6 ± 0.4	95.6 ± 0.3	69.5 ± 0.7	84.4 ± 0.6	88.8 ± 0.5	67.8 ± 0.7	83.5 ± 0.5	88.1 ± 0.5	+43.5	+49.1	+49.1

Table 11: Pass@1/3/5 performance of the post-trained models on the **synthetic** test set, reported with 95% CIs across 3 training and 5 evaluation seeds. The final column reports percentage-point changes in Total Reward relative to the base checkpoint. Best absolute value within each model family is shown in bold.

	Functional Compliance			Geometric Correctness			Total Reward			Δ Total Reward (vs Base)		
Model	@1	@3	@5	@1	@3	@5	@1	@3	@5	@1	@3	@5
FLOORA-0.6B (Qwen3) – SFT	57.0 ± 1.3	86.5 ± 0.9	93.6 ± 0.5	49.3 ± 1.5	79.7 ± 1.4	88.8 ± 0.9	48.7 ± 1.5	79.2 ± 1.4	88.5 ± 0.9	-41.5	-14.8	-6.4
FLOORA-0.6B (Qwen3) – GRPO (RM+VR)	97.4 ± 0.2	98.5 ± 0.1	98.6 ± 0.1	96.8 ± 0.2	98.2 ± 0.2	98.3 ± 0.1	96.8 ± 0.2	98.1 ± 0.2	98.3 ± 0.2	+6.6	+4.1	+3.3
FLOORA-0.6B (Qwen3) – GRPO (VR only)	97.9 ± 0.2	98.4 ± 0.1	98.5 ± 0.1	97.5 ± 0.1	98.2 ± 0.1	98.3 ± 0.1	97.3 ± 0.1	98.1 ± 0.2	98.2 ± 0.2	**+7.1**	+4.0	+3.2
FLOORA-1.7B (Qwen3) – SFT	57.1 ± 0.3	86.0 ± 0.3	92.9 ± 0.3	50.7 ± 0.3	80.3 ± 0.3	88.9 ± 0.2	50.1 ± 0.3	79.9 ± 0.3	88.7 ± 0.2	-40.6	-14.0	-6.0
FLOORA-1.7B (Qwen3) – GRPO (RM+VR)	97.6 ± 0.2	98.5 ± 0.1	98.6 ± 0.1	97.0 ± 0.2	98.2 ± 0.1	98.3 ± 0.1	97.0 ± 0.2	98.1 ± 0.1	98.2 ± 0.1	+6.3	+4.2	+3.6
FLOORA-1.7B (Qwen3) – GRPO (VR only)	**98.2 ± 0.1**	**98.6 ± 0.1**	**98.7 ± 0.1**	**97.8 ± 0.2**	**98.3 ± 0.1**	**98.4 ± 0.1**	**97.7 ± 0.2**	**98.3 ± 0.1**	**98.3 ± 0.1**	+7.0	**+4.3**	**+3.7**
FLOORA-410M (Pythia) – SFT	46.2 ± 0.7	76.8 ± 0.6	86.8 ± 0.5	39.5 ± 0.8	68.8 ± 0.9	79.8 ± 0.7	39.0 ± 0.8	68.4 ± 0.9	79.5 ± 0.7	-50.6	-25.8	-15.6
FLOORA-410M (Pythia) – GRPO (RM+VR)	95.8 ± 0.4	98.4 ± 0.1	98.6 ± 0.1	94.0 ± 0.6	97.6 ± 0.1	98.0 ± 0.1	94.0 ± 0.6	97.6 ± 0.1	98.0 ± 0.1	+4.4	+3.4	+2.8
FLOORA-410M (Pythia) – GRPO (VR only)	97.0 ± 0.6	98.5 ± 0.1	98.7 ± 0.1	95.8 ± 0.9	97.9 ± 0.2	98.1 ± 0.2	95.7 ± 0.9	97.8 ± 0.2	98.1 ± 0.2	+6.1	+3.6	+3.0
FLOORA-1.4B (Pythia) – SFT	54.8 ± 0.8	84.2 ± 0.6	91.9 ± 0.4	48.6 ± 0.9	77.9 ± 0.7	87.0 ± 0.5	48.0 ± 0.9	77.5 ± 0.7	86.6 ± 0.5	-42.2	-16.5	-8.2
FLOORA-1.4B (Pythia) – GRPO (RM+VR)	96.2 ± 0.3	98.5 ± 0.1	**98.7 ± 0.1**	95.1 ± 0.3	97.9 ± 0.2	98.2 ± 0.1	95.1 ± 0.3	97.9 ± 0.2	98.2 ± 0.2	+4.9	+3.9	+3.3
FLOORA-1.4B (Pythia) – GRPO (VR only)	**98.0 ± 0.1**	**98.5 ± 0.1**	98.6 ± 0.1	**97.4 ± 0.1**	**98.2 ± 0.1**	**98.3 ± 0.1**	**97.4 ± 0.1**	**98.1 ± 0.1**	**98.3 ± 0.1**	**+7.1**	**+4.2**	**+3.4**

Table 12: Pass@1/3/5 comparison of the FLOORA-0.6B (Qwen3) – GRPO (RM+VR) and frontier baselines on the **OSM** test set. Best absolute value is shown in bold. Results are obtained on a single seed.

	Functional Compliance			**Geometric Correctness**			**Total Reward**		
Model	@1	@3	@5	@1	@3	@5	@1	@3	@5
FLOORA-0.6B (Qwen3) – GRPO (RM+VR)	**84.8**	**95.3**	**97.2**	**79.5**	**91.6**	**94.0**	**78.1**	**90.8**	**93.5**
Claude Opus 4.8	46.1	73.6	83.0	36.4	60.8	72.0	24.7	45.0	55.6
Claude Sonnet 4.6	19.3	38.4	49.4	10.0	20.2	26.8	5.1	10.7	14.5
Gemini 2.5 Flash Image	37.6	62.0	71.7	25.5	44.3	53.8	11.1	20.3	26.0
Gemini 3.5 Flash	52.6	76.6	83.9	42.0	69.1	79.7	33.3	57.4	68.2
OpenAI GPT 5.4	46.6	71.6	79.7	30.3	50.4	60.4	19.9	36.0	45.0

Table 13: Pass@1/3/5 comparison of the FLOORA-0.6B (Qwen3) – GRPO (RM+VR) model and frontier baselines on the **synthetic** test set. Best absolute value is shown in bold. Results are obtained on a single seed.

	Functional Compliance			**Geometric Correctness**			**Total Reward**		
Model	@1	@3	@5	@1	@3	@5	@1	@3	@5
FLOORA-0.6B (Qwen3) – GRPO (RM+VR)	**97.4**	**98.3**	**98.4**	**96.5**	**97.8**	**98.0**	**96.5**	**97.7**	**97.9**
Claude Opus 4.8	66.5	92.2	97.1	38.6	62.9	72.9	30.9	54.2	64.5
Claude Sonnet 4.6	39.0	70.2	82.5	8.1	15.5	20.4	6.9	13.5	17.9
Gemini 2.5 Flash Image	64.1	90.7	96.7	15.6	29.7	38.0	12.9	24.0	30.6
Gemini 3.5 Flash	75.3	96.4	99.2	39.9	66.2	77.0	39.1	65.2	76.0
OpenAI GPT 5.4	69.8	93.8	98.1	27.4	44.1	52.2	23.9	40.5	48.5

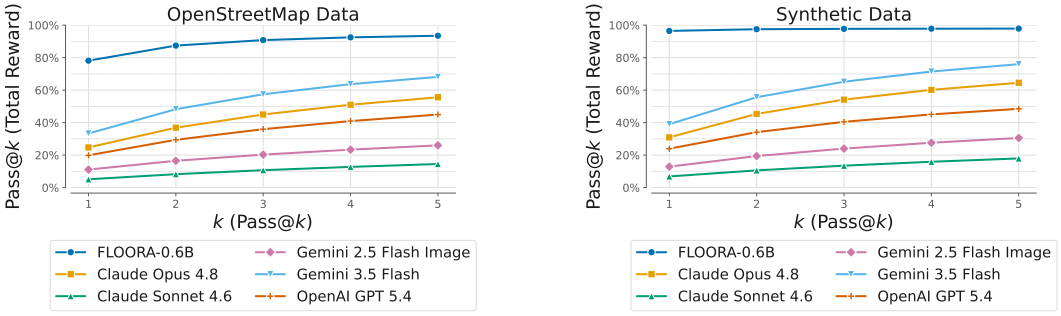
H.5 Comparison with Frontier Models

Verifier-Based Evaluations. We provide additional results for the comparison between our FLOORA-0.6B (Qwen3) GRPO (RM+VR) model and the frontier-model baselines evaluated under the inference protocol in Appendix G. Figure 26 and Tables 12 and 13 supplement Figure 9a by reporting the corresponding pass@1, pass@3, and pass@5 results on the OSM and synthetic test sets. Across both benchmarks, our model achieves the highest total reward pass@k values.

Pairwise Win Rates Using a VLM Judge. We report the full pairwise VLM judge outcomes (Appendix F) in Figure 27, supplementing Figure 9b. FLOORA-0.6B is preferred over every frontier baseline, with win rates of 73.8%–96.0% across OSM and synthetic prompts. These results show that the gains extend beyond verifier-defined checks to visual and architectural quality.

Human Evaluations. We conduct a human preference evaluation on 100 test prompts, comprising 50 OSM and 50 synthetic samples. For each prompt, 10 labelers view outputs from all six models and select the single best layout according to the architectural criteria in Appendix C.2. The evaluation labelers are distinct from the architects who provided training feedback. Model identities are hidden, and output order is randomized for each prompt to mitigate position bias.

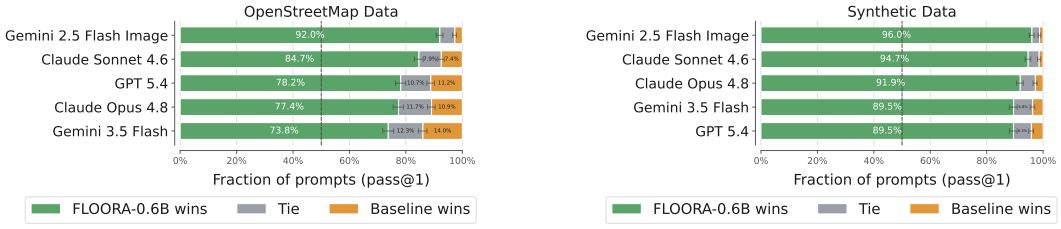
We report each model's preference rate as its fraction of votes across all labelers and prompts. Since the labelers evaluate the same prompts, votes within a prompt are correlated and cannot be treated as independent. We therefore compute 95% CIs using a cluster bootstrap over prompts. In each of 10,000 iterations, we sample 100 prompts with replacement, retain all 10 associated votes for each sampled prompt, and recompute each model's preference rate. The 2.5th and 97.5th percentiles of the bootstrap distribution define the confidence interval. This preserves the clustered evaluation structure and avoids understating uncertainty by treating all 1,000 votes as independent.



(a) Pass@k on the OSM test set.

(b) Pass@k on the synthetic test set.

Figure 26: Pass@k comparison of the FLOORA-0.6B (Qwen3) GRPO (RM+VR) model and frontier baselines, where success requires passing both geometric and functional checks. Our model substantially outperforms all evaluated frontier baselines.



(a) Win rates on the OSM test set.

(b) Win rates on the synthetic test set.

Figure 27: Pairwise VLM judge outcomes for the FLOORA-0.6B (Qwen3) GRPO (RM+VR) model against frontier baselines. Bars show the fraction of prompts for which the FLOORA model wins, ties, or loses. Our model is preferred over all evaluated frontier baselines. Error bars indicate 95% Wilson intervals over the evaluation prompts.

Results are shown in Figure 10. The FLOORA-0.6B output is selected as the best layout in 89.3% of evaluations, substantially exceeding all frontier baselines.

Qualitative Comparisons. The comparisons in Appendix H.6.1 illustrate this distinction. While some frontier-model outputs pass automatic checks, they often exhibit weak corridor connectivity, irregular unit subdivisions, or implausible proportions. In contrast, our model more consistently produces coherent circulation, regular unit organization, and architecturally plausible layouts, suggesting that domain-specific representation and post-training improve both validity and design quality.

H.6 Qualitative Analysis

H.6.1 Qualitative Comparison with Frontier Models

Figures 28 and 29 compare FLOORA-0.6B GRPO (RM+VR) with frontier baselines on OSM and synthetic samples. Even when frontier outputs pass geometric and functional checks, they may remain architecturally impractical, with poor connectivity, irregular units, or implausible proportions.

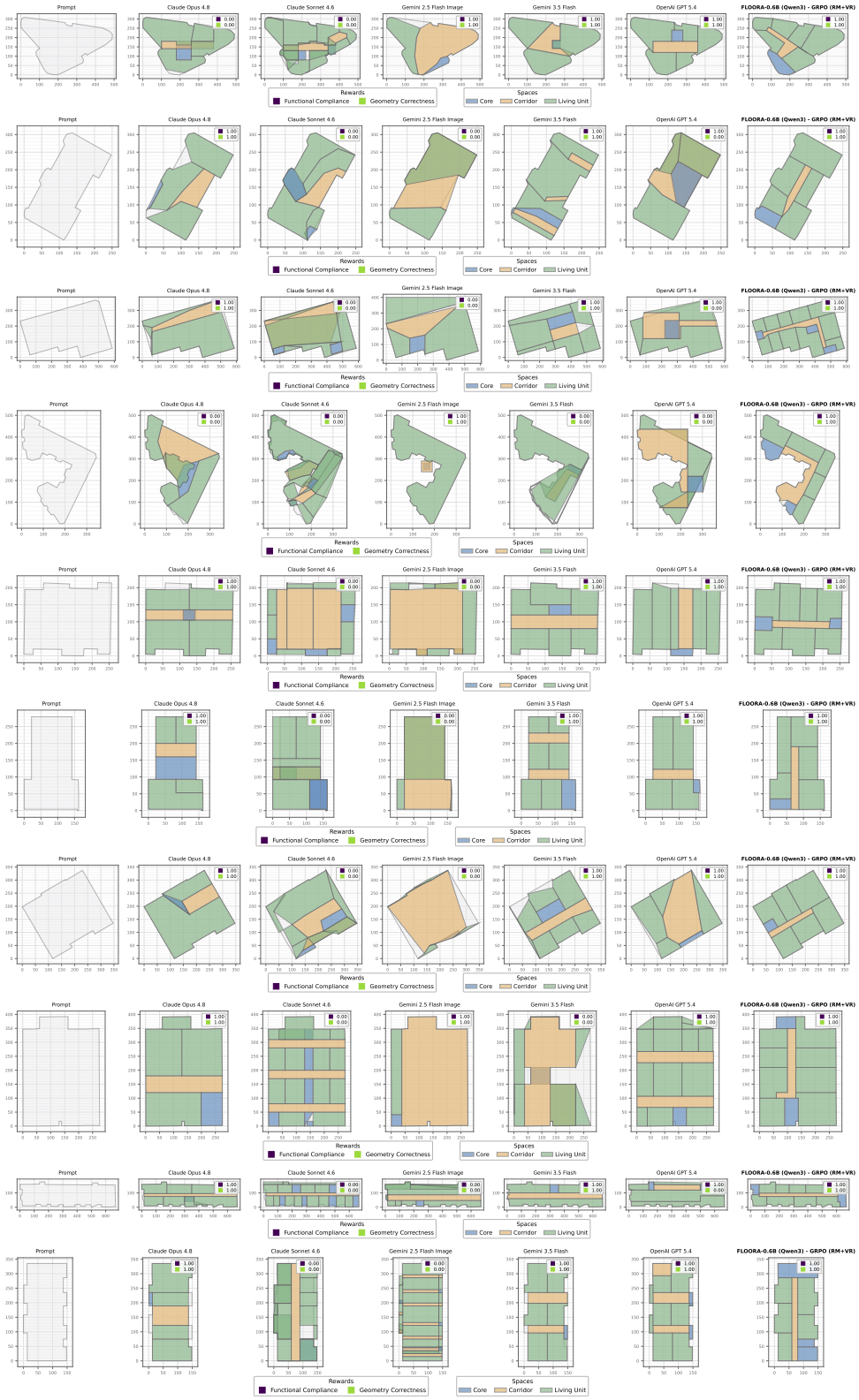


Figure 28: Qualitative comparison between the FLOORA-0.6B (Qwen3) GRPO (RM+VR) model and frontier models on the **OSM** test set.

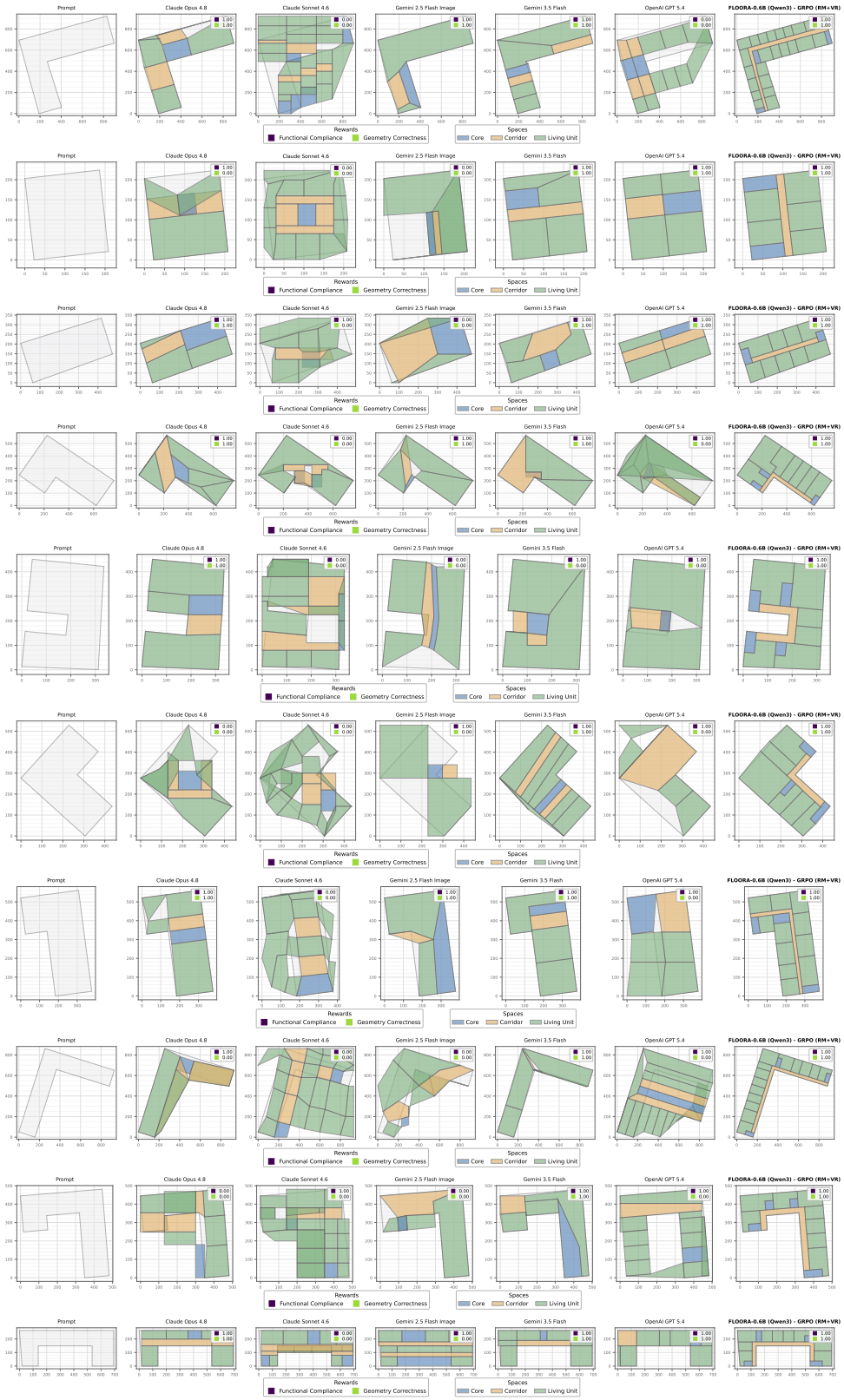


Figure 29: Qualitative comparison between the FLOORA-0.6B (Qwen3) GRPO (RM+VR) model and frontier models on the **synthetic** test set.

H.6.2 POST-TRAINING IMPROVEMENTS

Figures 30 and 31 show the qualitative improvements from SFT, GRPO (RM+VR), and GRPO (VR only) for the FLOORA-0.6B (Qwen3) model. Both GRPO variants substantially improve over the base and SFT models, producing more complete and geometrically regular layouts. However, GRPO (RM+VR) is preferable: it more consistently produces coherent space layouts, better unit sizes and proportions, better-placed cores with more appropriate counts, and more effective corridor positioning. This suggests that verifiable rewards enforce hard constraints, while the reward model adds an architectural-quality signal.

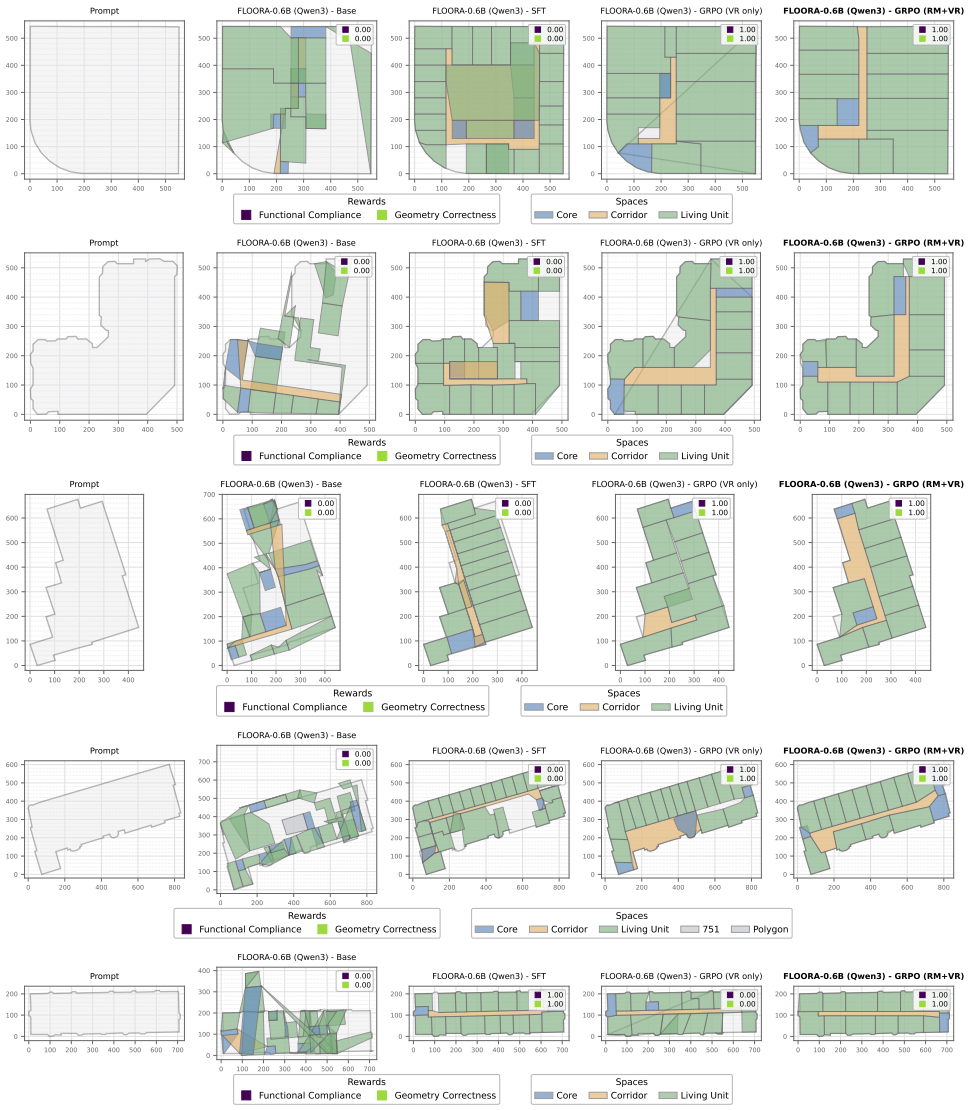


Figure 30: Qualitative comparison between the FLOORA-0.6B (Qwen3) Base, SFT, GRPO (RM+VR), and GRPO (VR only) models on the **OSM** test set.

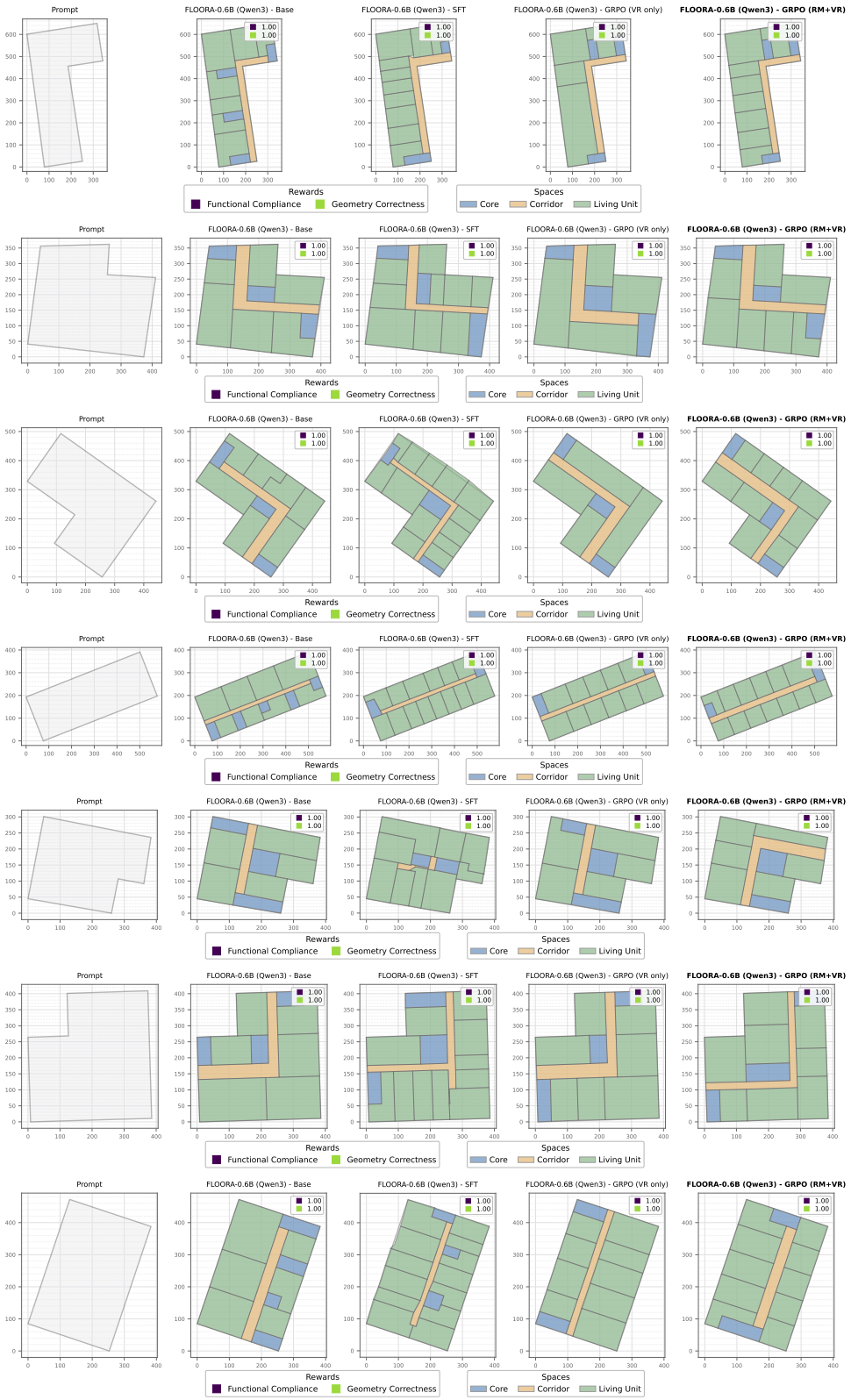


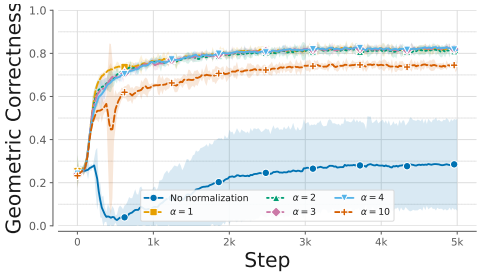
Figure 31: Qualitative comparison between the FLOORA-0.6B (Qwen3) Base, SFT, GRPO (RM+VR), and GRPO (VR only) models on the **synthetic** test set.

I Ablation Studies

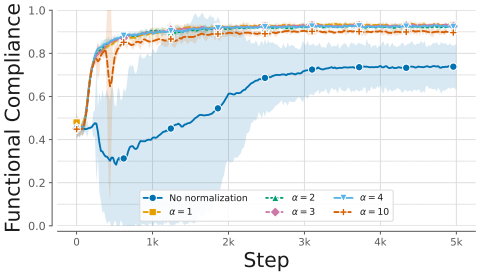
I.1 Ablation on Reward Model Normalization

As discussed in Section 6.3, the reward model produces an unconstrained scalar score, whereas the verifiable rewards are bounded to $[0, 1]$ by construction. Without normalization, the RM can therefore dominate the combined reward, effectively changing the relative importance of the learned preference signal and the verifiable geometric and functional constraints. We ablate the normalization scale α in Equation (1) to study this effect during GRPO training.

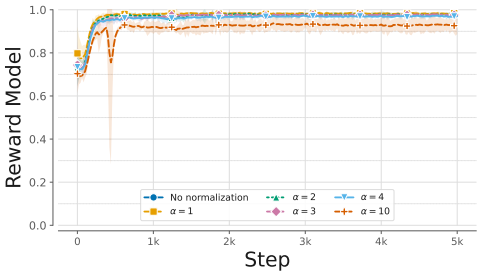
The results in Figure 32 and Table 14 show that normalization is important for stable optimization. The experiments were conducted on the FLOORA-0.6B (Qwen3) model on 3 training seeds. Without normalization, the RM score grows to a much larger magnitude than the verifiable rewards, while geometric correctness and functional compliance remain substantially worse. In contrast, normalized settings with $\alpha \in [1, 4]$ achieve consistently strong verifiable rewards. This indicates that a moderate normalization scale is sufficient to keep the learned reward aligned with the bounded verifiable rewards. Among the normalized settings, $\alpha \in [1, 4]$ performs consistently well. We select $\alpha = 3$ because it provides the best overall tradeoff between smooth RM learning dynamics and high verifiable rewards, while larger scales such as $\alpha = 10$ degrade both geometric correctness and functional compliance.



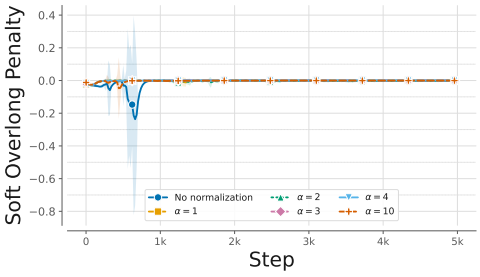
(a) Geometric correctness reward curves.



(b) Functional compliance reward curves.



(c) Reward model curves.



(d) Soft overlong penalty curves.

Figure 32: Ablation of the RM normalization scale α during GRPO training with the FLOORA-0.6B (Qwen3) model. Moderate scales $\alpha \in [1, 4]$ stabilize training and achieve high verifiable rewards. Shaded regions show 95% CIs across 3 training seeds.

Table 14: Final RM and verifiable reward scores for different normalization scales α, reported with 95% CIs across 3 training seeds.

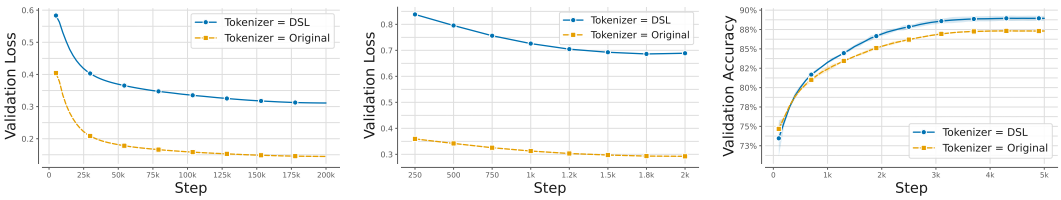
Setting	Reward Model	Geometric Correctness	Functional Compliance	Soft Overlong Penalty
No normalization	28.719 ± 5.376	0.383 ± 0.192	0.761 ± 0.224	0.000 ± 0.000
$\alpha = 1$	0.994 ± 0.002	0.853 ± 0.026	0.951 ± 0.016	0.000 ± 0.000
$\alpha = 2$	0.990 ± 0.012	0.846 ± 0.017	0.944 ± 0.002	0.000 ± 0.000
$\alpha = 3$	0.988 ± 0.011	0.851 ± 0.027	0.948 ± 0.007	0.000 ± 0.000
$\alpha = 4$	0.982 ± 0.015	0.850 ± 0.017	0.943 ± 0.012	0.000 ± 0.000
$\alpha = 10$	0.954 ± 0.049	0.771 ± 0.032	0.922 ± 0.026	0.000 ± 0.000

I.2 Ablation on Original and DSL Tokenizers

As discussed in Section 5 and Table 5, the DSL tokenizer substantially reduces sequence length and pre-training cost. With the original tokenizer, the per-device batch size must be halved, and pre-training takes approximately 80 hours on 16 GPUs. In comparison, the DSL tokenizer completes pre-training in approximately 55 hours on 8 GPUs.

Figures 33 and 34 compare the training dynamics between the original and DSL tokenizers. Token-level loss and perplexity are not directly comparable across tokenizers because they segment sequences differently. For example, the original Qwen tokenizer represents digits individually, whereas the DSL tokenizer uses a single token for each quantized numeric value. Reward-model accuracy is tokenizer-independent and is higher with the DSL tokenizer.

Downstream results in Tables 15 and 16 show that the DSL tokenizer substantially improves OSM performance, with pass@5 gains of approximately 6.5-7.2 percentage points for the GRPO models. On the synthetic test set, the original tokenizer performs slightly better, but the largest pass@5 difference is only 0.9 percentage points.



(a) Pre-training validation loss. (b) SFT validation loss. (c) RM validation accuracy.

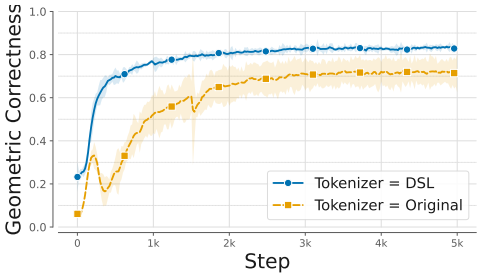
Figure 33: Tokenizer ablation for FLOORA-0.6B (Qwen3) over optimizer steps. Shaded regions show 95% CIs across 3 training seeds, except pre-training which uses a single seed.

Table 15: Pass@1/3/5 performance of FLOORA-0.6B (Qwen3) across training stages using the original and DSL tokenizers on the **OSM** test set, reported with 95% CIs across 3 training and 5 evaluation seeds. Best absolute value is shown in bold.

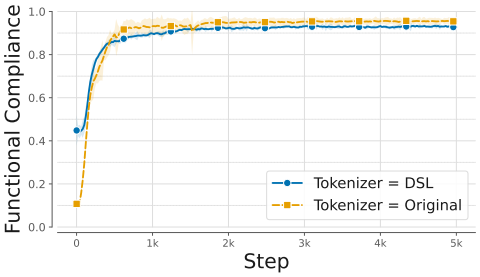
	Total Reward		
Model	@1	@3	@5
FLOORA-0.6B (Qwen3) – Base [DSL tok.]	23.8 ± 0.1	33.4 ± 0.4	38.1 ± 0.7
FLOORA-0.6B (Qwen3) – Base [Orig. tok.]	19.6 ± 0.4	27.3 ± 0.5	31.1 ± 0.5
FLOORA-0.6B (Qwen3) – GRPO (RM+VR) [DSL tok.]	$\mathbf{79.3 \pm 0.7}$	$\mathbf{91.3 \pm 0.5}$	$\mathbf{94.0 \pm 0.5}$
FLOORA-0.6B (Qwen3) – GRPO (RM+VR) [Orig. tok.]	71.6 ± 2.9	83.2 ± 2.0	86.8 ± 1.6
FLOORA-0.6B (Qwen3) – GRPO (VR only) [DSL tok.]	74.7 ± 0.7	89.0 ± 0.5	92.5 ± 0.5
FLOORA-0.6B (Qwen3) – GRPO (VR only) [Orig. tok.]	66.7 ± 0.5	81.5 ± 0.6	86.0 ± 0.6

Table 16: Pass@1/3/5 performance of FLOORA-0.6B (Qwen3) across training stages using the original and DSL tokenizers on the **synthetic** test set, reported with 95% CIs across 3 training and 5 evaluation seeds. Best absolute value is shown in bold.

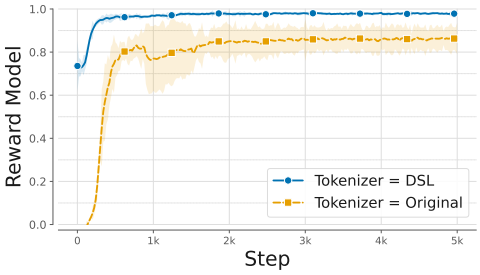
	Total Reward		
Model	@1	@3	@5
FLOORA-0.6B (Qwen3) – Base [DSL tok.]	90.2 ± 1.0	94.0 ± 1.0	94.9 ± 1.0
FLOORA-0.6B (Qwen3) – Base [Orig. tok.]	88.9 ± 0.7	95.1 ± 0.6	96.2 ± 0.6
FLOORA-0.6B (Qwen3) – GRPO (RM+VR) [DSL tok.]	96.8 ± 0.2	98.1 ± 0.2	98.3 ± 0.2
FLOORA-0.6B (Qwen3) – GRPO (RM+VR) [Orig. tok.]	93.5 ± 1.3	97.6 ± 0.5	98.4 ± 0.3
FLOORA-0.6B (Qwen3) – GRPO (VR only) [DSL tok.]	$\mathbf{97.3 \pm 0.1}$	98.1 ± 0.2	98.2 ± 0.2
FLOORA-0.6B (Qwen3) – GRPO (VR only) [Orig. tok.]	95.2 ± 0.3	$\mathbf{98.6 \pm 0.1}$	$\mathbf{99.1 \pm 0.1}$



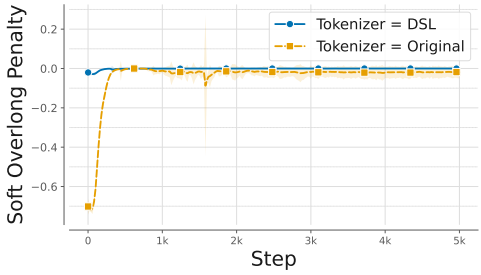
(a) Geometric correctness reward curves.



(b) Functional compliance reward curves.



(c) Reward model curves.



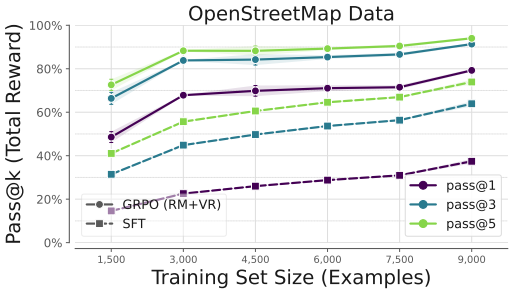
(d) Soft overlong penalty curves.

Figure 34: Tokenizer ablation during GRPO (RM+VR) training with the FLOORA-0.6B (Qwen3) model. Shaded regions show 95% CIs across 3 training seeds.

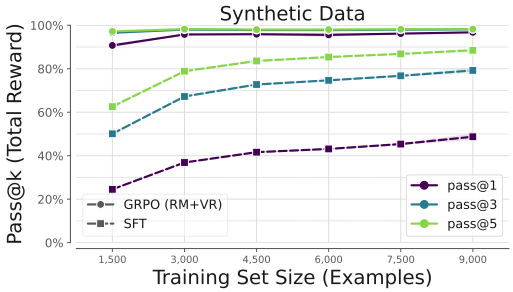
I.3 Ablation on Human Feedback Data Scale

As discussed in Section 6, human feedback is collected in multiple rounds, with the collection model post-trained between rounds. We evaluate cumulative chronological subsets in increments of 1,500 examples, from 1,500 to 9,000, retraining SFT, the reward model, and GRPO for each subset. Because data quantity is coupled with collection stage and collection-model quality, this analysis reflects the practical collection trajectory rather than isolating the effect of data quantity alone. All subset sizes refer to the number of samples before rotation augmentation (see Appendix C).

As shown in Figure 35, performance generally improves across the collection trajectory for both SFT and GRPO. The largest gains occur at 3,000 and 9,000 examples, particularly on OSM, while synthetic performance saturates earlier. Overall, marginal gains diminish as the dataset grows, suggesting that tracking performance during collection can help identify when additional expert annotation provides limited benefit.



(a) Pass@1/3/5 on the OSM test set.



(b) Pass@1/3/5 on the synthetic test set.

Figure 35: Human feedback data-scale ablation for FLOORA-0.6B (Qwen3) across SFT and GRPO (RM+VR) on the OSM and synthetic test sets. Shaded regions show 95% CIs across 3 training seeds and 5 evaluation seeds.