# FreeFEM-CodeGen: A QLoRA-Fine-Tuned Language Model for Domain-Specific Code Generation in Finite Element Simulations

**Anonymous ACL submission**


## Abstract

We present a two-stage QLoRA fine-tuning pipeline that adapts a general-purpose code model with around 7B parameters (such as Qwen2.5-Coder-7B, Qwen-7B for short) to generate specialized FreeFEM code for solving partial differential equations (PDEs) with the finite element method. Continued pretraining on 852 FreeFEM source files followed by supervised fine-tuning on 739 instruction–response pairs yields a Qwen-7B model with a 73% execution pass rate on 74 test prompts, up from 0% for the base model. The fine-tuned Qwen-7B model outperforms larger zero-shot models (such as Mistral Large 2.1 – 123B), and the pipeline generalizes across architectures: DeepSeek-Coder-6.7B reaches 79.7%, matching Gemini 3 Pro. Training for Qwen-7B completes in 67 minutes on a single A100 GPU. The training pipeline, the curated dataset, and an execution-based evaluation framework will be released freely.


## 1 Introduction

Scientific computing relies heavily on specialized software for numerical simulations. FreeFEM (Hecht, 2012) is a widely used domain-specific language that allows researchers to specify partial differential equations (PDEs) and obtain their numerical solutions using mathematical formulas (the variational form of PDEs) directly, without the tedious, detailed handling of elements, basis functions, integration rules, loops, and index management. Despite recent advances in code generation with large language models (Chen et al., 2021; Hui et al., 2024; Rozière et al., 2023), these models perform poorly on FreeFEM due to its minimal representation in standard training corpora.

Generating correct FreeFEM code presents several challenges. The language uses mathematical syntax distinct from mainstream programming languages and requires knowledge of weak formulations, finite element spaces, and mesh generation. It combines declarative PDE specifications with imperative control flow and demands both domain expertise and programming proficiency. Subtle errors in boundary conditions or solver parameters can produce incorrect numerical results despite syntactically valid code. Here is an example of FreeFEM code for solving a Poisson equation

$$-\Delta u = 1,$$

on a unit cube with mixed boundary conditions:

```
load "msh3"
int nn = 20; // Mesh quality
int[int] labs = [1, 2, 1, 2, 1, 2];
mesh3 Th = cube(nn, nn, nn, label=
    labs);
fespace Vh(Th, P1);
Vh u,v; // Trial and test functions
macro Grad(u) [dx(u), dy(u), dz(u)]
    //
// Define the weak form and solve
solve Poisson(u, v, solver=CG)
= int3d(Th)(Grad(u)' * Grad(v))
- int3d(Th)(1 * v)
+ on(1, u=0);
// Plot the solution
plot(u, nbiso=15);
```

This paper presents a two-stage fine-tuning approach to adapt general-purpose code models to FreeFEM. The pipeline consists of: (1) continued pretraining on 852 FreeFEM source files from the official repository to expose the model to idiomatic syntax and common patterns; and (2) supervised fine-tuning on 739 instruction-response pairs that cover diverse simulation tasks. QLoRA (Dettmers et al., 2023), a parameter-efficient method using 4-bit quantization and low-rank adapters, enables training Qwen2.5-Coder-7B on a single NVIDIA A100 GPU in under 67 minutes total.

For evaluation, we develop a hierarchical scoring framework that combines empirical code analysis with execution-based validation. We compile and execute each generated FreeFEM program, assigning scores at multiple levels: compilation, runtime error detection, and successful termination.

The execution outcome determines the final score. Code that fails to compile receives zero credit; code that executes successfully is scored with a 70/30 execution-empirical weighting. The test set comprises 74 prompts spanning seven categories: basic PDEs, time-dependent problems, mesh generation, boundary conditions, coupled systems, visualization, and solver configuration.

Our contributions are:

- A complete two-stage QLoRA pipeline for domain-specific code generation, with detailed data preprocessing and training configurations
- A curated FreeFEM dataset containing 852 pretraining examples and 739 instruction-following pairs across seven task categories
- An execution-based evaluation framework with automatic FreeFEM compilation and scoring
- Experimental results showing a 73% execution pass rate on Qwen2.5-Coder-7B, with cross-architecture validation showing that DeepSeek-Coder-6.7B reaches 79.7%, matching Gemini 3 Pro zero-shot.

## 2 Related Work

Code-specific language models such as Qwen2.5-Coder (Hui et al., 2024), CodeLlama (Rozière et al., 2023), StarCoder (Li et al., 2023), and DeepSeek-Coder (Guo et al., 2024) have advanced code generation for mainstream languages, but scientific domain-specific languages (DSLs) receive little coverage. Low-Rank Adaptation (LoRA) (Hu et al., 2021) and its quantized variant QLoRA (Dettmers et al., 2023) enable parameter-efficient fine-tuning; we additionally use rank-stabilized LoRA (rsLoRA) (Kalajdzievski, 2023) for stability.

Adapting LLMs to DSLs remains challenging due to data scarcity and evaluation gaps (Joel et al., 2024). Recent work includes grammar-guided prompting (Wang et al., 2023) and DomCoder (Gu et al., 2024), which injects external domain knowledge. For scientific computing, CodePDE (Li et al., 2025) generates PDE solvers with self-refinement, while AutoFreeFEM (Allaire and Gfrerer, 2024) is a classical (non-machine-learning) Python package that generates FreeFEM code for shape optimization through symbolic differentiation in SymPy. Our work fine-tunes language models on diverse FreeFEM examples rather than relying on symbolic computation. We follow the instruction-tuning paradigm, using ideas from Evol-Instruct (Luo et al., 2024) and OSS-Instruct (Wei et al., 2024) to generate FreeFEM training pairs from existing code. Our evaluation framework extends the execution-based methodology of HumanEval (Chen et al., 2021) to the FreeFEM environment.

## 3 Methodology

### 3.1 Data Collection and Preprocessing

We construct two datasets corresponding to the continued pretraining and supervised fine-tuning stages of our pipeline.

**Continued Pretraining Corpus** We collect 852 FreeFEM source files from the official FreeFEM v4.15 repository. These include example scripts (.edp), include files (.idp), and documentation samples. We filter the corpus to remove duplicates, non-executable fragments, and examples that depend on unavailable plugins. To provide contextual information during training, we prepend a lightweight metadata header to each file in the style of YAML front matter: a structured block delimited by `---` markers that records the source file path and document type (code or documentation). This format follows StarCoder (Li et al., 2023), which prepends repository names and file paths to training documents. Yu et al. (Yu et al., 2025) show that such metadata conditioning accelerates pretraining and improves downstream performance. We separate documents with an `<|endoftext|>` token to mark boundaries. The resulting corpus contains 2.5 MB of text spanning 107,000 lines of FreeFEM code and covers problems from canonical Poisson equations to multi-physics simulations.[1]

**Supervised Fine-Tuning Dataset** We construct 739 training examples, 85 validation examples, and 74 test examples in instruction-response format. Examples are derived from two sources: (1) manual curation from the official FreeFEM GitHub repository, documentation, and community resources; and (2) for existing FreeFEM codes without descriptions, we employ AI-assisted instruction generation using Gemini 2.0 Flash, where we provide the FreeFEM code snippets, generate corresponding natural language instructions, and then have humans verify them.

[1] We will release all datasets and code upon publication.

The dataset spans seven categories: (1) `pde_basic`: canonical elliptic and parabolic PDEs (Poisson, Laplace, heat) with Dirichlet/Neumann boundary conditions; (2) `mesh`: mesh generation, adaptive refinement, and geometry construction; (3) `boundary`: boundary condition specification and mixed-type constraints; (4) `pde_timedep`: time-dependent problems with explicit/implicit temporal schemes; (5) `systems`: coupled PDEs including Stokes flow and linear elasticity; (6) `visualization`: post-processing and data export; (7) `solver`: iterative methods, preconditioners, and convergence parameters.

Each example consists of a natural language prompt that describes the simulation task and a complete FreeFEM script as the target output. Prompts range from abstract specifications ("solve the heat equation on a square domain") to detailed requirements with explicit parameters and boundary conditions.

### 3.2 Two-Stage Training Pipeline

The complete structure of the training pipeline is shown in Figure 1.

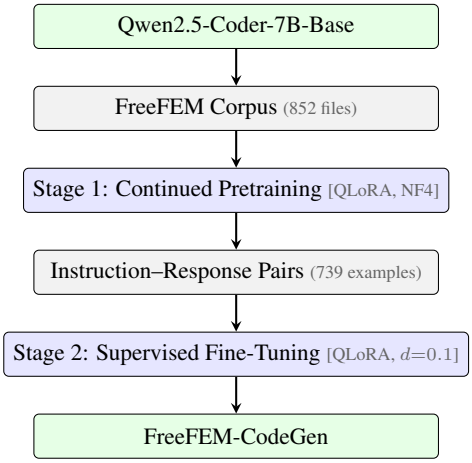


Figure 1: Two-stage training pipeline: continued pretraining on FreeFEM source files followed by supervised fine-tuning on instruction-response pairs.

**Stage 1: Continued Pretraining** We initialize from the Qwen2.5-Coder-7B base model and perform continued pretraining on the FreeFEM corpus using QLoRA (Dettmers et al., 2023). Our configuration uses 4-bit NormalFloat (NF4) quantization with double quantization, targeting all linear projection layers (q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, down_proj) in each transformer block. Table 1 details the complete hyperparameter configuration.

Table 1: QLoRA hyperparameters for continued pretraining (CPT) and supervised fine-tuning (SFT). Both stages share the same quantization, LoRA rank, learning rate, and batch configuration. They differ in dropout, number of epochs, and loss computation.

| **Hyperparameter** | **CPT** | **SFT** |
|---|---|---|
| Quantization | 4-bit NF4 + double quant | |
| LoRA rank ($r$) | 64 | |
| LoRA alpha ($\alpha$) | 128 | |
| LoRA scaling | rsLoRA ($\alpha/\sqrt{r}$) | |
| Target modules | q/k/v/o/gate/up/down_proj | |
| Epochs | 3 | 2 |
| Learning rate | $2 \times 10^{-4}$ | |
| LR schedule | Cosine + warmup (100 steps) | |
| Batch size | 4 | |
| Gradient accumulation | 4 (effective batch size: 16) | |
| Max sequence length | 2048 tokens | |
| Dropout | 0.05 | 0.1 |
| Loss computation | Full text | Completions only |
| Trainable params | 161M / 7B (2.3%) | |
| Training time | ∼45 min | ∼22 min |
| Peak VRAM | ∼38GB | |

The continued pretraining stage exposes the model to FreeFEM syntax, idioms, and domain patterns. Training requires approximately 45 minutes on a single NVIDIA A100 40GB GPU.

**Stage 2: Supervised Fine-Tuning** Beginning from the continued pretraining checkpoint, we fine-tune on instruction-response pairs using the same QLoRA configuration with dropout increased to 0.1. We train for 2 epochs with identical optimization settings. Supervised fine-tuning requires approximately 22 minutes, yielding a total training time of 67 minutes.

**Training Objective** Both stages minimize the standard next-token cross-entropy loss. Given a sequence $\mathbf{x} = (x_1, \ldots, x_N)$, the training objective is defined as follows:

$$\mathcal{L} = -\frac{1}{|\mathcal{T}|} \sum_{i \in \mathcal{T}} \log P_\theta(x_i \mid x_1, \ldots, x_{i-1}), \quad (1)$$

where $\theta$ denotes the trainable LoRA adapter parameters and $\mathcal{T}$ is the set of token positions that contribute to the loss. In continued pretraining, $\mathcal{T} = \{1, \ldots, N\}$, so every token is predicted. In supervised fine-tuning, the loss is restricted to the assistant-response portion of each example. Instruction-prompt tokens are masked (labels set to $-100$) so that only assistant-response tokens contribute to the gradient.

Our QLoRA configuration results in 161M trainable parameters (2.3% of the 7B total), with

peak GPU memory consumption of approximately 38 GB. This parameter efficiency enables rapid experimentation on single-GPU infrastructure.

### 3.3 Evaluation Framework

We design a hierarchical scoring system that evaluates generated code at increasing levels of correctness. Since execution-based validation is the strongest indicator of functional quality (Chen et al., 2021), we condition the combined score on the execution outcome and reserve empirical credit as a secondary signal for partially correct solutions; see Figure 2. This multi-level design follows the hierarchical evaluation strategy used in scientific code generation (Li et al., 2025).

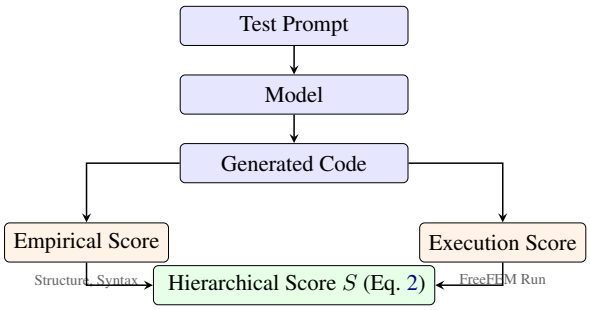


Figure 2: Dual evaluation framework combining empirical analysis with execution-based validation.

**Empirical Scoring** The empirical component $s_{\text{empi}} \in [0, 1]$ evaluates syntactic structure and FreeFEM-specific patterns without requiring compilation. We assign up to 130 raw points across five categories, then normalize to $[0, 1]$:

- **Code block formatting (30 pts):** Presence of properly delimited markdown code blocks, verifying that the model follows the expected output format.
- **FreeFEM keywords (up to 30 pts):** Presence of domain primitives from a vocabulary of 31 keywords (`mesh`, `fespace`, `solve`, `varf`, `border`, `int2d`, `dx`, etc.) at 2 points each, capped at 30.
- **Structural components (30 pts):** Mesh definition (10), finite element space declaration (10), and problem or variational formulation (10).
- **Code quality (25 pts):** Reasonable code length, between 5 and 100 lines (20), and presence of output statements such as `plot` or `cout` (5).
- **Syntax validity (15 pts):** Balanced delimiters, i.e., parentheses, braces, and brackets (10), and absence of garbage tokens in the response (5).

As an example, a generated script containing a mesh definition with `square`, an `fespace` declaration, a `solve` block, and a `plot` call within a properly formatted code block of 40 lines would score approximately $126/130 = 0.97$. This empirical score captures idiomatic code structure and remains informative even when execution infrastructure is unavailable.

**Execution Scoring** The execution component validates functional correctness by compiling and running each generated script through the `FreeFem++` binary with a 30-second timeout. We assign execution scores at four levels:

- $s_{\text{exec}} = 1.0$: exit code 0, no error patterns detected (success).
- $s_{\text{exec}} = 0.8$: exit code 0 with compiler warnings (partial success).
- $s_{\text{exec}} = 0.3$: error patterns detected in output (runtime error).
- $s_{\text{exec}} = 0.0$: non-zero exit code or timeout (compilation failure).

Error detection uses pattern matching against FreeFEM-specific diagnostics including syntax errors, undefined variables, type mismatches, and plugin load failures.

**Hierarchical Combined Score** We combine the two components using a hierarchical scheme that conditions on the execution outcome:

$$S = \begin{cases} 0 & s_{\text{exec}} = 0 \\ 0.3\, s_{\text{empi}} & s_{\text{exec}} = 0.3 \\ 0.7\, s_{\text{exec}} + 0.3\, s_{\text{empi}} & \text{otherwise} \end{cases} \quad (2)$$

Code that fails to compile receives zero credit regardless of syntactic plausibility. Code with runtime errors receives only empirical partial credit, capped at 0.3. Code that executes successfully is scored with a 70/30 execution-empirical weighting. This hierarchy reflects the observation that syntactically plausible code frequently fails at runtime (Section 5), making execution the dominant quality signal. We also report the **pass rate**, defined as the fraction of prompts achieving $s_{\text{exec}} \geq 0.8$.

## 4 Experimental Setup

**Models** We evaluate four configurations of our fine-tuning pipeline: (1) **Base**: Qwen2.5-Coder-7B-Base with zero-shot prompting; (2) **+CPT**: Base model after continued pretraining on FreeFEM

code; (3) **SFT only**: Base model fine-tuned directly on instruction-response pairs without CPT; (4) **+CPT+SFT**: full pipeline with continued pretraining followed by supervised fine-tuning.

We also apply the same pipeline to **CodeLlama-7B** (Rozière et al., 2023) and **DeepSeek-Coder-6.7B** (Guo et al., 2024) with identical hyperparameters (Table 1).

As frontier model baselines, we evaluate **Gemini 3 Flash** (Google DeepMind, 2025), **Gemini 3 Pro** (Google DeepMind, 2025), **Claude Sonnet 4.5** (Anthropic, 2025), and **Mistral Large 2.1** (123B) (Mistral AI, 2024) on the same test set using their respective APIs. We use a minimal system prompt ("When providing code, wrap it in a fenced code block.") that provides no FreeFEM-specific examples or domain knowledge. This evaluates zero-shot frontier performance. Few-shot prompting may improve their results. All other generation parameters (temperature 0.3, max tokens 2048–4096) match those used in our fine-tuned model evaluations.

**Test Set** We constructed the evaluation set independently of the training data, with no overlap between test prompts and training instructions. It contains 74 prompts spanning seven categories: basic PDEs (26), mesh generation (20), boundary conditions (8), time-dependent PDEs (7), coupled systems (6), visualization (5), and solver configuration (2). Approximately 60% are classified as easy difficulty (single equation, standard conditions), while 40% are medium difficulty (coupled systems, complex geometries, time-dependent formulations). All prompts have verified solutions in FreeFEM documentation or example repositories.

**Metrics** We report three primary metrics: (1) **Average score**: mean combined score across all test prompts; (2) **Pass rate**: percentage of prompts producing code that compiles and executes without errors; (3) **Empirical score**: mean normalized empirical score based on syntactic validation. Per-category breakdowns identify performance patterns across problem types. We also report statistical pass@$k$ estimates (Chen et al., 2021) by generating $N$=64 samples per prompt at temperature 0.6 and computing the unbiased estimator of Chen et al.

**Infrastructure** All experiments ran on the NCSA Delta supercomputer using NVIDIA A100 40GB GPUs. Training used Hugging Face Transformers 4.36, PyTorch 2.1, and PEFT for QLoRA implementation. Inference applied 8-bit quantization for efficiency. CPT required approximately 45 minutes and SFT approximately 22 minutes on a single A100.

## 5 Results

Table 2 shows overall performance across the four model configurations. The base Qwen2.5-Coder-7B model achieves a 0% execution pass rate despite a high empirical score (0.88), indicating that it generates syntactically plausible FreeFEM code that fails every execution test. The base model also exhibits degenerate behavior, leaking training artifacts (e.g., README text) into generated outputs.

Continued pretraining alone produces counter-intuitive results: the empirical score drops sharply to 0.06 while the execution pass rate rises to 5.4%. CPT converts the base model into a completion model that generates raw FreeFEM code without instruction-following structure. The empirical scorer rejects most outputs due to missing code block formatting. However, a small number of completions happen to produce valid executable scripts.

SFT applied directly to the base model without CPT reaches a 67.6% pass rate, 5.4 percentage points below the full CPT+SFT pipeline. CPT exposure to FreeFEM syntax before instruction tuning accounts for this gap.

Supervised fine-tuning recovers instruction-following behavior and combines it with domain knowledge from CPT. The full pipeline (+CPT+SFT) reaches a 73.0% pass rate with an average score of 0.788. The gap between empirical scores (∼0.96) and execution rates (0.73) indicates that syntactic plausibility does not guarantee semantic correctness.

Table 2: Overall model comparison across pipeline stages for Qwen2.5-Coder-7B.

| Model | Avg Score | Pass Rate | Empirical |
|---|---|---|---|
| Qwen-7B | 0.263 | 0.0% (0/74) | 0.88 |
| + CPT | 0.057 | 5.4% (4/74) | 0.06 |
| SFT only | 0.749 | 67.6% (50/74) | 0.93 |
| + CPT + SFT | **0.788** | **73.0% (54/74)** | **∼0.96** |

Table 3 breaks down performance by category. Solver configuration achieves perfect accuracy (100%), likely because the solver APIs follow predictable patterns with limited variation. Time-dependent PDEs (85.7%) and visualization (80.0%)

perform well, while boundary conditions and mesh generation cluster around 75%. Basic PDEs, despite being the largest category, achieve only a 69.2% pass rate. Coupled systems prove most difficult at 50%. This reflects the complexity of multi-variable formulations and compatibility requirements between finite element spaces.

Table 3: Per-category results for the full pipeline (+CPT+SFT) for Qwen2.5-Coder-7B. Systems involving coupled PDEs remain the most challenging.

| Category | Count | Pass Rate | Avg Score |
|---|---|---|---|
| `solver` | 2 | **100.0%** | **1.000** |
| `pde_timedep` | 7 | 85.7% | 0.891 |
| `visualization` | 5 | 80.0% | 0.846 |
| `boundary` | 8 | 75.0% | 0.815 |
| `mesh` | 20 | 75.0% | 0.765 |
| `pde_basic` | 26 | 69.2% | 0.775 |
| `systems` | 6 | 50.0% | 0.645 |

**Cross-Architecture Comparison** Table 4 compares three base models under the same fine-tuning pipeline. DeepSeek-Coder-6.7B reaches the highest pass rate at 79.7%, followed by Qwen2.5-Coder-7B at 73.0% and CodeLlama-7B at 70.3%. All three outperform every frontier model except Gemini 3 Pro. Category-level results differ across architectures: Qwen leads on time-dependent PDEs (0.891 vs. 0.717 for DeepSeek), DeepSeek leads on mesh generation and basic PDEs, and CodeLlama trails on visualization (0.569) but matches the others on boundary conditions. All three models score 0.650 on solver tasks, a shared ceiling that likely reflects the training data rather than base model capability.

Table 4: Cross-architecture comparison. All models fine-tuned with the identical CPT+SFT pipeline and hyperparameters. Inference uses greedy decoding with no few-shot examples.

| Base Model | Avg Score | Pass Rate |
|---|---|---|
| CodeLlama-7B FT | 0.773 | 70.3% (52/74) |
| Qwen2.5-Coder-7B FT | 0.788 | 73.0% (54/74) |
| DeepSeek-Coder-6.7B FT | **0.827** | **79.7% (59/74)** |

**Frontier Model Comparison** Table 5 compares our fine-tuned models against four frontier models evaluated zero-shot. Claude Sonnet 4.5 and Gemini 3 Flash achieve similar execution pass rates (41.9% and 40.5%). Mistral Large 2.1 (123B parameters) reaches 60.8%, yet our Qwen-based 7B model outperforms all three at 73.0%. Under the same pipeline, DeepSeek-Coder-6.7B reaches 79.7%, matching Gemini 3 Pro's zero-shot rate.

The empirical-execution gap varies across models. Gemini 3 Flash and Mistral Large 2.1 both achieve high empirical scores (0.89 and 0.90) but substantially lower execution rates. Their generated code appears syntactically plausible yet fails at runtime. Claude Sonnet 4.5 shows a lower empirical score (0.67), partly because its verbose responses cause the code extractor to capture markdown artifacts instead of FreeFEM code. Gemini 3 Pro maintains strong empirical quality (0.87) with correspondingly high execution rates.

Per-category analysis reveals that Claude Sonnet 4.5 and Gemini 3 Flash both score 0% on coupled systems, where FreeFEM-specific API knowledge (e.g., Taylor-Hood element pairs) is essential. Claude Sonnet 4.5 additionally fails all time-dependent PDEs (0%), while Gemini 3 Flash achieves 85.7% on that category. Gemini 3 Pro achieves 100% on time-dependent PDEs and 83.3% on coupled systems. It is the only frontier model to substantially succeed on multi-physics problems. Our Qwen-based model achieves 50% on coupled systems. Fine-tuning helps, but the most complex multi-physics tasks still benefit from the broader reasoning capabilities of larger models.

Table 5: Comparison with zero-shot frontier models. DeepSeek-Coder-6.7B FT matches Gemini 3 Pro at 79.7% pass rate.

| Model | Avg Score | Pass Rate |
|---|---|---|
| Claude Sonnet 4.5 | 0.485 | 41.9% (31/74) |
| Gemini 3 Flash | 0.547 | 40.5% (30/74) |
| Mistral Large 2.1 | 0.690 | 60.8% (45/74) |
| Qwen2.5-Coder-7B FT | 0.788 | 73.0% (54/74) |
| DeepSeek-Coder-6.7B FT | **0.827** | **79.7% (59/74)** |
| Gemini 3 Pro | 0.798 | 79.7% (59/74) |

Table 6 reports training dynamics. CPT shows higher loss (0.965 train, 1.134 eval) as the model adapts to domain-specific patterns. SFT achieves lower loss (0.735 train, 0.812 eval), confirming successful instruction-following behavior. The modest train-eval gap suggests QLoRA regularization mitigates overfitting despite the small dataset.

Table 6: Training metrics for the two-stage pipeline on a single A100 GPU for Qwen2.5-Coder-7B.

| Stage | Train Loss | Eval Loss | Duration |
|---|---|---|---|
| CPT | 0.965 | 1.134 | ∼45 min |
| SFT | **0.735** | **0.812** | ∼22 min |

**Statistical Pass@$k$** Table 7 reports pass@$k$ estimates using 64 samples per prompt at temperature 0.6. pass@1 is 76.2%, close to the greedy pass rate. pass@5 reaches 96.6% and pass@10 reaches 98.2%. The model produces at least one correct solution for nearly all prompts given a small sample budget. Per-category pass@1 follows the same ranking as greedy evaluation: coupled systems remain the weakest category at 57.0% pass@1, while time-dependent PDEs lead at 90.0%.

Table 7: Statistical pass@$k$ for Qwen2.5-Coder-7B FT ($N$=64, temperature 0.6).

| pass@1 | pass@5 | pass@10 |
|---|---|---|
| 76.2% | 96.6% | 98.2% |

**Data Efficiency** To assess the relationship between training set size and model capability, we train on 10%, 25%, 50%, 75%, and 100% of the 739 SFT examples (74, 185, 370, 554, and 739 examples) with three random seeds each. All other hyperparameters remain fixed at the values in Table 1. Figure 3 reports mean pass@$k$ with standard deviation across seeds. Performance scales monotonically with training data. pass@1 rises from 12.4% at 74 examples to 75.3% at 739 examples. The curve exhibits diminishing returns: the first 111 additional examples (74→185) yield a 15.7 pp gain, while the last 185 examples (554→739) yield 10.1 pp. Variance decreases with scale. At 74 examples, pass@1 standard deviation is 8.3 pp. One seed produces only 3.6% pass@1 while the other two reach 13.6% and 20.0%. At 739 examples, the standard deviation drops to 2.2 pp. pass@5 and pass@10 approach saturation earlier: pass@5 exceeds 93% at 554 examples (75% of the data). The model has not saturated at 739 examples for pass@1, and additional training data would likely yield further gains. Per-category breakdowns (Appendix B) show that coupled systems and solvers remain the weakest categories at all data sizes, while boundary conditions and basic PDEs benefit most from additional data.

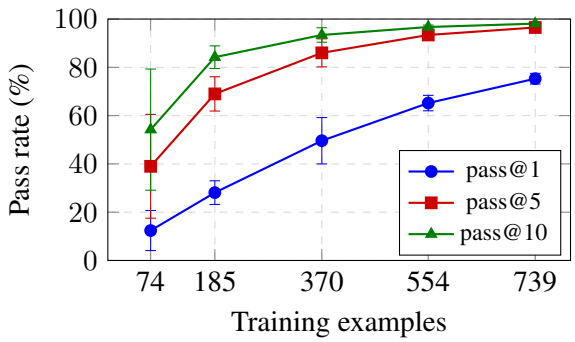


Figure 3: Data scaling ablation: pass@$k$ versus training set size ($N$=64, temperature 0.6, three seeds). Error bars show ±1 standard deviation.

**LoRA Rank Sensitivity** We train with LoRA ranks $r \in \{8, 16, 32, 64\}$ and set $\alpha = 2r$ to maintain the same ratio as our default configuration. rsLoRA (Kalajdzievski, 2023) is enabled throughout, so the effective scaling factor is $\alpha/\sqrt{r} = 2\sqrt{r}$. Table 8 reports mean pass@$k$ across three seeds.

The results reveal a threshold effect rather than gradual improvement. Ranks 8 and 16 produce near-identical performance (5.6% and 6.6% pass@1), indicating insufficient adapter capacity for this task. Rank 32 provides a 4× jump to 27.7% pass@1. Rank 64 yields another large gain to 74.2%, approaching the full model's 75.3% pass@1 with all 739 training examples. The transition from $r$=16 to $r$=64 represents an 11× improvement in pass@1.

Rank 64 achieves the lowest variance across seeds (0.3 pp standard deviation on pass@1). Rank 32 shows the highest (3.8 pp). This suggests that rank 64 provides sufficient capacity for stable convergence, while rank 32 operates near a critical threshold where small initialization differences produce larger outcome variation. Even at $r$=32, the model generates at least one correct solution for 68.2% of prompts given five attempts (pass@5). The $r$=64 configuration uses the same hyperparameters as the $n$=739 data scaling runs with different random seeds. The 1.1 pp difference between their mean pass@1 (74.2% vs. 75.3%) confirms experiment consistency. Full per-seed and per-category results appear in Appendix B.

Table 8: LoRA rank ablation ($\alpha$=2$r$, rsLoRA enabled, $N$=64, three seeds). Ranks below 32 lack sufficient adapter capacity for this task (Qwen2.5-Coder-7B).

| Rank ($r$) | pass@1 | pass@5 | pass@10 |
|---|---|---|---|
| 8 | $5.6 \pm 0.4$ | $23.6 \pm 1.1$ | $39.3 \pm 1.3$ |
| 16 | $6.6 \pm 0.8$ | $26.9 \pm 3.0$ | $43.6 \pm 4.1$ |
| 32 | $27.7 \pm 3.8$ | $68.2 \pm 4.8$ | $82.7 \pm 4.7$ |
| 64 | $\mathbf{74.2 \pm 0.3}$ | $\mathbf{95.8 \pm 0.9}$ | $\mathbf{97.4 \pm 0.9}$ |

## 6 Analysis

The observed gap between empirical scores and execution pass rates (Section 5) shows that models can produce syntactically plausible code without grasping the underlying numerical constraints. The most challenging category is coupled PDE systems. A coupled system is a set of PDEs in which multiple unknown fields (e.g., velocity and pressure) appear in several equations and cannot be solved independently (Boffi et al., 2013). Stokes flow, for instance, demands compatible finite element pairs (e.g., P2-P1 Taylor-Hood elements satisfying the inf-sup condition); choosing an incompatible pair leads to non-physical oscillations. Linear elasticity involves vector-valued displacement fields with proper stress-strain tensor formulations. These multi-field problems expose the model's difficulty with cross-variable dependencies and compatibility constraints, explaining the 19-percentage-point gap between coupled systems (50%) and the next-lowest category (basic PDEs, 69.2%). In contrast, solver configuration and visualization benefit from well-represented, fixed-syntax APIs. A detailed breakdown of typical failure modes (undefined variables, plugin hallucinations, type mismatches, etc.) is provided in Appendix C.

## 7 Limitations

This work has several limitations. First, our test set contains only 74 prompts; while pass@$k$ estimates with 64 samples give distributional insights, a larger and more diverse test set would increase confidence. We evaluate three 7B-class base models; other sizes (1B, 14B+) may yield different results. Frontier model evaluations are zero-shot; few-shot prompting could improve their performance.

Second, our execution-based evaluation is binary: code either runs without errors or fails. We do not verify numerical correctness. A script can execute successfully but produce incorrect solutions due to wrong discretizations or boundary conditions. Proper validation would require comparison with analytical or reference solutions.

Third, the approach is specific to FreeFEM. Adapting the pipeline to other FEM frameworks (FEniCS, deal.II) would require new data and likely framework-specific modifications; we do not investigate cross-framework transfer. Finally, our test prompts focus on canonical PDEs of moderate complexity. We do not evaluate performance on large-scale, production-type simulations involving complex geometries, adaptive refinement, or parallel solvers.

## 8 Conclusion

We presented a two-stage QLoRA pipeline that adapts code LLMs to FreeFEM. Continued pretraining on 852 FreeFEM source files and supervised fine-tuning on 739 instruction-response pairs give a Qwen2.5-Coder-7B model a 73% execution pass rate on 74 test prompts (pass@5 of 96.6%). The full training run completes in 67 minutes on a single A100 GPU. The 7B model outperforms zero-shot commercial models many times its size, and the pipeline generalizes to DeepSeek-Coder-6.7B (79.7%) and CodeLlama-7B (70.3%). Our evaluation reveals a persistent gap between syntactic plausibility and functional correctness. Ablation studies show that pass@1 scales monotonically with training data and that a LoRA rank of at least 32 is required for this task, providing practical guidance for adapting language models to other scientific DSLs. Future work includes larger training corpora, reinforcement learning with execution feedback, numerical correctness validation, and extending the approach to other FEM frameworks.

## Code and Data Availability

All source code, model weights, and datasets will be made publicly available. The training pipeline and evaluation framework will be released under the Apache 2.0 License. Fine-tuned model weights will be hosted on Hugging Face. The curated FreeFEM dataset (852 CPT files and 739 SFT pairs) will be released under CC BY 4.0.

## Acknowledgments

This work used the Delta advanced computing resource at the National Center for Supercomputing Applications (NCSA), which is supported by the National Science Foundation.
ChatGPT is used to fix grammatical errors.

## References

Grégoire Allaire and Michael H Gfrerer. 2024. Autofreefem: Automatic code generation with FreeFEM and LaTeX output for shape and topology optimization of non-linear multi-physics problems. *Structural and Multidisciplinary Optimization*, 67:213.

Anthropic. 2025. Claude: Anthropic's ai assistant. https://www.anthropic.com/claude. Claude Sonnet 4.5 evaluated via Claude Code CLI.

Daniele Boffi, Franco Brezzi, and Michel Fortin. 2013. *Mixed Finite Element Methods and Applications*, volume 44 of *Springer Series in Computational Mathematics*. Springer.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36.

Google DeepMind. 2025. Gemini 3. https://deepmind.google/models/gemini/. Gemini 3 Flash and Gemini 3 Pro evaluated via Google GenAI API.

Xiaodong Gu, Meng Chen, Yalan Lin, Yuhan Hu, Hongyu Zhang, Chengcheng Wan, Zhao Wei, Yong Xu, and Juhong Wang. 2024. On the effectiveness of large language models in domain-specific code generation. *ACM Transactions on Software Engineering and Methodology*. ArXiv:2312.01639.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y Wu, YK Li, and 1 others. 2024. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

Frédéric Hecht. 2012. New development in freefem++. *Journal of Numerical Mathematics*, 20(3-4):251–265.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, and 1 others. 2024. Qwen2.5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Sathvik Joel, Jie JW Wu, and Fatemeh H Fard. 2024. A survey on llm-based code generation for low-resource and domain-specific programming languages. *ACM Transactions on Software Engineering and Methodology*. ArXiv:2410.03981.

Damjan Kalajdzievski. 2023. A rank stabilization scaling factor for fine-tuning with lora. *arXiv preprint arXiv:2312.03732*.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, and 1 others. 2023. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*.

Shanda Li, Tanya Marwah, Junhong Shen, Weiwei Sun, Andrej Risteski, Yiming Yang, and Ameet Talwalkar. 2025. Codepde: An inference framework for llm-driven pde solver generation. *arXiv preprint arXiv:2505.08783*.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2024. Wizardcoder: Empowering code large language models with evol-instruct. In *Proceedings of the Twelfth International Conference on Learning Representations (ICLR)*.

Mistral AI. 2024. Mistral large 2. https://mistral.ai/news/mistral-large-2407. Mistral Large 2.1 (123B, `mistral-large-2411`) evaluated via Mistral API.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, and 1 others. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Bailin Wang, Zi Wang, Xuezhi Wang, Yuan Cao, Rif A Saurous, and Yoon Kim. 2023. Grammar prompting for domain-specific language generation with large language models. In *Advances in Neural Information Processing Systems*, volume 36.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. 2024. Magicoder: Empowering code generation with oss-instruct. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, pages 52632–52657.

Tianyu Yu, Aixin Chen, and Shafiq Joty. 2025. Metadata conditioning accelerates language model pre-training. *Proceedings of the 42nd International Conference on Machine Learning (ICML)*. ArXiv:2501.01956.

# A Training Data Examples

We provide representative examples from both training stages. All data will be released on Hugging Face upon publication.

## A.1 CPT Corpus Format

Each document in the continued pretraining corpus is preceded by a metadata header and separated by an `<|endoftext|>` token. The following is a representative entry (a Poisson solver with periodic boundary conditions):

```
<|endoftext|>
---
source: examples_tutorial_Periodic.edp
type: code
---
mesh Th=square(10,10,[2*x*pi,2*y*pi]);
fespace Vh(Th,P2,
periodic=[[2,y],[4,y],[1,x],[3,x]]);
Vh uh,vh;
func f=sin(x+pi/4.)*cos(y+pi/4.);
func g=0;

problem laplace(uh,vh) =
int2d(Th)(dx(uh)*dx(vh)+dy(uh)*dy(vh))
+ int2d(Th)(-f*vh);

laplace;
plot(uh, ps="period.eps", value=true);
```

The `type` field distinguishes code files (`.edp`, `.idp`) from documentation (`.rst`, `.md`, `.txt`). The corpus contains both types to expose the model to idiomatic usage patterns and natural language descriptions of FreeFEM concepts.

## A.2 SFT Instruction–Response Format

Each supervised fine-tuning example is a JSON object with an `instruction` field (natural language prompt) and a `response` field (complete FreeFEM script in a fenced code block). The following is a representative training pair:

**Instruction:** *"How do I solve the Laplace equation with a source term of 1, a Dirichlet boundary condition of 0 on all sides of a square mesh, and then plot the solution?"*

**Response:**

```
mesh Th=square(10,10);
fespace Vh(Th,P1);
Vh uh,vh;
func f=1;
func g=0;

problem laplace(uh,vh,solver=GMRES) =
int2d(Th)(dx(uh)*dx(vh)+dy(uh)*dy(vh))
- int2d(Th)(f*vh)
+ on(1,2,3,4,uh=g);

laplace;
plot(uh, value=true);
```

Instructions range from concise specifications ("solve Poisson on the unit square") to detailed requirements specifying element types, boundary labels, and solver parameters. The 739 training pairs span all seven task categories described in Section 3.1.

# B Ablation Study Details

Tables 9–12 report individual-seed results and per-category breakdowns for the data scaling and LoRA rank ablation studies described in Section 5. All runs use $N$=64 samples per prompt at temperature 0.6.

## B.1 Data Scaling: Per-Seed Results in Table 9

Table 9: Data scaling ablation: pass@$k$ for each seed (%). $n$ denotes the number of training examples.

| $n$ | Seed | pass@1 | pass@5 | pass@10 |
|---|---|---|---|---|
| 74 | 42 | 3.6 | 15.3 | 25.9 |
| | 123 | 13.6 | 44.3 | 62.7 |
| | 456 | 20.0 | 57.3 | 74.0 |
| 185 | 42 | 24.2 | 62.4 | 79.6 |
| | 123 | 33.7 | 76.6 | 89.1 |
| | 456 | 26.5 | 68.0 | 83.9 |
| 370 | 42 | 57.9 | 90.3 | 96.1 |
| | 123 | 39.0 | 79.3 | 90.1 |
| | 456 | 51.9 | 88.3 | 94.0 |
| 554 | 42 | 63.0 | 93.6 | 97.3 |
| | 123 | 68.9 | 93.0 | 95.8 |
| | 456 | 63.8 | 93.5 | 96.9 |
| 739 | 42 | 77.2 | 97.3 | 98.5 |
| | 123 | 75.9 | 95.9 | 97.8 |
| | 456 | 72.9 | 96.3 | 98.2 |

## B.2 LoRA Rank: Per-Seed Results in Table 10

## B.3 Per-Category Breakdowns in Tables 11 and 12

# C Failure Taxonomy

Manual inspection of the 20 failed examples reveals distinct error patterns. Undefined variable references account for 23% of failures, typically from incorrect scoping or forward usage. Plugin

Table 10: LoRA rank ablation: pass@$k$ for each seed (%). $\alpha$=2$r$ with rsLoRA enabled.

| $r$ | Seed | pass@1 | pass@5 | pass@10 |
|---|---|---|---|---|
| 8 | 42 | 6.0 | 24.8 | 40.6 |
| | 123 | 5.4 | 23.2 | 39.3 |
| | 456 | 5.4 | 22.8 | 38.1 |
| 16 | 42 | 7.5 | 30.3 | 48.3 |
| | 123 | 6.2 | 25.5 | 41.8 |
| | 456 | 6.1 | 24.9 | 40.7 |
| 32 | 42 | 24.2 | 63.4 | 77.6 |
| | 123 | 31.7 | 72.9 | 86.9 |
| | 456 | 27.3 | 68.2 | 83.6 |
| 64 | 42 | 74.2 | 96.6 | 98.4 |
| | 123 | 74.0 | 94.8 | 96.7 |
| | 456 | 74.6 | 96.0 | 97.2 |

loading failures (18%) occur when models hallucinate unavailable FreeFEM extensions. Type mismatches (15%) arise from incompatible finite element spaces in arithmetic operations. Boundary condition errors (12%) involve undefined labels or malformed condition syntax. Mesh-solver mismatches (10%) attempt operations unsupported by the mesh type. Timeouts (8%) result from non-terminating solver iterations or unbounded refinement loops. Remaining failures (14%) span miscellaneous compilation and runtime issues.

Table 11: Data scaling ablation: mean pass@1 (%) by category across three seeds. Categories ordered by test set size.

| $n$ | PDE Basic | Mesh | Boundary | Time-Dep. | Systems | Visual. | Solver |
|---|---|---|---|---|---|---|---|
| 74 | 15.9 | 7.3 | 17.6 | 7.0 | 11.9 | 16.1 | 7.3 |
| 185 | 31.8 | 25.5 | 33.1 | 22.2 | 18.0 | 31.6 | 28.1 |
| 370 | 55.6 | 44.0 | 58.4 | 49.0 | 32.9 | 48.1 | 47.1 |
| 554 | 68.4 | 62.0 | 72.3 | 70.2 | 46.5 | 69.3 | 56.5 |
| 739 | 79.6 | 71.7 | 81.1 | 80.2 | 56.7 | 79.9 | 60.4 |

Table 12: LoRA rank ablation: mean pass@1 (%) by category across three seeds.

| $r$ | PDE Basic | Mesh | Boundary | Time-Dep. | Systems | Visual. | Solver |
|---|---|---|---|---|---|---|---|
| 8 | 7.5 | 3.6 | 5.7 | 3.4 | 6.8 | 6.0 | 2.6 |
| 16 | 9.2 | 3.7 | 6.6 | 4.9 | 7.7 | 7.4 | 3.1 |
| 32 | 31.1 | 18.5 | 33.1 | 23.4 | 36.5 | 35.9 | 22.1 |
| 64 | 80.6 | 71.0 | 76.4 | 75.2 | 53.0 | 80.6 | 59.1 |