# Fine-Tuning Large Language Models: Enhancing Domain-Specific Performance for Automated Script Generation

## Abstract

Large Language Models (LLMs) have revolutionized natural language processing and code generation tasks, but their out-of-the-box performance often falls short for specialized applications. This white paper explores the process of fine-tuning LLMs, emphasizing its superiority over alternatives like prompting or Retrieval-Augmented Generation (RAG) in certain scenarios. We detail the advantages, step-by-step methodologies, recommended models, and data preparation best practices. A focused use case demonstrates fine-tuning for generating automated scripts from process mining videos, enabling LLMs to autonomously produce executable code from workflow descriptions. Drawing on recent advancements as of 2025, this guide provides actionable insights for researchers, engineers, and enterprises seeking to optimize LLMs for high-precision tasks.

## Introduction

The advent of LLMs such as those from the Llama and Mistral families has democratized AI capabilities, enabling tasks from text generation to code synthesis. However, generic pre-trained models may not align perfectly with niche domains like process mining, where workflows captured in videos need translation into automated scripts (e.g., Selenium or Playwright code for browser automation). Fine-tuning bridges this gap by adapting models to specific datasets, improving accuracy, efficiency, and domain expertise.

<argument name="citation\_id">70</argument>

<argument name="citation\_id">72</argument>

This paper outlines why and how to fine-tune, with a practical use case in process mining.

## Why Fine-Tune Over Other Options

Fine-tuning an LLM involves further training a pre-trained model on a task-specific dataset to adjust its parameters, making it intimately familiar with domain nuances that generic prompting or Retrieval-Augmented Generation (RAG) cannot fully replicate. While prompting relies on carefully crafted inputs to guide the model's output without altering its weights—suitable for quick, low-cost iterations—it often leads to inconsistent results, hallucinations, or verbose responses when dealing with complex, specialized tasks like generating precise automation scripts from process descriptions, as prompts can become overly long and brittle over time.

<argument name="citation\_id">20</argument>

<argument name="citation\_id">22</argument>

<argument name="citation\_id">28</argument>

RAG, on the other hand, augments the model with external knowledge retrieval during inference, excelling in dynamic, fact-based scenarios where real-time updates are needed, but it introduces latency, dependency on retrieval quality, and potential information overload, making it less efficient for static, high-precision domains where the model must internalize patterns rather than query them externally.

<argument name="citation\_id">16</argument>

<argument name="citation\_id">17</argument>

<argument name="citation\_id">18</argument>

<argument name="citation\_id">21</argument>

<argument name="citation\_id">25</argument>

In contrast, fine-tuning embeds domain knowledge directly into the model, yielding faster inference, reduced token costs, and superior consistency for repetitive tasks, though it demands computational resources and data preparation upfront—making it ideal when proprietary datasets (e.g., process mining videos) enable long-term performance gains over iterative prompting or RAG's ongoing retrieval overhead.

<argument name="citation\_id">15</argument>

<argument name="citation\_id">23</argument>

<argument name="citation\_id">24</argument>

<argument name="citation\_id">29</argument>

## Advantages of Fine-Tuning LLMs

Fine-tuning offers several compelling benefits, particularly for code generation and domain adaptation:

- \*\*Domain Fluency and Precision\*\*: By training on specialized data, fine-tuned models gain deep understanding of terminology and workflows, producing more accurate outputs than base models. For instance, in code generation, they can generate syntactically correct and context-aware scripts with fewer errors.

<argument name="citation\_id">24</argument>

<argument name="citation\_id">25</argument>

- \*\*Efficiency and Cost Savings\*\*: Post-fine-tuning, inference is faster and uses fewer tokens, reducing API costs and latency compared to verbose prompting or RAG's retrieval steps.

<argument name="citation\_id">22</argument>

<argument name="citation\_id">23</argument>

- \*\*Consistency and Customization\*\*: Outputs align closely with desired styles, formats, or behaviors, minimizing hallucinations and ensuring repeatable results for production environments.

<argument name="citation\_id">20</argument>

<argument name="citation\_id">18</argument>

- \*\*Handling Proprietary Data\*\*: It allows integration of sensitive or unique datasets (e.g., internal process videos) without exposing them during inference, enhancing privacy and performance on bespoke tasks.

<argument name="citation\_id">15</argument>

<argument name="citation\_id">19</argument>

- \*\*Scalability for Complex Tasks\*\*: For use cases like script generation from videos, fine-tuning enables the model to learn intricate patterns, outperforming alternatives in high-stakes applications.

<argument name="citation\_id">16</argument>

<argument name="citation\_id">29</argument>

However, it requires upfront investment in data and compute, making it best for scenarios with sufficient resources.

## How to Fine-Tune an LLM

Fine-tuning involves adapting a pre-trained LLM using techniques like Supervised Fine-Tuning (SFT) or Parameter-Efficient Fine-Tuning (PEFT). Below is a step-by-step guide using Hugging Face's ecosystem, which supports efficient methods like LoRA and QLoRA for 2025 workflows.

<argument name="citation\_id">60</argument>

<argument name="citation\_id">61</argument>

<argument name="citation\_id">62</argument>

<argument name="citation\_id">63</argument>

<argument name="citation\_id">67</argument>

<argument name="citation\_id">69</argument>

<argument name="citation\_id">71</argument>

<argument name="citation\_id">73</argument>

<argument name="citation\_id">74</argument>

1. \*\*Set Up Environment\*\*: Install libraries like `transformers`, `datasets`, `peft`, `trl`, and `accelerate`. Use PyTorch for backend. For efficiency, enable mixed precision (FP16) and GPU support.

<argument name="citation\_id">60</argument>

<argument name="citation\_id">64</argument>

2. \*\*Select and Load Base Model\*\*: Choose an open-source LLM (see next section). Load via `AutoModelForCausalLM.from\_pretrained(model\_name)` and tokenizer.

<argument name="citation\_id">61</argument>

<argument name="citation\_id">65</argument>

3. \*\*Prepare Dataset\*\*: Load and format data as instruction-completion pairs (details in next section). Use `datasets` library for splitting into train/validation.

<argument name="citation\_id">66</argument>

<argument name="citation\_id">67</argument>

4. \*\*Configure PEFT (e.g., LoRA)\*\*: Apply adapters with `LoraConfig` (e.g., r=16, target\_modules=["q\_proj", "v\_proj"]) to fine-tune efficiently without full parameter updates.

<argument name="citation\_id">61</argument>

<argument name="citation\_id">69</argument>

<argument name="citation\_id">71</argument>

5. \*\*Set Training Arguments\*\*: Use `TrainingArguments` for hyperparameters: epochs (3-5), batch size (2-8), learning rate (2e-4), evaluation strategy ("epoch"). Integrate monitoring with Weights & Biases.

<argument name="citation\_id">60</argument>

<argument name="citation\_id">65</argument>

6. \*\*Initialize Trainer\*\*: Create `SFTTrainer` or `Trainer` with model, args, datasets. For advanced, use QLoRA with quantization.

<argument name="citation\_id">60</argument>

<argument name="citation\_id">67</argument>

7. \*\*Train and Evaluate\*\*: Run `trainer.train()`. Monitor metrics like loss, BLEU for code similarity. Use early stopping to prevent overfitting.

<argument name="citation\_id">61</argument>

<argument name="citation\_id">73</argument>

8. \*\*Save and Deploy\*\*: Save model with `trainer.save\_model()`. Merge adapters if using PEFT, then deploy via Hugging Face Inference or local setup.

<argument name="citation\_id">60</argument>

<argument name="citation\_id">67</argument>

Tools like Unsloth can accelerate training by 2x on consumer hardware.

<argument name="citation\_id">73</argument>

## Recommended LLMs for Fine-Tuning

For code generation tasks in 2025, prioritize open-source models with strong coding benchmarks (e.g., HumanEval scores) and fine-tuning efficiency. Top recommendations include:

- \*\*Llama 3.1 (Meta)\*\*: Variants (8B, 70B, 405B) excel in versatility and fine-tuning; ideal for general code tasks with high parameter efficiency.

<argument name="citation\_id">1</argument>

<argument name="citation\_id">4</argument>

<argument name="citation\_id">7</argument>

<argument name="citation\_id">8</argument>

<argument name="citation\_id">11</argument>

<argument name="citation\_id">13</argument>

- \*\*Qwen 3.5 Coder (Alibaba)\*\*: Specialized for coding; 7B/32B variants top benchmarks for script generation, with excellent fine-tuning on code datasets.

<argument name="citation\_id">1</argument>

<argument name="citation\_id">2</argument>

<argument name="citation\_id">6</argument>

<argument name="citation\_id">8</argument>

<argument name="citation\_id">10</argument>

<argument name="citation\_id">12</argument>

- \*\*DeepSeek-V3 (DeepSeek AI)\*\*: 7B/32B models shine in coding and reasoning; cost-effective for fine-tuning on process-specific data.

<argument name="citation\_id">6</argument>

<argument name="citation\_id">8</argument>

- \*\*Mistral-8x22B (Mistral AI)\*\*: Balanced for multilingual code; efficient with PEFT for domain adaptation.

<argument name="citation\_id">1</argument>

<argument name="citation\_id">6</argument>

<argument name="citation\_id">7</argument>

- \*\*Gemma 2 (Google)\*\*: 9B/27B variants are lightweight yet powerful for code, suitable for resource-constrained fine-tuning.

<argument name="citation\_id">5</argument>

<argument name="citation\_id">8</argument>

<argument name="citation\_id">11</argument>

Start with smaller variants (e.g., 7B-9B) for experimentation, scaling to larger for production.

## How to Prepare Training Data

High-quality data is pivotal for effective fine-tuning, especially for code generation from process descriptions. Focus on structured, diverse pairs to teach the LLM patterns.

<argument name="citation\_id">45</argument>

<argument name="citation\_id">46</argument>

<argument name="citation\_id">47</argument>

<argument name="citation\_id">48</argument>

<argument name="citation\_id">49</argument>

<argument name="citation\_id">52</argument>

<argument name="citation\_id">53</argument>

<argument name="citation\_id">55</argument>

<argument name="citation\_id">56</argument>

<argument name="citation\_id">58</argument>

<argument name="citation\_id">59</argument>

1. \*\*Gather Raw Data\*\*: Collect process descriptions (e.g., transcribed from videos using Whisper) and corresponding scripts. Aim for 1,000+ pairs for robust results.

<argument name="citation\_id">45</argument>

<argument name="citation\_id">49</argument>

<argument name="citation\_id">58</argument>

2. \*\*Format as Instruction Pairs\*\*: Use JSONL with "prompt" (description) and "completion" (script). Ensure prompts are step-by-step and completions are executable.

<argument name="citation\_id">48</argument>

<argument name="citation\_id">49</argument>

<argument name="citation\_id">52</argument>

<argument name="citation\_id">53</argument>

<argument name="citation\_id">56</argument>

3. \*\*Clean and Augment\*\*: Remove noise, duplicates; augment with paraphrases or synthetic variations using another LLM. Balance diversity (simple/complex processes).

<argument name="citation\_id">45</argument>

<argument name="citation\_id">47</argument>

<argument name="citation\_id">48</argument>

<argument name="citation\_id">55</argument>

<argument name="citation\_id">59</argument>

4. \*\*Validate Quality\*\*: Test scripts for executability; ensure descriptions are concise (<512 tokens). Split into train/val/test.

<argument name="citation\_id">46</argument>

<argument name="citation\_id">53</argument>

<argument name="citation\_id">55</argument>

5. \*\*Tokenize and Prune\*\*: Apply model's tokenizer; prune outliers for efficiency.

<argument name="citation\_id">49</argument>

<argument name="citation\_id">53</argument>

For code gen, incorporate metrics like execution success during validation.

## Use Case: Fine-Tuning for Automated Script Generation from Process Mining Videos

Process mining involves analyzing workflow videos (e.g., screen recordings of business processes like invoice approval) to extract insights and automate tasks via scripts. With loads of such videos, fine-tuning enables LLMs to generate scripts autonomously, reducing manual effort.

<argument name="citation\_id">32</argument>

<argument name="citation\_id">35</argument>

<argument name="citation\_id">36</argument>

<argument name="citation\_id">37</argument>

<argument name="citation\_id">41</argument>

<argument name="citation\_id">42</argument>

<argument name="citation\_id">44</argument>

\*\*Workflow\*\*:

- \*\*Data Prep\*\*: Transcribe videos to step descriptions (e.g., "User navigates to ERP dashboard, enters data"). Pair with ground-truth scripts.

<argument name="citation\_id">35</argument>

<argument name="citation\_id">42</argument>

- \*\*Fine-Tune\*\*: Use Qwen 3.5 Coder with PEFT on pairs; train to output scripts from descriptions.

<argument name="citation\_id">34</argument>

<argument name="citation\_id">39</argument>

<argument name="citation\_id">41</argument>

- \*\*Outcome\*\*: The model generates executable scripts directly from new videos, improving automation in enterprises.

<argument name="citation\_id">32</argument>

<argument name="citation\_id">38</argument>

<argument name="citation\_id">44</argument>

This approach scales to thousands of videos, enabling self-sufficient LLM-driven automation.

## Conclusion

Fine-tuning empowers LLMs for specialized tasks like script generation from process mining, offering unmatched precision over alternatives. By following the outlined steps, selecting appropriate models, and prioritizing data quality, organizations can unlock significant efficiencies. As LLM technology evolves, fine-tuning remains a cornerstone for custom AI solutions.