SFT Fine-Tuning Workflow

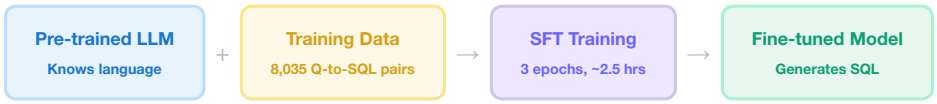
Complete Architecture and Data Flow for the Text-to-SQL Pipeline

TinyLlama 1.1B · LoRA · Apple MPS · Spider Dataset

MODEL	METHOD	DATASET	FINAL LOSS	TOKEN ACCURACY
TinyLlama-1.1B	**SFT + LoRA**	**8,035 examples**	**0.4315**	**87.5%**

1 What is SFT (Supervised Fine-Tuning)?

SFT teaches a pre-trained language model to perform a **specific task** by showing it thousands of **(input, output) examples**. The model already understands language. SFT teaches it to convert natural language questions into SQL queries.



METHOD	WHAT IT NEEDS	USED FOR	OUR CHOICE?
SFT (Supervised Fine-Tuning)	Clean (input, output) pairs	Task-specific training	YES - we used this
RLHF (Reinforcement Learning from Human Feedback)	Preference data (good vs bad answers)	Alignment, safety tuning	NO - no preference data available
DPO (Direct Preference Optimization)	Paired preferences	Simpler RLHF alternative	NO - not needed for this task

Why SFT? Our dataset has clean (question, SQL) pairs -- ideal for supervised learning. RLHF and DPO require preference data (which answer is better/worse), which we do not have.

2 Complete Architecture -- Step by Step

Step 1: Dataset Loading

Read 8,035 rows from the Spider CSV file. Each row contains a natural language question and its correct SQL answer. The data is split into **7,230 train** / **402 validation** / **402 test** examples.

`data/spider_text_sql.csv -- loaded by SpiderCSVPipeline in data/pipeline.py`

Step 2: Prompt Formatting

Each example is converted into an instruction format that the model learns to follow:

```
[INST] Generate SQL for the following question. Question: How many employees are in each department? [/INST]
SELECT department, COUNT(*) FROM employees GROUP BY department;
```

The model learns: everything before [/INST] is the question; everything after is the SQL to generate.

Step 3: Tokenization

Text is converted to numbers (tokens) that the model processes. Each sequence is truncated or padded to a maximum of **256 tokens** to save GPU memory.

```
"SELECT COUNT(*)" → [14021, 18678, 29898, 334]
```

Step 4: Load Base Model (TinyLlama 1.1B)

The pre-trained model is downloaded from HuggingFace and cached locally (~2.2 GB). It already understands language but does not know how to generate SQL from questions.

```
~/.cache/huggingface/hub/models--TinyLlama--TinyLlama-1.1B-Chat-v1.0/
```

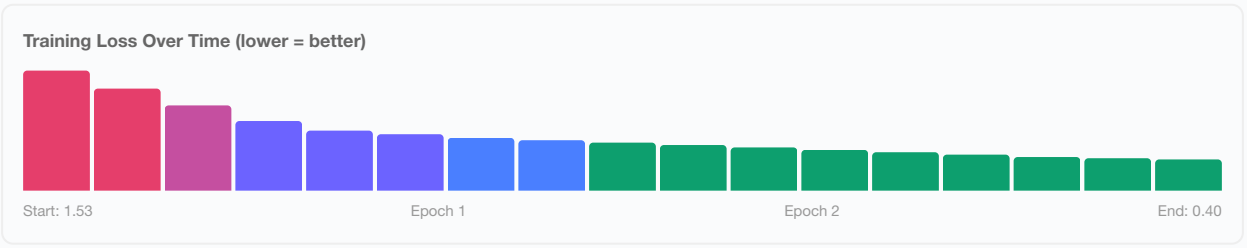
2 Complete Architecture -- Step by Step (continued)

Step 5: LoRA Adapter Injection

Instead of training all 1.1 billion parameters (very expensive), we attach small trainable "adapter" matrices to just two attention layers: `q_proj` and `v_proj`. Only **1.1 million parameters (0.10%)** are actually trained.

Step 6: Training Loop (SFT)

For each batch of examples: the model predicts SQL, the prediction is compared to the correct answer (loss), and the optimizer adjusts the LoRA weights to reduce the loss. This is repeated **1,356 times** across 3 full passes over the data.



Step 7: Save Fine-tuned Model

Only the LoRA adapter weights are saved -- just **4.5 MB** instead of the full 2.2 GB model. To use the model later, we load the base TinyLlama plus this small adapter file.

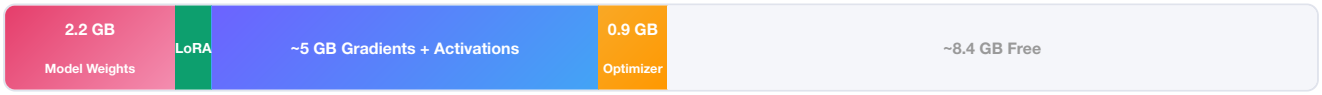
`outputs/spider/final_model/adapter_model.safetensors (4.5 MB)`

3 Where is Everything Stored?

COMPONENT	FILE PATH	SIZE	DESCRIPTION
Training Dataset	`data/spider_text_sql.csv`	**1.4 MB**	8,035 question-to-SQL training examples from Spider benchmark
Base Model (cached)	`~/.cache/huggingface/hub/models--TinyLlama/`	**~2.2 GB**	Pre-trained TinyLlama-1.1B, downloaded once from HuggingFace
Trained LoRA Adapter	`outputs/spider/final_model/adapter_model.safetensors`	**4.5 MB**	Your fine-tuned model weights -- this is the training output
Tokenizer	`outputs/spider/final_model/tokenizer.json`	**3.6 MB**	Vocabulary mapping (text to token IDs)
Training Checkpoints	`outputs/spider/checkpoints/checkpoint-600/`	**~10 MB each**	Intermediate saves for resuming interrupted training
Training Config	`configs/config_spider.yaml`	**2 KB**	All hyperparameters (learning rate, batch size, epochs, etc.)
Training Script	`scripts/train_spider.py`	**6 KB**	Main entry point -- run this to start training
API Server	`api/serve.py`	**6 KB**	FastAPI server connecting trained model to the frontend

4 GPU Memory Usage During Training

Your Mac has **16 GB unified memory** shared between CPU and GPU. Here is the breakdown during training:



BLOCK	SIZE	WHAT IT IS	STORED WHERE?
Model Weights	2.2 GB	TinyLlama frozen parameters (not trained)	GPU memory
LoRA Adapters	4 MB	Small trainable matrices attached to attention layers	GPU memory
Gradients + Activations	~5 GB	Temporary calculations for learning (deleted each step)	GPU memory
Optimizer States	~0.9 GB	AdamW momentum and variance for weight updates	GPU memory
Training Data	~50 MB	Dataset loaded into regular RAM, NOT on GPU	System RAM

Total GPU used: ~7.6 GB / 16 GB. No personal data from your Mac is used. Only the Spider dataset CSV.

5 Training Configuration

PARAMETER	VALUE	REASON
Base Model	TinyLlama-1.1B-Chat-v1.0	Small enough for 16GB Mac, capable enough for SQL generation
Fine-tuning Method	SFT + LoRA	Dataset has clean input/output pairs; LoRA works on MPS
LoRA Rank (r)	8	Lower rank = less memory, still effective for this task
Target Modules	q_proj, v_proj	The most impactful attention layers for fine-tuning
Trainable Parameters	1.1M / 1,101M (0.10%)	Only the LoRA adapter matrices are updated
Number of Epochs	3	Standard for a dataset of ~8,000 examples
Batch Size	1 x 16 accumulation = 16 effective	Batch of 1 for memory; accumulate 16 steps for stable gradients
Learning Rate	2e-4 with cosine decay	Standard learning rate for LoRA fine-tuning
Max Sequence Length	256 tokens	Halves memory compared to 512; most SQL fits within 256
Gradient Checkpointing	Enabled	Reduces activation memory by approximately 40%
Precision	float16	Half the memory compared to float32
Device	Apple MPS (Metal GPU)	Auto-detected; also supports CUDA and CPU

6 Training Results

TRAINING LOSS	EVAL LOSS	TOKEN ACCURACY	TRAINING TIME
0.40	0.43	87.5%	~2.5h
Down from 1.53 (74% reduction)	Close to train = no overfitting	Correct next-token predictions	On Apple MPS GPU

Sample Model Outputs:

INPUT QUESTION	GENERATED SQL
What is the average salary of all employees?	`SELECT avg(salary) FROM employees;`
Find the product with the highest price	`SELECT * FROM products ORDER BY price DESC LIMIT 1;`
How many employees are in each department?	`SELECT count(*), dept_code FROM employees GROUP BY dept_code;`
Show all orders with their product names	`SELECT * FROM orders WHERE product_id IN (SELECT product_id FROM products);`

7 How Inference Works (After Training)

STEP	ACTION	DETAILS
1	API receives request	Frontend sends POST /generate_sql with question and schema to FastAPI
2	Build prompt	Wraps question in [INST]...[/INST] format, same as during training
3	Tokenize	Convert text to token IDs and move tensor to GPU (MPS)
4	Generate	Model generates tokens one by one until SQL is complete (~0.5-3 seconds)
5	Extract and return	Parse generated text, extract SQL after [/INST], ensure semicolon, return as JSON

8 Project File Map

DATA		
`data/spider_text_sql.csv`	8,035 question-to-SQL training examples from the Spider benchmark	TRAINING DATA
`data/pipeline.py`	SpiderCSVPipeline class -- reads CSV, formats prompts, splits train/val/test	DATA LOADER
CONFIGURATION		
`configs/config_spider.yaml`	All training settings: model name, LoRA rank, batch size, learning rate, epochs	CONFIG
TRAINING		
`scripts/train_spider.py`	Main training script -- loads data, model, runs SFT loop, saves adapter	ENTRY POINT
`models/loader.py`	Model loading utilities for both training and inference	MODEL UTILS
TRAINED MODEL OUTPUT		
`outputs/spider/final_model/adapter_model.safetensors`	The trained LoRA adapter weights (4.5 MB) -- THIS IS YOUR MODEL	TRAINED WEIGHTS
`outputs/spider/final_model/adapter_config.json`	LoRA configuration (rank, alpha, target modules)	ADAPTER CONFIG
`outputs/spider/final_model/tokenizer.json`	Vocabulary mapping file for converting text to tokens	TOKENIZER
`outputs/spider/checkpoints/checkpoint-600/`	Intermediate checkpoint for resuming interrupted training	CHECKPOINT
INFERENCE AND API		
`inference/engine.py`	Inference engine -- builds prompt, runs model.generate(), extracts SQL	INFERENCE
`api/serve.py`	FastAPI server -- loads model, exposes /generate_sql endpoint for frontend	API SERVER
FRONTEND		
`frontend/`	Next.js web application -- user interface for entering questions and schemas	WEB UI

EXTERNAL (NOT IN PROJECT)		
`~/.cache/huggingface/hub/models--TinyLlama/`	Base TinyLlama-1.1B model (~2.2 GB), downloaded once and cached globally	BASE MODEL CACHE

Summary: The trained model is a 4.5 MB LoRA adapter file. Combined with the base TinyLlama (2.2 GB from the shared HuggingFace cache), it generates SQL from natural language questions. No personal data from your Mac was accessed or used during training -- only the public Spider benchmark dataset.