LLM Research - Lightweight Model and Fine Tuning

Lightweight Model:

Mistral-Small-24B-Instruct-2501-3bit

3 Billion parameter model quantized from 24 billion parameter model
Runs extremely fast on Mac, decent results.
May only perform well for Mac silicon users/users with a decent setup
~15 GB size

---

https://huggingface.co/mlx-community/Mistral-Small-24B-Instruct-2501-3bit
pip install mlx-lm

In Bash: python scripts/convert.py --hf-path mlx-community/Mistral-Small-24B-Instruct-2501-3bit -q

```
from mlx_lm import load, generate

model, tokenizer = load("mlx-community/Mistral-Small-24B-Instruct-2501-3bit")

prompt = "hello"

if tokenizer.chat_template is not None:
    messages = [{"role": "user", "content": prompt}]
    prompt = tokenizer.apply_chat_template(
        messages, add_generation_prompt=True
    )

response = generate(model, tokenizer, prompt=prompt, verbose=True)
```

Model Tuning Options:

Option A – Single-stage SFT (simplest)

Train Mistral Small 24B (or a smaller model) via **supervised fine-tuning** where it learns to:

- Read question + snippets + templates
- Output TEMPLATE: X + the final answer

Pros:

- Easiest conceptually (one model, one dataset)
- Will have the strongest tuning results

Cons:

- Requires huge amounts of examples
- 24B SFT is very very heavy computationally

Option B – LoRA/PEFT tuning

Train Mistral Small 24B (or a smaller model) via **LoRA/PEFT** where it learns to:

- Read question + snippets + templates
- Output TEMPLATE: X + the final answer

Pros:

- Easiest computationally

Cons:

- Performance may be worse due than prior setup
- Will require huge amounts of examples

Option C – Two-stage: classifier + generator

Sometimes nicer for a project:

1. **Small classifier head** (could even be a light model, or logistic regression on embeddings) that predicts a **template label** {A, B, C, …} from:
    - question
    - maybe snippet summaries
2. **LLM (Mistral)** fine-tuned (or just prompted) to **fill in the template** once you tell it which template to use.

Pros:

- Much cheaper to train the classifier (non-24B)
- Easier evaluation: template accuracy, confusion matrix
- You can swap in a different LLM for generation if needed

Cons:

- Slightly more plumbing in the system
- Need labeled template IDs separately

What will tuning look like?

	Comment	Response
1	tention mask for decoders on slide 10 is reversed	s-data-science/how-to-build-an-llm-from-scratch-8c477768f1f9?sk=18c351c5cae9ac89df682dd14736a9f3
2	and answered many questions I had. Thank you.	Great to hear, glad it was helpful :)
3	meframe which I thought was almost impossible.	My pleasure, glad it was informative yet concise :)
4	. It's so rare to find good communicators like you!	Thanks, glad it was clear
5	clearly, wish you led the courses at my university.	Thanks for the kind words! Maybe one day
6	tput of your PCA video and ran ICA on top of that	CA: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.FastICA.html Hope that helps!
7	use a seed or similar to control this randomness?	e link below for you reference. https://www.mathworks.com/help/stats/rica.html#bvnjjp8-5 Hope that helps!
8	se, no random personal opinions or bias. Thanks!	Thanks, glad u liked it :)
9	er so I can create one for the assets and images.	stion! When you hit “Add a File”, you can prepend the filename with the path e.g. assets/img/headshot.png
10	othing usefull until this video. thank you so much	I was in a similar situation. That's a big reason for this video. Glad it was helpful!
11	displaying the practical application in a fun way :)	Glad it was helpful!
12	you explained totally wrong, and waist my time.	I'm sorry, which parts are wrong?

```
training_args = TrainingArguments(
    output_dir="mistral24b-full-sft",
    per_device_train_batch_size=1,      # will be small for 24B
    gradient_accumulation_steps=8,      # effective batch size = 8
    learning_rate=1e-5,
    num_train_epochs=1,                 # or steps if you prefer
    logging_steps=10,
    save_steps=200,
    eval_strategy="steps",
    eval_steps=200,
    bf16=True,                          # bfloat16 on TPU
    tpu_num_cores=8,                    # adjust to your TPU
    max_grad_norm=1.0,
)

trainer = SFTTrainer(
    model=model,
    tokenizer=tokenizer,
    train_dataset=train_ds,
    eval_dataset=val_ds,
    formatting_func=formatting_func,
    max_seq_length=4096,   # be conservative with 24B
    args=training_args,
)

trainer.train()

# Save the fully fine-tuned model
trainer.model.save_pretrained("mistral24b-full-sft-final")
tokenizer.save_pretrained("mistral24b-full-sft-final")
```

How will using an adapted model look like?

Run inference with fine-tuned model

```
adapter_path = "adapters.npz" # same as default
max_tokens_str = str(max_tokens)
```

```
# define command
command = ['python', 'scripts/lora.py', '--model', model_path, '--adapter-file', adapter_path, '--max-tokens', ma

# run command and print results continuously
run_command_with_live_output(command)
```

```
# Configure LoRA
peft_config = LoraConfig(
    r=64,                       # rank (tuneable)
    lora_alpha=16,
    lora_dropout=0.05,
    bias="none",
    task_type="CAUSAL_LM",
    target_modules=[            # typical for Mistral-style models
        "q_proj", "k_proj", "v_proj", "o_proj",
        "gate_proj", "up_proj", "down_proj",
    ],
)

model = get_peft_model(model, peft_config)

training_args = TrainingArguments(
    output_dir="mistral24b-lora-sft",
    per_device_train_batch_size=2,      # often can be larger than full SFT
    gradient_accumulation_steps=8,
    learning_rate=2e-4,                 # LoRA can use higher LR
    num_train_epochs=1,
    logging_steps=10,
    save_steps=200,
    eval_strategy="steps",
    eval_steps=200,
    bf16=True,
    tpu_num_cores=8,
    max_grad_norm=1.0,
)
```

What should a single training example look like?

Given your pipeline, each example should basically encode:

- **User question**
- **Retrieved snippets** (from RAG)
- **The menu of answer templates**
- **The target behavior**: which template to “choose” and what final answer to produce

A nice, LLM-friendly format is something like:

Input:

You are a financial analysis assistant. Use ONLY the snippets and be explicit about uncertainty.

USER QUESTION:

{user_question}

RETRIEVED SNIPPETS:

[1] {snippet_1}

[2] {snippet_2}

[3] {snippet_3}

AVAILABLE ANSWER TEMPLATES:

[A] {template_A}

[B] {template_B}

[C] {template_C}

For this question, choose ONE template (A, B, C, …) that best fits the situation, then fill it in with information from the snippets. Always provide 2–3 sentences, and cite snippet IDs like [1], [2].

Output:

TEMPLATE: B

ANSWER:

Company XYZ has a volatile but overall positive performance over the last year, with revenue growing 12% year-over-year while margins have compressed slightly due to higher input costs [1], [2]. Despite short-term price swings, analyst commentary and earnings guidance point to continued growth in core segments [3].

How many examples will we need?

Rough guidelines for your setting (financial Q&A, templates, RAG):

- **Bare-minimum to see some benefit**:

  ~100–200 *really* good, diverse examples.

- **Comfortable for style + template selection**:

  ~500–1,000 examples.

- **Stronger robustness (different companies, macro situations, edge cases)**:

  ~2,000–5,000 examples if you can manage it.

Within that:

- Try to have **dozens of examples per template** at least.
- Make sure you cover:
    - bull / bear / sideways market cases
    - missing or conflicting data in snippets
    - trickier questions (risk factors, uncertainty, “we don’t know yet” cases)

Given this is for a course project, I’d aim for:

> **500–1,000 curated examples** with a very clear format, plus a held-out ~100-example test set that you manually evaluate carefully.

We can use a stronger LLM to create curated examples

Deliverables:

example-code.ipynb

lora-example-code.ipynb

Template List

Example Prompt