The Goal

Goal is to fine tune an LLM such that it can generate a domain suggestion for a website, based on the prompt.

The setup

LLM

As base opensource LLM, unsloth/mistral-7b is used. It is a rather small model (which fits perfectly for the highly specific task) + already an instruct model. Although one could agree, that it is even too powerfull because it is multimodal (which is useless now but good to have for the future).

Library

For fine tuning, the opensource framework unsloth is used.

General settings

For the sake of simplicity, synthetic data is genereted with an LLM. It is common (for example in distilation) to use the same LLM that is later fine tuned for generating input for this LLM. We will use Mistral AI here. Mistral AI is opensource, avaiable in small sizes and powerful (LLama/GPT level).

For the usage, Thinking/CoT is enabled. Mainly for the purpose, that the LLM filters byself stupid suggestions out ("we-are-trustworthy-money.com" for example would be valid but stupid).

Generate 300 domain names from different business fields and provide a related prompt for each one. Send me your structure in JSON format, as is common for LLM fine-tuning ("text": "###").

Bear in mind that users may ask in strange ways, but some ask directly and seriously. Vary the prompts in the structure, saying, etc.

```
!pip install unsloth
```

```
Collecting unsloth
  Downloading unsloth-2025.7.11-py3-none-any.whl.metadata (47 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 47.3/47.3 kB 3.4 MB/s eta 0:00:00
```

```
Collecting unsloth_zoo>=2025.7.11 (from unsloth)
  Downloading unsloth_zoo-2025.7.11-py3-none-any.whl.metadata (8.1 kB)
Requirement already satisfied: torch>=2.4.0 in /usr/local/lib/python3.11/dist-pack
Collecting xformers>=0.0.27.post2 (from unsloth)
  Downloading xformers-0.0.31.post1-cp39-abi3-manylinux_2_28_x86_64.whl.metadata (
Collecting bitsandbytes (from unsloth)
  Downloading bitsandbytes-0.46.1-py3-none-manylinux_2_24_x86_64.whl.metadata (10
Requirement already satisfied: triton>=3.0.0 in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-package
Collecting tyro (from unsloth)
  Downloading tyro-0.9.27-py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: transformers!=4.47.0,!=4.52.0,!=4.52.1,!=4.52.2,!=4
Collecting datasets<4.0.0,>=3.4.1 (from unsloth)
  Downloading datasets-3.6.0-py3-none-any.whl.metadata (19 kB)
Requirement already satisfied: sentencepiece>=0.2.0 in /usr/local/lib/python3.11/d
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (fr
Requirement already satisfied: psutil in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: wheel>=0.42.0 in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (f
Requirement already satisfied: accelerate>=0.34.1 in /usr/local/lib/python3.11/dis
Collecting trl!=0.15.0,!=0.19.0,!=0.9.0,!=0.9.1,!=0.9.2,!=0.9.3,>=0.7.9 (from unsl
  Downloading trl-0.20.0-py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: peft!=0.11.0,>=0.7.1 in /usr/local/lib/python3.11/d
Requirement already satisfied: protobuf in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: huggingface_hub>=0.34.0 in /usr/local/lib/python3.1
Requirement already satisfied: hf_transfer in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: diffusers in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: torchvision in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: pyyaml in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.11/dis
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: pyarrow>=15.0.0 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: dill<0.3.9,>=0.3.0 in /usr/local/lib/python3.11/dis
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: requests>=2.32.2 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: xxhash in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: multiprocess<0.70.17 in /usr/local/lib/python3.11/d
Requirement already satisfied: fsspec<=2025.3.0,>=2023.1.0 in /usr/local/lib/pytho
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.11/d
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (
Collecting nvidia-cuda-nvrtc-cu12==12.4.127 (from torch>=2.4.0->unsloth)
  Downloading nvidia_cuda_nvrtc_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.me
Collecting nvidia-cuda-runtime-cu12==12.4.127 (from torch>=2.4.0->unsloth)
  Downloading nvidia_cuda_runtime_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.
Collecting nvidia-cuda-cupti-cu12==12.4.127 (from torch>=2.4.0->unsloth)
  Downloading nvidia_cuda_cupti_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.me
Collecting nvidia-cudnn-cu12==9.1.0.70 (from torch>=2.4.0->unsloth)
  Downloading nvidia_cudnn_cu12-9.1.0.70-py3-none-manylinux2014_x86_64.whl.metadat
Collecting nvidia-cublas-cu12==12.4.5.8 (from torch>=2.4.0->unsloth)
  Downloading nvidia_cublas_cu12-12.4.5.8-py3-none-manylinux2014_x86_64.whl.metada
Collecting nvidia-cufft-cu12==11.2.1.3 (from torch>=2.4.0->unsloth)
  Downloading nvidia_cufft_cu12-11.2.1.3-py3-none-manylinux2014_x86_64.whl.metadat
Collecting nvidia-curand-cu12==10.3.5.147 (from torch>=2.4.0->unsloth)
```

```
from unsloth import FastLanguageModel
import torch
```

```
🦥 Unsloth: Will patch your computer to enable 2x faster free finetuning.
🦥 Unsloth Zoo will now patch everything to make training faster!
```

```
import json
from datasets import Dataset

with open("prompts.json") as f:
    data = json.load(f)

mistral_data = Dataset.from_list([
    {"text": f"<s>[INST] {item['input']} [/INST] {item['output']}</s>"}
    for item in data
])
```

```
from unsloth import FastLanguageModel

model, tokenizer = FastLanguageModel.from_pretrained(
    model_name = "mistralai/Mistral-7B-v0.1",
    max_seq_length = 2048,
    dtype = None,
    load_in_4bit = True,
)

model = FastLanguageModel.get_peft_model(
    model,
    r = 16,  # LoRA-Rang
    target_modules = [
        "q_proj", "k_proj", "v_proj", "o_proj",
        "gate_proj", "up_proj", "down_proj"
    ],
    lora_alpha = 16,
    lora_dropout = 0,
    bias = "none",
    use_gradient_checkpointing = True,
    random_state = 3407,
    use_rslora = False,
    loftq_config = None,
)
```

```
==((====))==  Unsloth 2025.7.11: Fast Mistral patching. Transformers: 4.54.1.
   \\   /|    Tesla T4. Num GPUs = 1. Max memory: 14.741 GB. Platform: Linux.
O^O/ \_/ \    Torch: 2.7.1+cu126. CUDA: 7.5. CUDA Toolkit: 12.6. Triton: 3.3.1
\        /    Bfloat16 = FALSE. FA [Xformers = 0.0.31.post1. FA2 = False]
 "-____-"     Free license: http://github.com/unslothai/unsloth
Unsloth: Fast downloading is enabled - ignore downloading bars which are red color
model.safetensors: 100%                                 4.13G/4.13G [01:06<00:00, 106MB/
s]
generation_config.json: 100%                                155/155 [00:00<00:00, 13.0kB/
s]
```

tokenizer_config.json: 1.02k/? [00:00<00:00, 71.3kB/s]

tokenizer.model: 100% 493k/493k [00:00<00:00, 1.17MB/s]

```
mistral_data = mistral_data.train_test_split(test_size=0.1)
train_data = mistral_data['train']
val_data = mistral_data['test']
```

```
from trl import SFTTrainer
from transformers import TrainingArguments
from unsloth import is_bfloat16_supported

trainer = SFTTrainer(
    model=model,
    tokenizer=tokenizer,
    train_dataset=train_data,
    eval_dataset=val_data,
    dataset_text_field="text",
    max_seq_length=2048,
    dataset_num_proc=2,
    packing=True,
    args=TrainingArguments(
        per_device_train_batch_size=2,
        gradient_accumulation_steps=4,
        warmup_steps=5,
        max_steps=60,
        learning_rate=2e-4,
        fp16=not is_bfloat16_supported(),
        bf16=is_bfloat16_supported(),
        logging_steps=1,
        logging_strategy="steps",
        eval_steps=1,
        save_strategy="no",
        optim="adamw_8bit",
        weight_decay=0.01,
        lr_scheduler_type="linear",
        seed=3407,
        output_dir="outputs",
    ),
)
```

Unsloth: Tokenizing ["text"]: 100% 285/285 [00:00<00:00, 4235.20 examples/s]

Unsloth: Tokenizing ["text"]: 100% 32/32 [00:00<00:00, 970.00 examples/

```
trainer_stats = trainer.train()
```

```
==((====))==  Unsloth - 2x faster free finetuning | Num GPUs used = 1
   \\   /|    Num examples = 285 | Num Epochs = 2 | Total steps = 60
O^O/ \_/ \    Batch size per device = 2 | Gradient accumulation steps = 4
\        /    Data Parallel GPUs = 1 | Total batch size (2 x 4 x 1) = 8
 "-____-"     Trainable parameters = 41,943,040 of 7,283,675,136 (0.58% trained)
```

[60/60 03:01, Epoch 1/2]

Step	Training Loss
1	0.418400
2	0.538500
3	0.404500
4	0.502400
5	0.426900
6	0.417200
7	0.591900
8	0.493200
9	0.568100
10	0.533000
11	0.459200
12	0.471800
13	0.551100
14	0.496600
15	0.503300
16	0.508300
17	0.482200
18	0.537600
19	0.662600
20	0.624000
21	0.535400
22	0.586400
23	0.539900
24	0.512400
25	0.547600
26	0.567100
27	0.453900
28	0.445000

29	0.472400
30	0.408900
31	0.482400
32	0.454000
33	0.653400
34	0.699000
35	0.571100
36	0.473900
37	0.345700
38	0.338000
39	0.321200
40	0.429100
41	0.437400
42	0.324400
43	0.361300
44	0.363800
45	0.394200
46	0.315100
47	0.349900
48	0.362700
49	0.345500
50	0.380200
51	0.337700
52	0.344700
53	0.314800
54	0.327600
55	0.335000
56	0.338800
57	0.316400
58	0.371900
59	0.435100
60	0.309400

```
trainer.save_model("reference");
```

Beginnen Sie mit dem Programmieren oder generieren Sie Code mit KI.

Apparantly, the model is converging (although test data is so far not checked). However, their might be space for improvement because the training appears instable.

```
from unsloth import FastLanguageModel

model, tokenizer = FastLanguageModel.from_pretrained(
    model_name = "mistralai/Mistral-7B-v0.1",
    max_seq_length = 2048,
    dtype = None,
    load_in_4bit = True,
)

model = FastLanguageModel.get_peft_model(
    model,
    r = 16,  # LoRA-Rang
    target_modules = [
        "q_proj", "k_proj", "v_proj", "o_proj",
        "gate_proj", "up_proj", "down_proj"
    ],
    lora_alpha = 16,
    lora_dropout = 0,
    bias = "none",
    use_gradient_checkpointing = True,
    random_state = 3407,
    use_rslora = False,
    loftq_config = None,
)

trainer = SFTTrainer(
    model=model,
    tokenizer=tokenizer,
    train_dataset=train_data,
    eval_dataset=val_data,
    dataset_text_field="text",
    max_seq_length=2048,
    dataset_num_proc=2,
    packing=True,
    args=TrainingArguments(
        per_device_train_batch_size=2,
        gradient_accumulation_steps=4,
        warmup_steps=5,
        max_steps=60,
        learning_rate=1e-4,
        fp16=not is_bfloat16_supported(),
        bf16=is_bfloat16_supported(),
        logging_steps=1,
        logging_strategy="steps",
        eval_steps=1
```

```
        eval_steps=1,
        save_strategy="no",
        optim="adamw_8bit",
        weight_decay=0.01,
        lr_scheduler_type="cosine",
        seed=3407,
        output_dir="outputs",
    ),
)
trainer_stats = trainer.train()
```

```
==((====))==  Unsloth 2025.7.11: Fast Mistral patching. Transformers: 4.54.1.
   \\   /|    Tesla T4. Num GPUs = 1. Max memory: 14.741 GB. Platform: Linux.
O^O/ \_/ \    Torch: 2.7.1+cu126. CUDA: 7.5. CUDA Toolkit: 12.6. Triton: 3.3.1
\        /    Bfloat16 = FALSE. FA [Xformers = 0.0.31.post1. FA2 = False]
 "-____-"     Free license: http://github.com/unslothai/unsloth
Unsloth: Fast downloading is enabled - ignore downloading bars which are red color
```

Unsloth: Tokenizing ["text"]: 100% 285/285 [00:00<00:00, 5925.22 examples/s]

Unsloth: Tokenizing ["text"]: 100% 32/32 [00:00<00:00, 1236.19 examples/s]

```
==((====))==  Unsloth - 2x faster free finetuning | Num GPUs used = 1
   \\   /|    Num examples = 285 | Num Epochs = 2 | Total steps = 60
O^O/ \_/ \    Batch size per device = 2 | Gradient accumulation steps = 4
\        /    Data Parallel GPUs = 1 | Total batch size (2 x 4 x 1) = 8
 "-____-"     Trainable parameters = 41,943,040 of 7,283,675,136 (0.58% trained)
```

[60/60 02:57, Epoch 1/2]

Step	Training Loss
1	4.291000
2	4.706500
3	4.184500
4	3.992200
5	3.687800
6	3.284400
7	2.882700
8	2.433900
9	2.338000
10	2.102300
11	1.875100
12	1.938000
13	1.993900
14	1.900200
15	1.910100

16	1.826900
17	1.760600
18	1.833600
19	2.020300
20	1.917500
21	1.788600
22	1.597400
23	1.734500
24	1.690500
25	1.511400
26	1.627700
27	1.452700
28	1.409000
29	1.400600
30	1.314200
31	1.301800
32	1.221900
33	1.434700
34	1.399800
35	1.383900
36	1.201600
37	1.125400
38	1.177700
39	1.064900
40	1.185600
41	1.132200
42	0.957600
43	1.004700
44	0.962500
45	1.111400
46	0.863400
47	0.967000

48	0.977400
49	0.850500
50	0.902500
51	0.902600
52	0.866400
53	0.772500
54	0.723900
55	0.793700
56	0.861000
57	0.800400
58	0.888700
59	1.008200

Apparantly, this tuning did not work. The concrete hyperparamters would requiere a hyparamter tuning with for example the ray framework (see below).

```
!pip install ray[tune]
```

```
from unsloth import FastLanguageModel
from transformers import TrainingArguments
from trl import SFTTrainer
import ray
from ray import tune
from ray.tune.schedulers import PopulationBasedTraining
from ray.tune import CLIReporter
import os

# Initialize Ray
ray.init()

# Load base model and tokenizer
model, tokenizer = FastLanguageModel.from_pretrained(
    model_name="mistralai/Mistral-7B-v0.1",
    max_seq_length=2048,
    dtype=None,
    load_in_4bit=True,
)

# Define train function for Ray Tune
def train_mistral(config):
    # Apply LoRA with configurable parameters
    model_peft = FastLanguageModel.get_peft_model(
        model,
        r=config["r"],
        target_modules=[
```

```
            "q_proj", "k_proj", "v_proj", "o_proj",
            "gate_proj", "up_proj", "down_proj"
        ],
        lora_alpha=config["lora_alpha"],
        lora_dropout=config["lora_dropout"],
        bias="none",
        use_gradient_checkpointing=True,
        random_state=3407,
        use_rslora=False,
        loftq_config=None,
    )

    # Training arguments with configurable params
    training_args = TrainingArguments(
        per_device_train_batch_size=config["batch_size"],
        gradient_accumulation_steps=config["gradient_accumulation_steps"],
        warmup_steps=config["warmup_steps"],
        max_steps=60,
        learning_rate=config["lr"],
        fp16=not torch.cuda.is_bf16_supported(),
        bf16=torch.cuda.is_bf16_supported(),
        logging_steps=1,
        logging_strategy="steps",
        eval_steps=1,
        save_strategy="no",
        optim="adamw_8bit",
        weight_decay=config["weight_decay"],
        lr_scheduler_type=config["scheduler"],
        seed=3407,
        output_dir=os.path.join(config["output_dir"], tune.get_trial_dir()),
    )

    # Create trainer
    trainer = SFTTrainer(
        model=model_peft,
        tokenizer=tokenizer,
        train_dataset=train_data,
        eval_dataset=val_data,
        dataset_text_field="text",
        max_seq_length=2048,
        dataset_num_proc=2,
        packing=True,
        args=training_args,
    )

    # Start training
    trainer.train()

    # Evaluate and report metrics to Tune
    eval_metrics = trainer.evaluate()
    tune.report(
        eval_loss=eval_metrics["eval_loss"],
        perplexity=float(eval_metrics["eval_loss"]),  # Example metric
    )
```

```
# Define search space
config = {
    "lr": tune.loguniform(1e-5, 1e-3),
    "batch_size": tune.choice([2, 4, 8]),
    "r": tune.choice([8, 16, 32]),
    "lora_alpha": tune.choice([8, 16, 32]),
    "lora_dropout": tune.uniform(0.0, 0.2),
    "weight_decay": tune.uniform(0.0, 0.1),
    "gradient_accumulation_steps": tune.choice([2, 4, 8]),
    "warmup_steps": tune.choice([5, 10, 20]),
    "scheduler": tune.choice(["linear", "cosine", "constant"]),
    "output_dir": "./ray_results",
}

# PBT Scheduler
scheduler = PopulationBasedTraining(
    time_attr="training_iteration",
    metric="eval_loss",
    mode="min",
    perturbation_interval=1,
    hyperparam_mutations={
        "lr": tune.loguniform(1e-5, 1e-3),
        "batch_size": [2, 4, 8],
        "lora_dropout": tune.uniform(0.0, 0.2),
        "weight_decay": tune.uniform(0.0, 0.1),
    }
)

# CLI Reporter
reporter = CLIReporter(
    parameter_columns={
        "lr": "lr",
        "batch_size": "bs",
        "r": "r",
        "lora_alpha": "alpha",
        "weight_decay": "wd",
    },
    metric_columns=["eval_loss", "perplexity", "training_iteration"],
)

# Run Tune experiment
analysis = tune.run(
    train_mistral,
    resources_per_trial={"gpu": 1},
    config=config,
    num_samples=8,  # Number of trials
    scheduler=scheduler,
    progress_reporter=reporter,
    local_dir="./ray_results",
    name="mistral_tuning",
)

print("Best config:", analysis.get_best_config(metric="eval_loss", mode="min"))
```

Dataset augmentation:

- I would use the same inputs but with different languages (German, French) and add some times errors, to make it robust against spelling.

LLM-as-a-Judge Evaluation Framework

I have neither an API to LLM nor a powerfull enough API, but I will describe how I would do it.

The following three judgments could be used as a reference for the LLM:

1. Expectation vs. answer: What domain would the LLM give, given the description of what the trained LLM above was given? Is it far or near enough?
2. Seriousness: Is the given domain serious or rather unserious? The LLM will be able to judge.
3. Usability: Is the domain name meaningful or nonsense?

```
from openai import OpenAI


client = OpenAI(api_key="<DeepSeek API Key>", base_url="https://api.deepseek.com")

response = client.chat.completions.create(
    model="deepseek-chat",
    messages=[
        {"role": "system",
         "content": "You are a domain judger.\
         You will check if the domain fits well to the describtion and\
         you will check if you would have suggested a similiar domain\
         to the given one. You will also check if the domain is a serious\
         one or nonensense. Your feedback is a score, wherby 3/3 means\
         every category is fullified."},
        {"role": "user", "content": "Hello"},
    ],
    stream=False
)
```

4. Edge Case Discovery and Analysis Possible edge cases and failure traps are where the given context does not make the business clear (for example, a rocket could be military or space-related). Language mixing also makes this unclear, as it is not clear in which language the domain should be. If the person speaks a non-Latin language, it can be difficult for the LLM because it may assume that the domain should be in this language too.

The main problem categories were:

1. Ambiguity: Too little context allows for the interpretation of the business or meaning.
2. Language misunderstanding: Using different languages in the same text can lead to

confusion over the target language.

One solution is to feed the network with exactly such cases. For example, it would be good to add Chinese prompts and even mixed prompts (Chinese with English terms).

Safety Guardrails

To adress this, the LLM could be trained with illegal prompts of which the output is then instead of the domain a warning.

Beginnen Sie mit dem Programmieren oder generieren Sie Code mit KI.