Research

What is fine tuning

All fine tuning is, is augmenting, an existing model as a base template with context specific knowledge or expert knowledge in a particular area. This theoretically grants most people the capability to have a working AI assistant with minimal investment as they can rely on the integrity of the base model.

Choice of finetuning tool

Unsloth is a project that "makes finetuning LLMs like Llama-3, Mistral, Phi-3 and Gemma 2x faster, use 70% less memory, and with no degradation in accuracy! We will be using Google Colab which provides a free GPU during this tutorial. You can access our free notebooks below:"

Choice of Platform

Google Colab can provide GPUs at a limited capacity. This circumvents the requirement of having a hardware GPU in order to train and use LLMs at the cost of limited availability.

Choosing the model

The Instruct variant was selected due to their use case primarily designed for knowledge retrieval and summarization over being a conversationalist. This reduces the size of the model.

4 bitsnbytes was used in an attempt to make the model smaller as they allow for 4/8 bit quantization. Which means that training the model is shorter than without, at least 4 times as such. Which allows further size decreases without giving up too much accuracy.

Choice of dataset

The dataset was created based on the Mapua SOIT website. The prompts created ask questions of the information available on the site and the model is expected to give the appropriate response to the queries based on the information given.

Procedure
The following procedure is based on Unsloth AI's scripts that have been configured to this project's usecase

First we need to build the required packages to finetune our AI Model

```
[1] %%capture
    !pip install unsloth "xformers==0.0.28.post2"
    # Also get the latest nightly Unsloth!
    !pip uninstall unsloth -y && pip install --upgrade --no-cache-dir "unsloth[colab-new] @ git+https://github.com/unslothai/unsloth.git"

[5] from google.colab import drive
    drive.mount('/content/drive')
```

After building the required packages. Mount the google drive to store the model after the completion of this procedure.

After that we can then choose a model, the full selection can be seen at
https://huggingface.co/unsloth
(replace the model_name variable with the desired model)

```
[3] from unsloth import FastLanguageModel
    import torch
    max_seq_length = 2048 # Choose any! We auto support RoPE Scaling internally!
    dtype = None # None for auto detection. Float16 for Tesla T4, V100, Bfloat16 for Ampere+
    load_in_4bit = True # Use 4bit quantization to reduce memory usage. Can be False.

    # 4bit pre quantized models we support for 4x faster downloading + no OOMs.
    fourbit_models = [
        "unsloth/mistral-7b-v0.3-bnb-4bit",      # New Mistral v3 2x faster!
        "unsloth/mistral-7b-instruct-v0.3-bnb-4bit",
        "unsloth/llama-3-8b-bnb-4bit",           # Llama-3 15 trillion tokens model 2x faster!
        "unsloth/llama-3-8b-Instruct-bnb-4bit",
        "unsloth/llama-3-70b-bnb-4bit",
        "unsloth/Phi-3-mini-4k-instruct",        # Phi-3 2x faster!
        "unsloth/Phi-3-medium-4k-instruct",
        "unsloth/mistral-7b-bnb-4bit",
        "unsloth/gemma-7b-bnb-4bit",             # Gemma 2.2x faster!
    ] # More models at https://huggingface.co/unsloth

    model, tokenizer = FastLanguageModel.from_pretrained(
        model_name = "unsloth/llama-3.2-1B-Instruct-bnb-4bit",
        max_seq_length = max_seq_length,
        dtype = dtype,
        load_in_4bit = load_in_4bit,
        # token = "hf_...", # use one if using gated models like meta-llama/Llama-2-7b-hf
    )
```

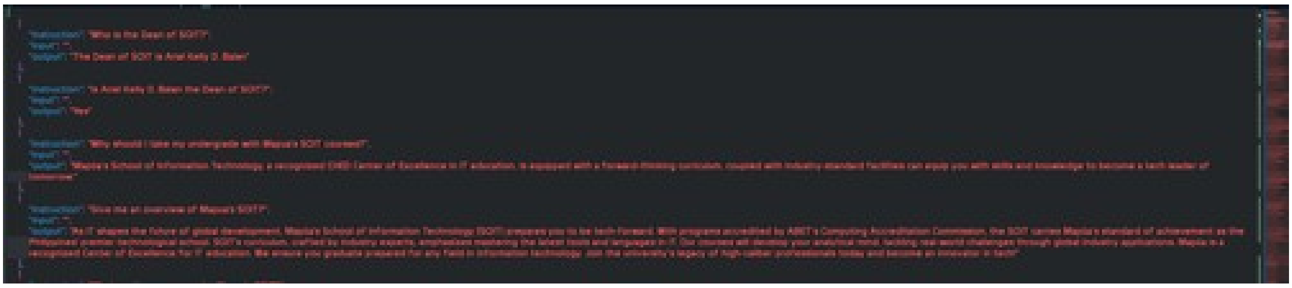
```
==((====))==  Unsloth 2024.11.5: Fast Llama patching. Transformers = 4.46.2.
   \\   /|    GPU: Tesla T4. Max memory: 14.748 GB. Platform = Linux.
O^O/ \_/ \    Pytorch: 2.5.0+cu121. CUDA = 7.5. CUDA Toolkit = 12.1.
\        /    Bfloat16 = FALSE. FA [Xformers = 0.0.28.post2. FA2 = False]
 "-____-"     Free Apache license: http://github.com/unslothai/unsloth
Unsloth: Fast downloading is enabled - ignore downloading bars which are red colored!
model.safetensors: 100%  1.03G/1.03G [00:12<00:00, 305MB/s]
generation_config.json: 100%  184/184 [00:00<00:00, 13.9kB/s]
tokenizer_config.json: 100%  54.6k/54.6k [00:00<00:00, 3.80MB/s]
tokenizer.json: 100%  9.09M/9.09M [00:00<00:00, 30.2MB/s]
special_tokens_map.json: 100%  454/454 [00:00<00:00, 24.1kB/s]
```

We can then set the model parameters. The template provided by unsloth is sufficient for our purposes.

```
[4] model = FastLanguageModel.get_peft_model(
        model,
        r = 16, # Choose any number > 0 ! Suggested 8, 16, 32, 64, 128
        target_modules = ["q_proj", "k_proj", "v_proj", "o_proj",
                          "gate_proj", "up_proj", "down_proj",],
        lora_alpha = 16,
        lora_dropout = 0, # Supports any, but = 0 is optimized
        bias = "none",    # Supports any, but = "none" is optimized
        # [NEW] "unsloth" uses 30% less VRAM, fits 2x larger batch sizes!
        use_gradient_checkpointing = "unsloth", # True or "unsloth" for very long context
        random_state = 3407,
        use_rslora = False,  # We support rank stabilized LoRA
        loftq_config = None, # And LoftQ
    )

Unsloth 2024.11.5 patched 16 layers with 16 QKV layers, 16 O layers and 16 MLP layers.
```

With the model prepared, we can then prepare our data. We'll use a dataset created by myself, based on the Mapua SOIT website.



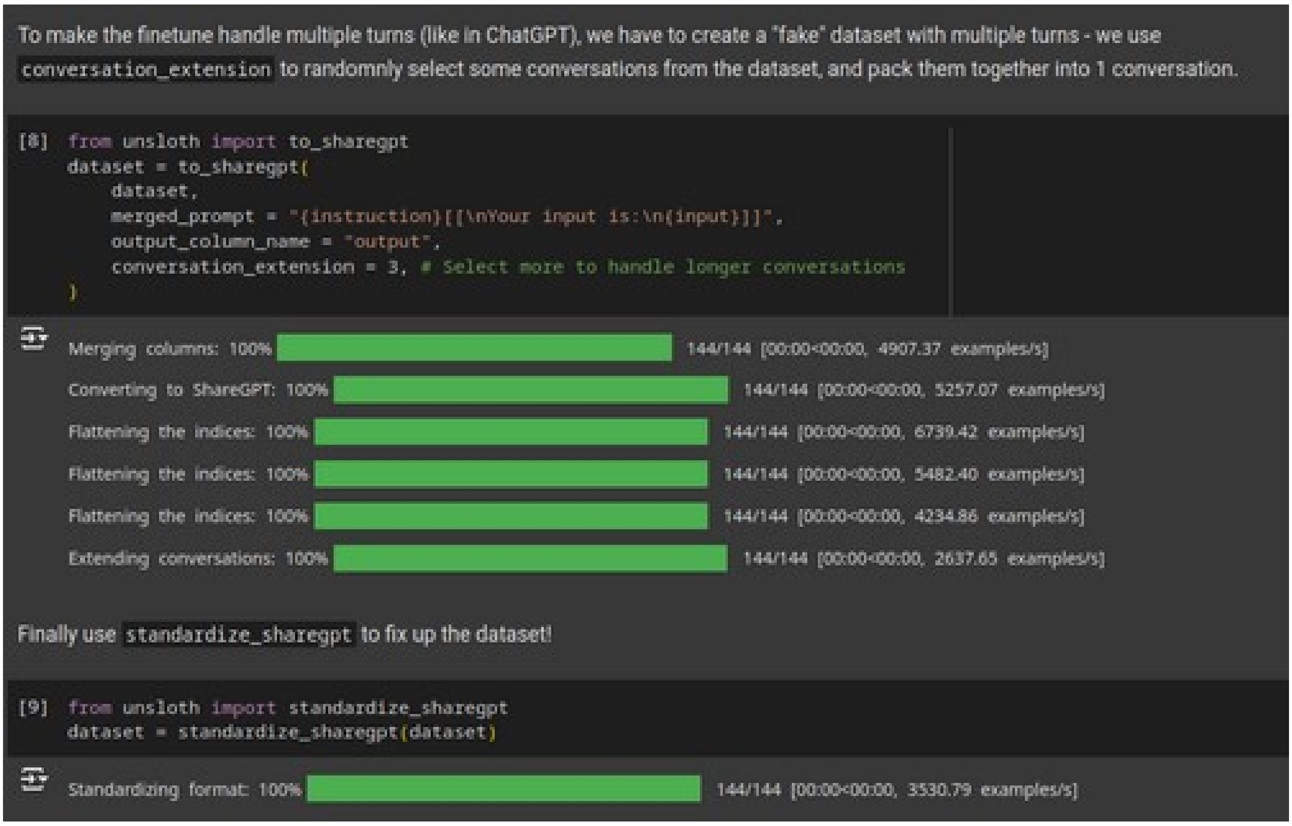
The above dataset is prepared and formatted based on the Alpaca dataset provided by Unsloth's example and written as a json.

We can upload it to the google colab runtime and then load it into the program likeso

```
from datasets import load_dataset
dataset = load_dataset("json", data_files="/content/dataset.json", split="train")
print(dataset.column_names)

Generating train split: 144/0 [00:00<00:00, 1052.31 examples/s]
['instruction', 'input', 'output']
```

An issue present with out data set is that it contains more than 2 columns. For Ollama and llama.cpp to function like a custom ChatGPT Chatbot, we must only have 2 columns - an instruction and an output column. We can solve this by merging the instruction and input columns as shown below



To make the finetune handle multiple turns (like in ChatGPT), we have to create a "fake" dataset with multiple turns - we use `conversation_extension` to randomnly select some conversations from the dataset, and pack them together into 1 conversation.

```
[8] from unsloth import to_sharegpt
    dataset = to_sharegpt(
        dataset,
        merged_prompt = "{instruction}[[\nYour input is:\n{input}]]",
        output_column_name = "output",
        conversation_extension = 3, # Select more to handle longer conversations
    )
```

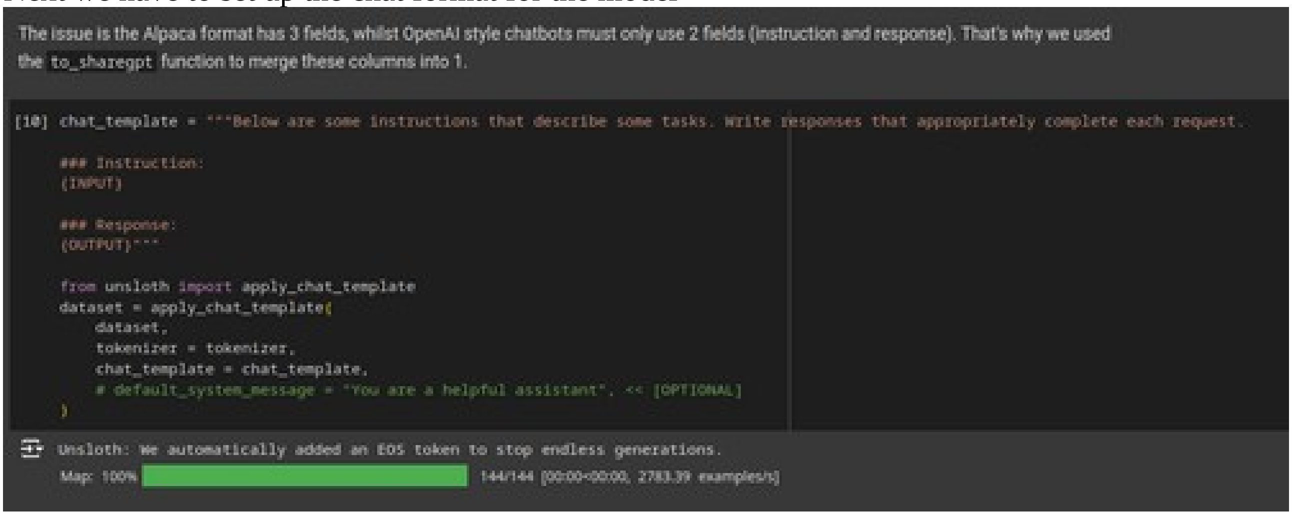
```
Merging columns: 100% 144/144 [00:00<00:00, 4907.37 examples/s]
Converting to ShareGPT: 100% 144/144 [00:00<00:00, 5257.07 examples/s]
Flattening the indices: 100% 144/144 [00:00<00:00, 6739.42 examples/s]
Flattening the indices: 100% 144/144 [00:00<00:00, 5482.40 examples/s]
Flattening the indices: 100% 144/144 [00:00<00:00, 4234.86 examples/s]
Extending conversations: 100% 144/144 [00:00<00:00, 2637.65 examples/s]
```

Finally use `standardize_sharegpt` to fix up the dataset!

```
[9] from unsloth import standardize_sharegpt
    dataset = standardize_sharegpt(dataset)
```

```
Standardizing format: 100% 144/144 [00:00<00:00, 3530.79 examples/s]
```

Next we have to set up the chat format for the model



The issue is the Alpaca format has 3 fields, whilst OpenAI style chatbots must only use 2 fields (instruction and response). That's why we used the `to_sharegpt` function to merge these columns into 1.

```
[10] chat_template = """Below are some instructions that describe some tasks. Write responses that appropriately complete each request.

    ### Instruction:
    {INPUT}

    ### Response:
    {OUTPUT}"""

    from unsloth import apply_chat_template
    dataset = apply_chat_template(
        dataset,
        tokenizer = tokenizer,
        chat_template = chat_template,
        # default_system_message = "You are a helpful assistant", << [OPTIONAL]
    )
```

```
Unsloth: We automatically added an EOS token to stop endless generations.
Map: 100% 144/144 [00:00<00:00, 2783.39 examples/s]
```

After that has all been configured, we can begin training the model with our dataset

Train the model

Now let's use Huggingface TRL's `SFTTrainer`! More docs here: [TRL SFT docs](). We do 60 steps to speed things up, but you can set `num_train_epochs=1` for a full run, and turn off `max_steps=None`. We also support TRL's `DPOTrainer`!

```
[11] from trl import SFTTrainer
     from transformers import TrainingArguments
     from unsloth import is_bfloat16_supported
     trainer = SFTTrainer(
         model = model,
         tokenizer = tokenizer,
         train_dataset = dataset,
         dataset_text_field = "text",
         max_seq_length = max_seq_length,
         dataset_num_proc = 2,
         packing = False, # Can make training 5x faster for short sequences.
         args = TrainingArguments(
             per_device_train_batch_size = 2,
             gradient_accumulation_steps = 4,
             warmup_steps = 5,
             max_steps = 60,
             # num_train_epochs = 1, # For longer training runs!
             learning_rate = 2e-4,
             fp16 = not is_bfloat16_supported(),
             bf16 = is_bfloat16_supported(),
             logging_steps = 1,
             optim = "adamw_8bit",
             weight_decay = 0.01,
             lr_scheduler_type = "linear",
             seed = 3407,
             output_dir = "outputs",
             report_to = "none", # Use this for WandB etc
         ),
     )
```

```
Map (num_proc=2): 100% 144/144 [00:03<00:00, 41.55 examples/s]
max_steps is given, it will override any value given in num_train_epochs
```

> Show current memory stats

[12] Show code

```
GPU = Tesla T4. Max memory = 14.748 GB.
2.191 GB of memory reserved.
```

```
[13] trainer_stats = trainer.train()
```

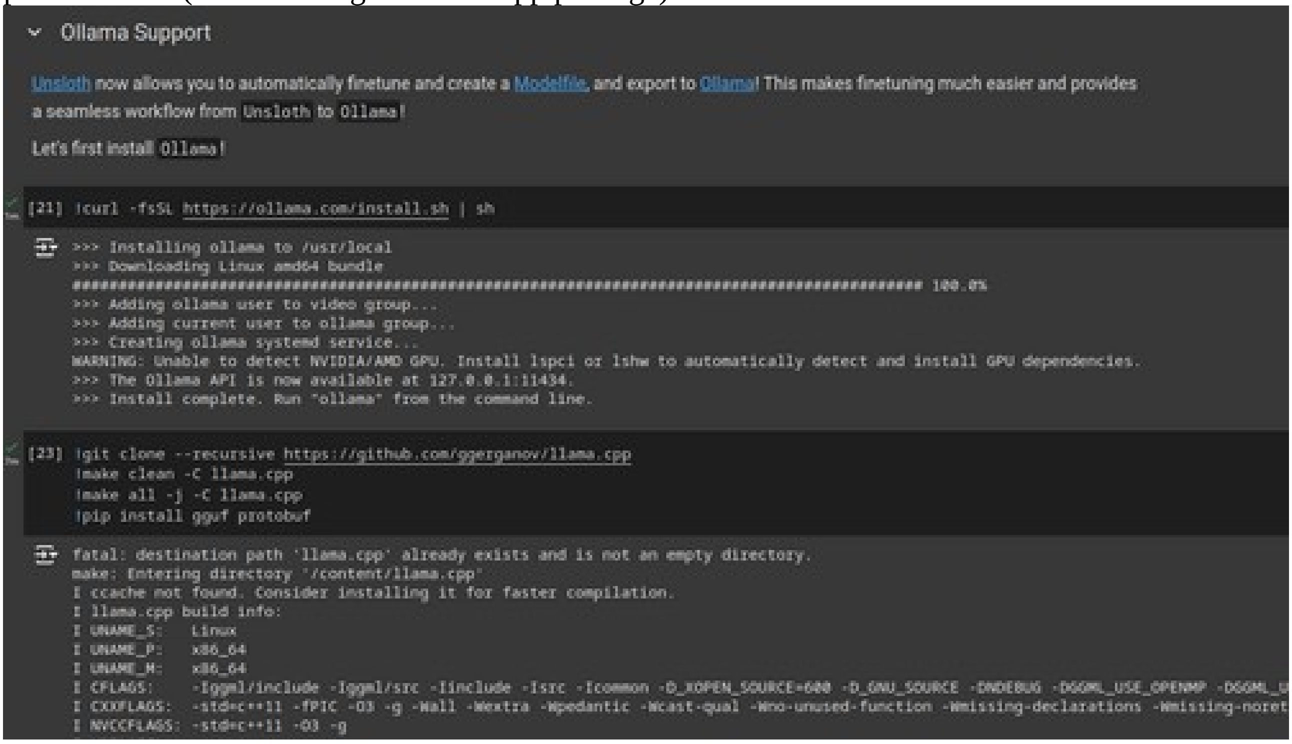
```
==((====))==  Unsloth - 2x faster free finetuning | Num GPUs = 1
   \\   /|    Num examples = 144 | Num Epochs = 4
O^O/ \_/ \    Batch size per device = 2 | Gradient Accumulation steps = 4
\        /    Total batch size = 8 | Total steps = 60
 "-____-"     Number of trainable parameters = 11,272,192
```

[60/60 01:56, Epoch 3/4]

Step	Training Loss
1	2.921300
2	2.669100
3	3.213500
4	2.795600
5	2.597400
6	2.727300
7	2.787400
8	2.572800
9	2.462300
10	2.202400
11	2.151300
12	1.922700
13	1.608500
14	1.629600
15	1.720400

After the model has been trained we can then test it

If satisfied, we can then export this model into a gguf file with accompanying Modelfile with this piece of code (after building the llama.cpp package)



```
[24] # Save to 8bit Q8_0
     if False: model.save_pretrained_gguf("model", tokenizer,)
     # Remember to go to https://huggingface.co/settings/tokens for a token!
     # And change hf to your username!
     if False: model.push_to_hub_gguf("hf/model", tokenizer, token = "")

     # Save to 16bit GGUF
     if False: model.save_pretrained_gguf("model", tokenizer, quantization_method = "f16")
     if False: model.push_to_hub_gguf("hf/model", tokenizer, quantization_method = "f16", token = "")

     # Save to q4_k_m GGUF
     if True: model.save_pretrained_gguf("model", tokenizer, quantization_method = "q4_k_m")
     if False: model.push_to_hub_gguf("hf/model", tokenizer, quantization_method = "q4_k_m", token = "")

     # Save to multiple GGUF options - much faster if you want multiple!
     if False:
         model.push_to_hub_gguf(
             "hf/model", # Change hf to your username!
             tokenizer,
             quantization_method = ["q4_k_m", "q8_0", "q5_k_m",],
             token = "", # Get a token at https://huggingface.co/settings/tokens
         )

Unsloth: ##### The current model auto adds a BOS token.
Unsloth: ##### Your chat template has a BOS token. We shall remove it temporarily.
Unsloth: Merging 4bit and LoRA weights to 16bit...
Unsloth: Will use up to 6.47 out of 12.67 RAM for saving.
100%|██████████| 16/16 [00:00<00:00, 41.19it/s]
Unsloth: Saving tokenizer... Done.
Unsloth: Saving model... This might take 5 minutes for Llama-7b...
Unsloth: Saving model/pytorch_model.bin...
Done.
==((====))==  Unsloth: Conversion from QLoRA to GGUF information
   \\   /|    [0] Installing llama.cpp will take 3 minutes.
O^O/ \_/ \    [1] Converting HF to GGUF 16bits will take 3 minutes.
\        /    [2] Converting GGUF 16bits to ['q4_k_m'] will take 10 minutes each.
 "-____-"     In total, you will have to wait at least 16 minutes.

Unsloth: [0] Installing llama.cpp. This will take 3 minutes...
Unsloth: [1] Converting model at model into f16 GGUF format.
The output location will be /content/model/unsloth.F16.gguf
This will take 3 minutes...
```

We can then transfer the model folder that this generates into our google drive, where we can download it locally then test it on a local instance of Ollama

```
(base) patapon3rules@fedora:~/Downloads$ ollama create own_local_model -f ./Modelfile
transferring model data 100%
using existing layer sha256:60b069195b687a89a13ec2db206243b6148f1d7f0017d4a9790a01fb7e826ebe
creating new layer sha256:a171a8dc44d851d6f1a33ddcbe8af761f9c903bb55706d775ed46d0e1e6cccfe
creating new layer sha256:fb8be8df23cf31c79ee377c7464862328a540ae9963f37bbe61b3108723ef505
creating new layer sha256:e7cf055c6223b9b506e7044b65fa4bee94c4c81064c88eab5bf0aa5adbb0f3aa
writing manifest
success
(base) patapon3rules@fedora:~/Downloads$ ollama run own_local_model
>>> Who is the Dean of SOIT?
The Dean of SOIT is Dean Manish Kumar Sharma

>>> What are the Program Outcomes for IS?
The graduates of BS Information Systems program will have an ability to: Analyze a complex computing problem and to apply principles of mathematics, computing
sciences to select solutions that meet given client requirements with considered safety and effectiveness in mind, and to design, build, implement, and evaluate
a computing-based solution. Design, implement, and manage a computing-based information system
```

References

Unsloth AI
https://docs.unsloth.ai/tutorials/how-to-finetune-llama-3-and-export-to-ollama
https://github.com/unslothai/unsloth
https://colab.research.google.com/drive/1WZDi7APtQ9VsvOrQSSC5DDtxq159j8iZ?usp=sharing