Models built and uploaded onto HuggingFace:

1024 tokens: <https://huggingface.co/ccfai/squirmy-the-chatbot-v4>

256 tokens: <https://huggingface.co/ccfai/squirmy-the-chatbot-v5>

512 tokens: <https://huggingface.co/ccfai/squirmy-the-chatbot-v6>

NOTE: if you wish to try and use the model, you’ll need to either request access/send Chun Fai your username on HuggingFace

Model chosen from Meta Tools to build chatbot:

**meta-llama/Llama-2-7b-chat-hf**

Why Llama 2 instead of Llama 3?

There were more existing projects, videos, and articles about Llama 2 than Llama 3 which better facilitated our research and understanding of how it works in general. The release of Llama 3 model was still quite recent, thus not many resources regarding a chatbot or text generation were available at this point of time.

Why not models with more parameters?

Computational resources limitations… ☹

Why Llama-2-7b-**chat** instead of standard Llama-2-7b?

The chat version is specifically designed and trained for conversational interactions; thus, it can understand the nuances of dialogue flow, including:

* Intent recognition: better grasps the purpose behind user prompts.
* Response generation: excels at crafting natural and engaging responses tailored to the conversation.
* Improved helpfulness: delivers responses that are more likely to be useful and address user needs.

**Configurations used to build the model:**

Data fed is the final.csv, obtained after performing data preprocessing from convo.ipynb.

|  |  |
| --- | --- |
| Bitsandbytes configuration | |
| A python library designed to simplify model quantization. Model quantization is a technique that reduces the precision of the numerical values stored within a large LLM like Llama-2-7b-chat.  These models typically use high-precision data types like 32-bit floating-point numbers. Quantization allows the values to be represented with lower-precision data types, such as 4-bit/8-bit integers.  It can reduce memory footprint as lower precision data types require less storage space. Making this the perfect approach to deploy models on devices with limited memory. (for e.g. running the model on my local GPU) | |
| load\_in\_4bit | True |
| bnb\_4bit\_quan\_type | nf4 |
| bnb\_4bit\_compute\_dtype | bfloat16 |
| bnb\_4bit\_use\_double\_quant | False |

|  |  |
| --- | --- |
| Peft configurations | |
| Introduced a small set of additional parameters that ‘adapt’ the pre-trained model to the new task. Therefore, instead of modifying all the parameters within the pre-trained model, we use a new set of parameters that can significantly reduce the computational cost while achieving performance comparable to full fine-tuning. | |
| lora\_alpha (controls scaling factor applied) | 32 |
| lora\_dropout (dropout rate applied to LoRA weight matrices) | 0 |
| r (determines the dimenstionality of the additional LoRA params, lower rank 🡪 reduces size but might limit adaptability) | 16 |
| Bias (to learn bias terms for LoRA weight or no) | none |
| task\_type | CAUSAL\_LM |

|  |
| --- |
| Training Arguments |
| Training arguments used were pretty standard, either obtained from Llama2 documentation from HuggingFace or through research. The batch size was reduced to 2 because of limited number of data we have. (21 rows)  training\_arguments = TrainingArguments(      output\_dir="./results",      num\_train\_epochs=1,      per\_device\_train\_batch\_size=2,      gradient\_accumulation\_steps=1,      optim="paged\_adamw\_32bit",      save\_steps=0,      logging\_steps=2,      learning\_rate=2e-4,      weight\_decay=0.001,      fp16=False,      bf16=False,      max\_grad\_norm=0.3,      max\_steps=-1,      warmup\_ratio=0.03,      group\_by\_length=True,      lr\_scheduler\_type="cosine",      report\_to="tensorboard"  ) |

Max\_seq\_length chosen was 1024 because 256 and 512 were previously tested, 256 may require more than one prompt to provide more context, 512 sometimes generate outputs that are truncated.