stripe | Stripe Young Scientist & Technology Exhibition

DAVIS COLLEGE

The Impact of Training Data Variations on Accuracy and reliability in Machine Learning Models

By George Joseph

Stand No. 5223
Project No. 9003

Table of Contents

Introduction

Introduction

Project Summary

In 3 years, LLMs have become widely accessible to the public. More AI generated texts, videos and images are appearing online. AI companies use internet scrapers to collect vast amounts of data for machine learning. I investigated how variations in composition and quality of AI training data can influence the factuality and accuracy of the model. More specifically the following questions:

How gaps/missing data from training data can affect a model's ability to provide a factual output.

How AI generated information in a dataset can affect the accuracy and reliability of a model.

I carried out 2 experiments that used fine-tuning to simulate training on a smaller scale. In test 1, a model was fine tuned on a dataset with a gap in its knowledge. Its performance was compared against a control model to see the performance loss. Results show minimal performance differences between control and variant models on a large dataset, but on a reduced dataset, there was losses of upto 12% and higher.

In test 2, I simulate a phenomenon called model collapse. This is when a model is recursively trained on AI generated training data, causing the model's performance to degrade as more iterations were done. In my simulation of this, I did not see this phenomenon. I came to the conclusion that without the randomness and noise found on the internet, model collapse wouldn't happen. However the model did create extra data, showing us the summarisation model was creating new, unchecked data.

My project highlights the need for screening AI training data and ensure data quality is the highest it can be.

Introduction

What is this project about

In this project I investigated how variations in the composition and quality of AI training data can affect the models output and reliability. In simpler terms I wanted to see how making changes to how to AI **training data** (The info a AI model is trained on) is collected, made and structured affect how it learns and how well it affects its abilities.

In particular, I wanted to see how the following scenarios affect the model:

1. How gaps in the training data affect its performance. This is when a subtopic **e.g.** cancer, pain etc is removed from a training dataset of a genre **e.g.** medical information. I also wanted to see how removing a topic affected a model's performance on other topics and how not knowing a topic affected its ability in other areas.

2. How **Synthetic** (AI generated data), affects the models ability and performance. More specifically I test a phenomenon called **Model Collapse**, when a model is trained on highly processed data over multiple iterations

What is this project about

Over the last 4 years, there has been a boom in Artificial Intelligence. These models can produce in multiple **multi-media** formats including text, audio, video and imagery. These models have been spread all over the internet. The problem arises with **web scraping**. This is when bots scan the internet and harvest data. This is used to train AI models on natural language, speech and communication. If this data is not thoroughly screened, there can also be inconsistencies and gaps in the training data. This is where the 2 common problems arise: AI training on AI generated data, and inconsistent, data with gaps. This project can shed vital insight into the affect these problems can have and some ways to mitigate risk.

Introduction

Goals for this project

As I already stated, I tested 2 scenarios. But more specifically I had more detailed criteria

Test 1

I would be comparing a model with knowledge gaps against one without. So my first goal was to see how gaps affect a model's performance against the **control model** (model with no gaps). I wanted to see how the model's performance was affected in these scenarios:

1. Performance on the removed subtopic. Would the model pick up on other dataset patterns or not?
2. Performance on everything else. Did removing a topic affect its performance answering other questions (**overlap, leakage**)?

I also wanted to see how reducing the size of the dataset it was trained on affect the model. This could've prevented the model from picking up general patterns in the dataset. So my second goal was to see how reducing the training dataset by 90% affected the model against the control model.

Test 2

I would be seeing how model collapse occurred on the model over training iterations. My first goal was to see how performance improved or worsened as more iterations were done.

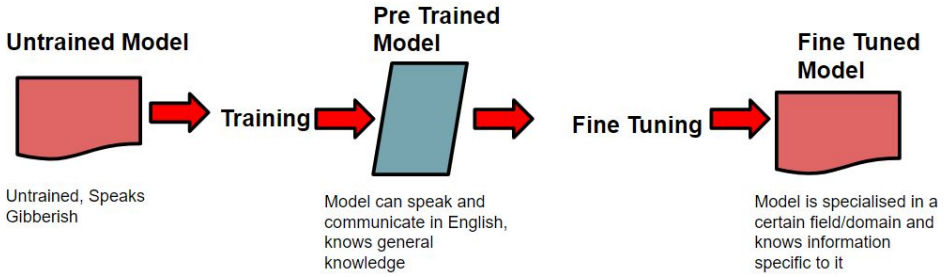
This process would generate a training dataset for each iteration of the experiment. These training datasets comprised of **QA** pairs. My second goal was to see how the amount of QA pairs fluctuated with each iteration.

Methodology

Methodology

Limitations

One very big limitation I had when doing this project was **Computational Resources**. Large AI companies use massive data centers to train these AI models. An average Chatbot **e.g.** Claude, Chat GPT go through these main stages:



The problem I faced was that training needs lots of GPUs to update the model's internal parameters called **weights**. This could take months and is very expensive (in the billions of Euro range). To combat this computational and monetary problem I used **fine-tuning**. This is done after training, so the model already knows how to speak and communicate, and makes it better at doing a certain task **e.g.** Coding, Medical Questions etc. This is done on a **Pre-trained Model**. So now that I had reduced the resources required for training, there was one other issue.

I used Google Colab and Kaggle to fine-tune the models. THese services allowed me to use GPUs for free, at a max quota of 30 hrs a week. Both of these services let me use the **Nvidia T4 GPU** which has 16gb of **VRAM** (Video Random Access Memory). This is where the model is stored and updated. However 16gb isn't enough to fine tune a model. To combat this limitation I used 2 technologies:

1. **Quantisation** - A model comprises of billions of parameters. Normally these are stored in **32 bit** sequences (really precise but takes up a load of space). This technology records it down to 4 bit (small and still accurate to a degree)
2. **Low Rank Adaptation** - This is a complex strategy where the model isn't updated directly but is stored in **matrices** which are applied after training.

Methodology

Environment

All tests were run in a closed off, reproductible environment for the best accuracy and to ensure the only variable being changed was the one being tested. To do this I used the following methods.

1. All randomness in the code was controlled by a **set seed**. This allowed all randomly generated outputs **e.g.** training, shuffling to be reproducible.
2. Dependency installation was automated. An library's the project required was downloaded first in a common notebook block. This made all libraries and external dependencies have the same versions.
3. All tests were run in the same environment. For tests run on kaggle, they used the default baseline runtime and the equivalent was used in Google Colab. This fixed parameters like GPU capacity and performance, storage etc.

This ensured the testing environment was controled and reproducible.

Methodology

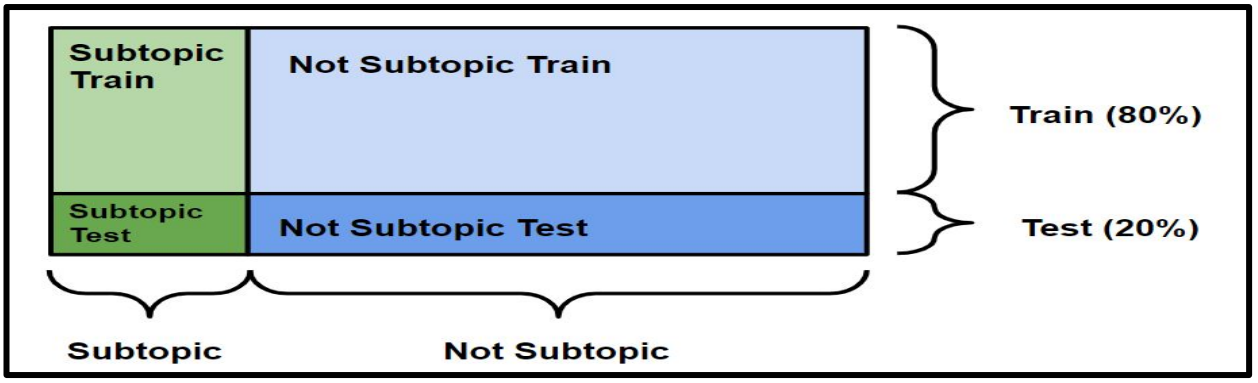
Test 1

Control Variables

- **Model =** "unsloth/gemma-3-4b-it-unsloth-bnb-4bit"
- **Original Dataset** = “FreedomIntelligence/medical-o1-reasoning-SFT”
- **LoRa settings** = See Appendices
- **Training settings** = See Appendices

Steps

1. First the environment was setup and all dependencies were installed to their correct versions for reproducibility.
2. Then the model was initialized using Quantisation and LoRa. Provisions were implemented for crash recovery.
3. Then the dataset was set up. It was split into the following sections:



   It was split up into 2 sections, one which had all the information regarding the subtopic and the other with everything else other than the subtopic. 20% of each section was set aside for testing. The control model had both top 2 training sections while the variant model didn't have the subtopic training section to create a knowledge gap
4. Then the model were both fine-tuned using the same training parameters

Methodology

Steps

5. Then both models are tested in 2 categories as per my goals:
   a. Answering questions about the subtopic
   b. Answering questions that aren't about the subtopic

   Both of these were done using the in-built evaluation tools from the training library. Performance is measured in a variable called "***eval_loss***". This basically see how different the model's response was to the correct answer. THis means the lower the number the closer the model was, hence the lower the better.
6. I then had the control and variant model, with the variant having a knowledge gap. Their performance metrics were stored in a spreadsheet for analysis.
7. This process was repeated 2 more times with different subtopics. This allowed me to assess the impact of different knowledge gap sizes. I tested one at 7% ("Fever"), 15% ("Pain") and 53% ("old").
8. This process was again repeated but with a 90% reduction on the original dataset.

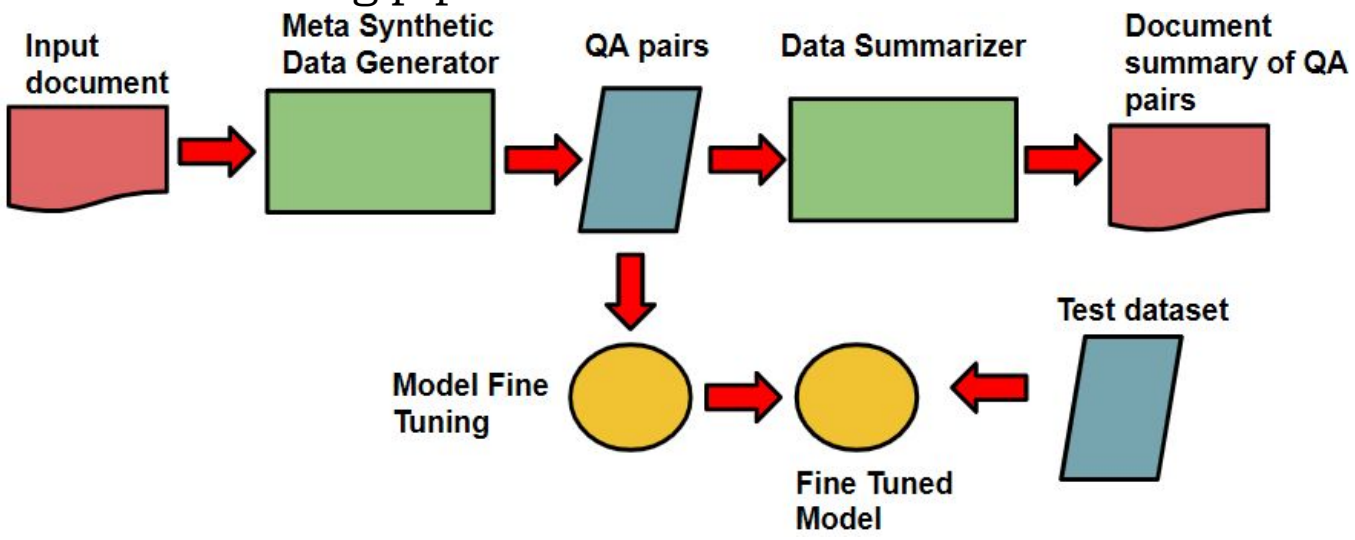
Here were the final metrics:

Topic Name	Sample Type	Training Loss	Subtopic Eval	General Eval	Total Samples	Samples Removed	% Removed
Fever	Control	1.699686962	3.178089619	3.201206446	15762	0	0.00%
Fever	Variant	1.698433899	3.202566862	3.210449934	15762	1120	7.11%
% Diff		**-0.07%**	**0.76%**	**0.29%**			
Old	Control	1.38972706	2.292306185	2.476054907	15762	0	0.00%
Old	Variant	1.458270177	2.311952353	2.463835955	15762	8408	53.34%
% Diff		**4.70%**	**0.85%**	**-0.50%**			
Pain	Control	1.57735	2.385363102	2.295841217	15762	0	0.00%
Pain	Variant	1.53009	2.385864019	2.306565046	15762	2387	15.14%
% Diff		**-3.09%**	**0.02%**	**0.46%**			
			90% Reduction				
Fever	Control	1.963287522	2.442798376	2.627936363	1575	0	0.00%
Fever	Variant	2.334617165	2.789121151	3.00925374	1575	112	7.11%
% Diff		**15.91%**	**12.42%**	**12.67%**			
Old	Control	1.96797	2.694545269	2.477348328	1575	0	0.00%
Old	Variant	2.43412	3.413097858	3.766328812	1575	830	52.70%
% Diff		**19.15%**	**21.05%**	**34.22%**			
Pain	Control	2.89702	2.550717354	2.584369183	1575	0	0.00%
Pain	Variant	2.80968	2.915373087	2.966330528	1575	247	15.68%
% Diff		**-3.11%**	**12.51%**	**12.88%**			

Methodology

Test 2

In this test I experimented with a term called **model collapse**. This is when a model is recursively trained on **synthetic** (AI generated data) causing the model's performance to degrade and the data to lose its original meaning. To test this affect I used the following pipeline:



This pipeline:

1. Takes the input document and converts it into QA pairs to create a training dataset.
2. Fine tunes the model on the dataset.
3. Evaluates its performance.
4. Summarises the dataset back into a long input document

Note: The evaluation dataset is a subset of the first iterations dataset, allowing the first iteration to act as a benchmark

This process occurred over 3 steps:

1. Data Generator
2. Tuner & Tester
3. Data Clumper

Methodology

Test 2

Data Generator

Control variables

- **Model** = *"unsloth/Llama-3.2-3B-Instruct"*
- **Max tokens** = *600 # How much characters the model can generate*
- **Number of pairs per chunk** = 25
- **Summarization prompt** = *See Appendices*
- **Generation prompt** = See Appendices

Steps

1. First I found a suitable starting document. I used The Official GNU C Programming Reference.
2. Then all dependencies were setup and the Meta Synthetic Data Generator was setup using the parameters specified above.
3. The input document was converted into multiple chunks. This breaks down the long document into pieces that the model can handle.
4. Then these chunks are:
   a. Summarised into their key points using the summarisation prompt.
   b. Converted into QA pairs using the generation prompt.

   The output is then stored in the new dataset

Here is an example of a generated pair:

"What type of data can store a float value in C? The float data type

"You are a helpful assistant.Can arrays in C store zero elements? Yes, arrays in C can store zero elements"

Methodology

Test 2

Tuner and Tester

Control variables

- **Model** = *"unsloth/Llama-3.2-3B-Instruct"*
- **LoRa settings** = See Appendices
- **Training parameters** = "*See Appendicies*"

Steps

1. First the environment was setup and all dependencies were installed to their correct versions for reproducibility.
2. Then the model was initialized using Quantisation and LoRa. Provisions were implemented for crash recovery.
3. Then the training dataset is loaded. This is comprised of the QA pairs generated by the Data Generator. This type of data is used in **Supervised Fine Tuning** (the type of training I used)
4. Then the model is fine tuned on the dataset using the training parameters
5. The model was evaluated using the inbuilt evaluation tools. The evaluation dataset was a subset of the 1st iterations training dataset to act as a baseline.

Methodology

Test 2

Data Clumper

Control variables

- **Model** = *"unsloth/Llama-3.2-3B-Instruct"*
- **Prompt** = "*See Appendices*"

Steps

1. The training dataset is loaded into the program.
2. Then each example of the dataset was fed into the model. I used a tool called **vLLM** to batch this process and significantly speed it up.
3. This outputted a long text document. This will be the new input for the next iteration of the model collapse cycle

Methodology

Hypotheses

Test 1

- I expected that all control models would outperform the variant models in both subtopic and non-subtopic performance due to the absence of knowledge gaps in their training dataset.
- I also thought that the models trained on the dataset with a 90% reduction would have more of a performance difference between the control and variant compared to the full dataset models. I thought this would happen due to there being less information the could use to learn patterns and get the bigger picture.

Test 2

- I predicted that the model's performance would degrade as more passes through the pipeline were done. I thought that
- The model would summarise that data too much, causin the original meaning to be lost.
- The model would add unnecessary information that would negatively affect the model. This would also lead to the number of pairs to increase

Results & Raw Data

Results and Raw Data

Test 1

Topic Name	Sample Type	Training Loss	Subtopic Eval	General Eval	Total Samples	Samples Removed	% Removed
Fever	Control	1.699686962	3.178089619	3.201206446	15762	0	0.00%
Fever	Variant	1.698433899	3.202566862	3.210449934	15762	1120	7.11%
% Diff		**-0.07%**	**0.76%**	**0.29%**			
Old	Control	1.38972706	2.292306185	2.476054907	15762	0	0.00%
Old	Variant	1.458270177	2.311952353	2.463835955	15762	8408	53.34%
% Diff		**4.70%**	**0.85%**	**-0.50%**			
Pain	Control	1.57735	2.385363102	2.295841217	15762	0	0.00%
Pain	Variant	1.53009	2.385864019	2.306565046	15762	2387	15.14%
% Diff		**-3.09%**	**0.02%**	**0.46%**			

90% Reduction

Fever	Control	1.963287522	2.442798376	2.627936363	1575	0	0.00%
Fever	Variant	2.334617165	2.789121151	3.00925374	1575	112	7.11%
% Diff		**15.91%**	**12.42%**	**12.67%**			
Old	Control	1.96797	2.694545269	2.477348328	1575	0	0.00%
Old	Variant	2.43412	3.413097858	3.766328812	1575	830	52.70%
% Diff		**19.15%**	**21.05%**	**34.22%**			
Pain	Control	2.89702	2.550717354	2.584369183	1575	0	0.00%
Pain	Variant	2.80968	2.915373087	2.966330528	1575	247	15.68%
% Diff		**-3.11%**	**12.51%**	**12.88%**			

Results and Raw Data

Test 2

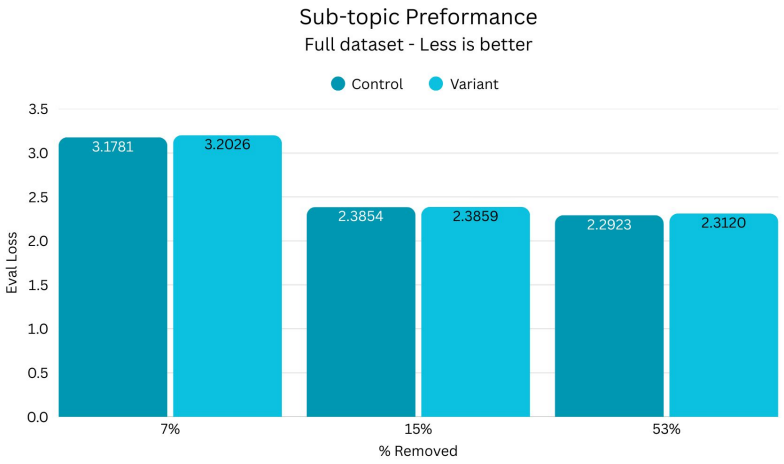
#	Turn	Chunks	Pairs	Eval Loss
	1	53	989	1.528052688
	2	87	1244	1.708606958
	3	87	830	1.67955339
	4	81	1244	1.738954782
	5	87	1337	1.738954782
	6		2274	1.707172632
	7			
	8			
	9			
	10			

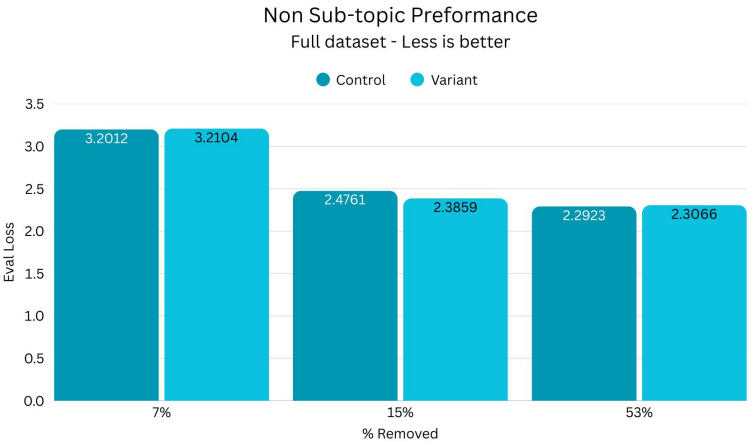
Analysis

Analysis

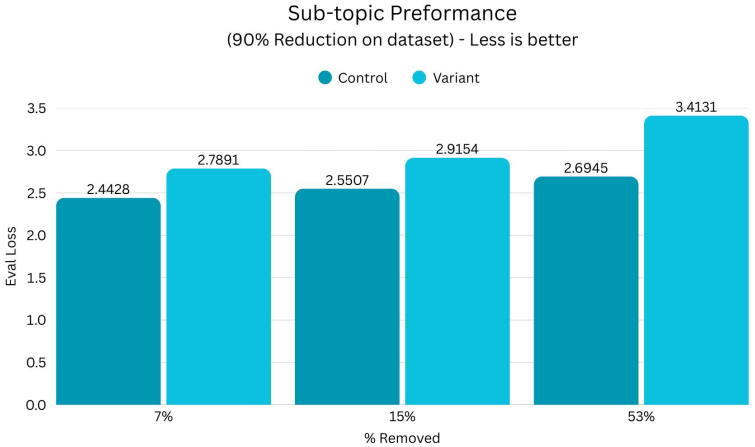
Test 1

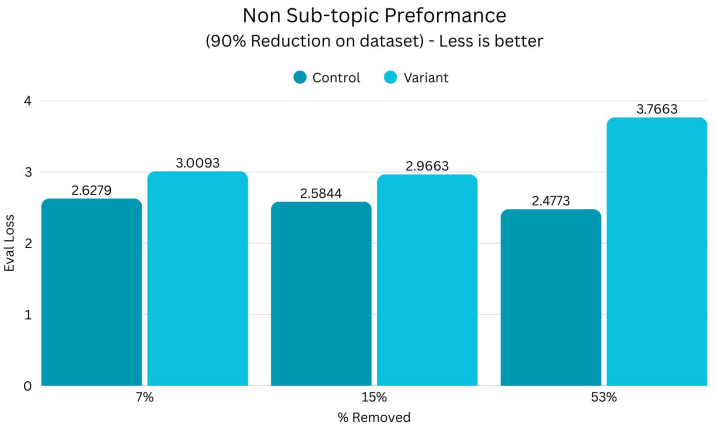
First there is the tests run with the full dataset (no reduction).

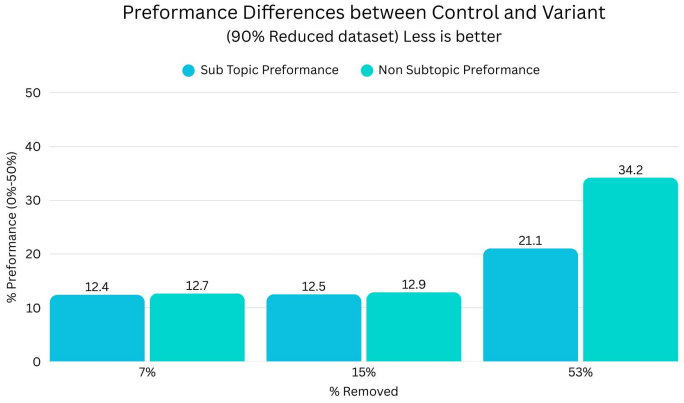




The data indicates a very small difference between the control anc variant models in both subtopic and non-subtopic performance when training using a full original dataset. I believe these subtle variations to occurs due to noise and inefficiencies during training. We also see that the variant performed similarly to the control. This is most likely due to the model being able to fill its knowledge gaps using patterns and trends from the rest of the dataset, which I had predicted. Secondly there is the tests run on the reduced dataset.







Analysis

Test 1

From the data we can clearly see the effect reducing the dataset had on all the gap sizes:

- In both sub-topic and non-subtopic performance the control model with no gaps performs better than the variant at least 12%.
- We can see that with less data, the model could not compensate for its knowledge gap by analyzing the patterns and trends in the dataset.
- In non-subtopic performance, there is a outlier. The control model out-performed the variant model, who had a 53% gap, by 34%. This surprised me as the model did not have gaps in its non-subtopic information, so it shouldn't happen. I think such a big gap developed due to it having only 47% of its training data. Such a big gap may have caused the model to fail to learn patterns, and therefore suffer a massive performance penalty. Removing 53% of all total training data, on an already reduced dataset, is quite a big chunk of information that the model can’t learn from.

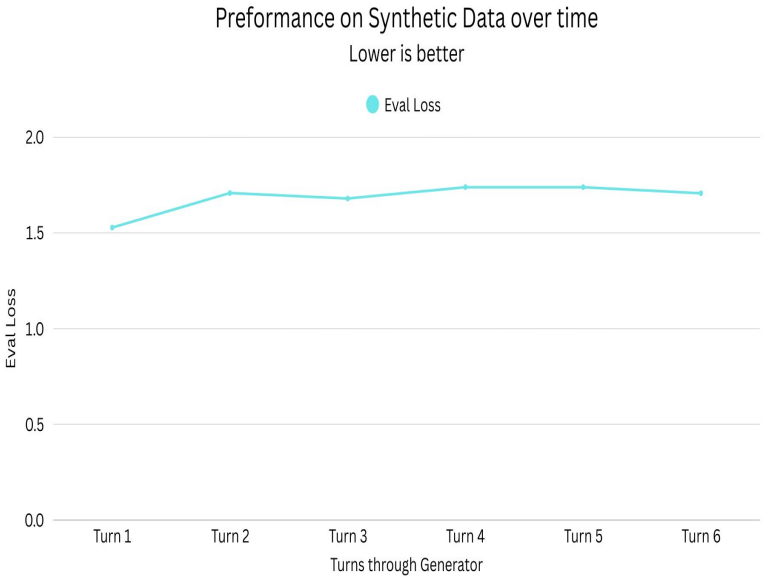
Test 1 Analysis Summary

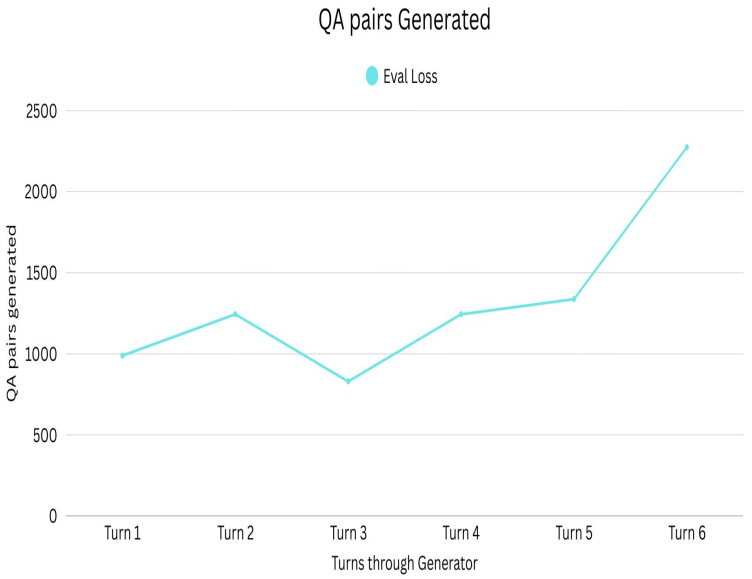
- There was minimal performance differences with all gap sizes with a full dataset. However, reducing the dataset by 90%, caused all gaps to increase to at least 12%.
- It shows us that when the model has more information, it can compensate for the gaps by picking up on patterns in the genera dataset.
- There is also severe losses when removing more the 50% of the dataset, due to their being insufficient data to properly train the model

Analysis

Test 2

Here is the data from test 2:





From this data the results are interesting.

- The eval_loss did not change significantly over 6 iterations and plateaued at around 1.7 This is interesting as according to the phenomenon of model collapse, performance should degrade as more iterations are done.
- I believed model collapse **didn't** occur in my experiment because:
  - The test environment was closed and there wasn't any randomness that wasn't strictly controlled. Model Collapse happens in the real word over the internet. Without these variations and the uncontrolled environment model collapse could have occurred.
- The number of generated pairs steadily increased from 980 to 2200. This lines up with my prediction that the model would create extra filler data.

Conclusion

Conclusion

Takeaways

1. **Screen Data** - This means to check and evaluate the quality of data. In test 1 we can see how having any sized gap in a small training dataset can lead to massive performance penalties. This can be mitigated by screening te data to ensure topics are represented fairly and equally.
2. **Original Data > Synthetic Data** - In test 2 I didn't get a results that aligned with the norm of model collapse. However this doesn't mean there is risks. In a uncontrolled environment model collapse can happen. I recommend:
    a. Dont source training data from the internet. It can be unreliable and nobody knows where this data can from
    b. Dont use synthetic data. Synthetic data is used when there isn't enough data harvest from the original application. Studies have shown that a model can learn effectively with limited training data, making synthetic data redundant
3. **Hallucinations** - When a dataset has gaps it can make stuff up. This can happen a lot. For example when 53% of the dataset was removed in Test 1, there was massive performance dips. The model had to create a plausible answer using very limited examples, most likely causing a hallucination. This shows the importance of being cautious and screening data.

Conclusion

Lessons Learned

Doing this project over the last 2 months has been great fun. I have learned a lot not only about AI and coding, but the scientific method, doing proper experiments and conducting good research. While some things didn't go as planned like the many reruns, bugs and failures, I go here in the end.

With this project I' have highlighted some concerns regarding variations in AI training data, adn how mistakes in its collect, processing and use affect an AI model. With AI models becoming more and more popular, it is important that we ensure our models are factual and don't hallucinate. We must also ensure that we provide AI models with factual information and stay away from patterns like model collapse.

Appendices & Resources

Appendices

Papers

1. De, S. (2023). Importance of Web Scraping as a Data Source for Machine Learning Algorithms - Review. doi:https://doi.org/10.1109/iciis58898.2023.10253502.
2. Kalai, A.T. et al. (2025) Why language models hallucinate. https://arxiv.org/abs/2509.04664.
3. Iskander, S. et al. (2024) Quality Matters: Evaluating Synthetic Data for Tool-Using LLMs. https://arxiv.org/abs/2409.16341.
4. Huang, Y. et al. (2024) 'Key-Point-Driven Data Synthesis with its Enhancement on Mathematical Reasoning,' arXiv (Cornell University) [Preprint]. https://doi.org/10.48550/arxiv.2403.02333.
5. Chen, J., Cai, Z., Ji, K., Wang, X., Liu, W., Wang, R., Hou, J. and Wang, B. (2024). HuatuoGPT-o1, Towards Medical Complex Reasoning with LLMs. [online] arXiv.org. Available at: https://arxiv.org/abs/2412.18925.
6. Arxiv.org. (2020). 60 Data Points are Sufficient to Fine-Tune LLMs for Question-Answering. [online] Available at: https://arxiv.org/html/2409.15825v2.

Tools, Code and Models

7. Unsloth (2025). unsloth/gemma-3-12b-it-unsloth-bnb-4bit · Hugging Face. [online] Huggingface.co. Available at: https://huggingface.co/unsloth/gemma-3-12b-it-unsloth-bnb-4bit.
8. Unsloth (2024). Mistral_(7B)-Text_Completion [online] Available at: https://colab.research.google.com/github/unhttpsslothai/notebooks/blob/main/nb/Mistral_(7B)-Text_Completion.ipynb#scrollTo=ekOmTR1hSNcr.
9. Unsloth (2025). Welcome | Unsloth Documentation. [online] Unsloth.ai. Available at: https://docs.unsloth.ai/
10. Unsloth (2025). Llama3 (8B) [online] Google.com. Available at: https://colab.research.google.com/github/unsloth/notebooks/blob/main/nb/Llama3_%288B%29-Ollama.ipynb#scrollTo=6bZsfBuZDeCL [Accessed 22 Dec. 2025].

Appendices

Tools, Code and Models

11. Unsloth (2025). notebooks/nb/Gemma3_(4B).ipynb at main · unslothai/notebooks. [online] GitHub. Available at: https://github.com/unslothai/notebooks/blob/main/nb/Gemma3_(4B).ipynb [Accessed 22 Dec. 2025].
12. Huggingface (2025). Supervised Fine-tuning Trainer. [online] huggingface.co. Available at: https://huggingface.co/docs/trl/en/sft_trainer.
13. Unsloth (2024a). Meta_Synthetic_Data_Llama3_2_(3B) [online] Google.com. Available at: https://colab.research.google.com/github/unslothai/notebooks/blob/main/nb/Meta_Synthetic_Data_Llama3_2_%283B%29.ipynb#scrollTo=QaJSR1m043dj [Accessed 22 Dec. 2025].
14. GNU (2008). The GNU C Reference Manual. [online] www.gnu.org. Available at: https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html
15. Unsloth (2025). unsloth/gemma-3-4b-it-unsloth-bnb-4bit · Hugging Face. [online] Huggingface.co. Available at: https://huggingface.co/unsloth/gemma-3-4b-it-unsloth-bnb-4bit [Accessed 23 Dec. 2025].

Blogs, Forums and Wikis

16. Unsloth (2025). Gemma 3 - How to Run Guide | Unsloth Documentation. [online] Unsloth.ai. Available at: https://docs.unsloth.ai/models/gemma-3-how-to-run-and-fine-tune.
17. Amit, H. (2025). Fine-Tuning vs Continued Pretraining. [online] Medium. Available at: https://medium.com/@heyamit10/fine-tuning-vs-continued-pretraining-c8058e5040cf.
18. IBM (2024). Model Collapse. [online] Ibm.com. Available at: https://www.ibm.com/think/topics/model-collapse

Appendices

Blogs, Forums and Wikis

19. Unsloth (2025b). Tutorial: How to Finetune Llama-3 and Use In Ollama | Unsloth Documentation. [online] Unsloth.ai. Available at: https://docs.unsloth.ai/get-started/fine-tuning-llms-guide/tutorial-how-to-finetune-llama-3-and-use-in-ollama [Accessed 23 Dec. 2025].
20. Hugging Face (2022). Trainer.evaluate() vs trainer.predict(). [online] Hugging Face Forums. Available at: https://discuss.huggingface.co/t/trainer-evaluate-vs-trainer-predict/28132/3 [Accessed 23 Dec. 2025].
21. Capelle, T. (2023). How to Fine-tune an LLM Part 3: The HuggingFace Trainer. [online] W&B. Available at: https://wandb.ai/capecape/alpaca_ft/reports/How-to-Fine-tune-an-LLM-Part-3-The-HuggingFace-Trainer--Vmlldzo1OTEyNjMy.
22. Rakesh Rajpurohit (2023). Customized Evaluation Metrics with Hugging Face Trainer. [online] Medium. Available at: https://medium.com/@rakeshrajpurohit/customized-evaluation-metrics-with-hugging-face-trainer-3ff00d936f99 [Accessed 23 Dec. 2025].
23. M, V. (2023). What is the difference between Training Loss Validation Loss and Evaluation Loss. [online] Medium. Available at: https://medium.com/@penpencil.blr/what-is-the-difference-between-training-loss-validation-loss-and-evaluation-loss-c169ddeccd59.
24. Hugging Face (2022a). How to save my model to use it later. [online] Hugging Face Forums. Available at: https://discuss.huggingface.co/t/how-to-save-my-model-to-use-it-later/20568 [Accessed 23 Dec. 2025].
25. devzzzero (2024). Is it possible to resume a unsloth QLora Fine tune? If so how? [online] GitHub. Available at: https://github.com/unslothai/unsloth/issues/591 [Accessed 23 Dec. 2025].

Appendices

Blogs, Forums and Wikis

26. Pandey, A. (2025). Accessing file from local machine | Kaggle. [online] Kaggle.com. Available at: https://www.kaggle.com/discussions/general/69663 [Accessed 23 Dec. 2025].
27. Yikun (2025). [doctest]: ValueError: 'aimv2' is already used by a Transformers config, pick another name. [online] GitHub. Available at: https://github.com/vllm-project/vllm-ascend/issues/2046 [Accessed 23 Dec. 2025].

Code and tools directly used and repurposed

28. unslothai (2025). Wiki. [online] GitHub. Available at: https://github.com/unslothai/unsloth/wiki [Accessed 23 Dec. 2025].
29. Kaggle. Kaggle: Free GPU Runtime. [online] www.kaggle.com. Available at: https://kaggle.com.
30. meta-llama (2025). GitHub - meta-llama/synthetic-data-kit: Tool for generating high quality Synthetic datasets. [online] GitHub. Available at: https://github.com/meta-llama/synthetic-data-kit [Accessed 23 Dec. 2025].
31. VLLM Project (2023). vllm-project/vllm. [online] GitHub. Available at: https://github.com/vllm-project/vllm.
32. Google. Google Colab: Free GPU Runtime. [online] www.colab.google.
33. Unsloth. Code and Fine-tuning Notebooks. [online] Available at: https://docs.unsloth.ai/get-started/unsloth-notebooks

Appendices

Test 1

```
**Installation**
"""

resume_training = False

print("---Installing Dependecies---")
import os, re # Import OS and Regex tools
import torch; # Install torch
os.environ["PYTORCH_CUDA_ALLOC_CONF"] =
"expandable_segments:True"
v = re.match(r"[0-9\.]{3,}", str(torch.__version__)).group(0) # Version for
xformers
xformers = "xformers==" + ("0.0.32.post2" if v == "2.8.0" else
"0.0.29.post3")
!python3 -m pip install --no-deps bitsandbytes accelerate {xformers}
peft trl triton cut_cross_entropy unsloth_zoo # Install other ML
dependencies
!pip install sentencepiece protobuf "datasets>=3.4.1,<4.0.0"
"huggingface_hub>=0.34.0" hf_transfer # Install huggingface API
!pip install --no-deps unsloth # Install Unsloth. We use this to speed up
training
!pip install transformers==4.56.2
!pip install --no-deps trl==0.22.2
!pip install "numpy<2.0"
!python -m xformers.info

"""**Model settings**"""

print("---Initialising model---")

from unsloth import FastLanguageModel
import torch
max_seq_length = 2048
dtype = torch.float16
load_in_4bit = True # Quantisation
```

Appendices

Test 1

```
if not resume_training:
    print("Loading fresh model")
    model, tokenizer = FastLanguageModel.from_pretrained(
        model_name = "unsloth/gemma-3-4b-it-unsloth-bnb-4bit", # Base
Model
        max_seq_length = max_seq_length,
        dtype = dtype,
        load_in_4bit = load_in_4bit,
    ) # Returns a tuple with model (actaul neural network) and tokeiser
(text processor)
else:
    print("Loading trained model")
    model, tokenizer = FastLanguageModel.from_pretrained(
        model_name = "/kaggle/input/trained/other/default/1/trained",
# Saved checkpoint in case of a crash
        max_seq_length = max_seq_length,
        dtype = dtype,
        load_in_4bit = load_in_4bit,
    )
"""**Fine tuning settings**"""

if not resume_training:
    print("Applying Adapters")
    model = FastLanguageModel.get_peft_model(
        model,
        r = 16,
        target_modules = ["q_proj", "k_proj", "v_proj", "o_proj",
                          "gate_proj", "up_proj", "down_proj",],
        lora_alpha = 32, # Scale factor. Larger values make the model learn
quicker
        lora_dropout = 0,
        bias = "none",
        use_gradient_checkpointing = "unsloth",
        random_state = 13, # Random number, keep at 0013 for
reproducability
        use_rslora = False,
        loftq_config = None,
```

Appendices

Test 1

```
) # This takes the base model as input and over writes the variabe with
the model but with LoRa adapters onto it
else:
    print("Skipping PEFT adapters since training resuming")
"""**Data Preperation**"""

print("---Loading Data---")
from datasets import load_dataset

dataset =
load_dataset('FreedomIntelligence/medical-o1-reasoning-SFT', 'en',
split = "train") # Original dataset
print(dataset.column_names)
dataset.column_names

from datasets import concatenate_datasets
d_full, unused = dataset.train_test_split(test_size=0.9, seed=13).values()
# Reduced Dataset
# d_full = dataset # Full Dataset

print(f"Full dataset: {d_full.num_rows} rows")

# Replace "old" with the required subtopic
d_test_subtopic = d_full.filter(
    lambda x: "old" in x["Question"].lower()
) # Everything that has cancer mentioned in response
print(f"Subtopic: {d_test_subtopic.num_rows}")

d_test_NOTsubtopic = d_full.filter(
    lambda x: "old" not in x["Question"].lower()
)
print(f"NOT Subtopic: {d_test_NOTsubtopic.num_rows}")
# Since this is the control model, however if we train it and then test it
on the same info it will be pointless so we remove half of the rows abput
the subtopic for testing
```

Appendices

Test 1

```
d_train, d_test_NOTsubtopic =
d_test_NOTsubtopic.train_test_split(test_size=0.2, seed=13).values() #
Set seed
temp, d_test_subtopic = d_test_subtopic.train_test_split(test_size=0.2,
seed=13).values()

d_train = concatenate_datasets([d_train, temp])

print(f"Train dataset: {d_train.num_rows}")
print(f"Subtopic test: {d_test_subtopic.num_rows}")
print(f"NOT Subtopic test: {d_test_NOTsubtopic.num_rows}")

from unsloth.chat_templates import get_chat_template
tokenizer = get_chat_template(
    tokenizer,
    chat_template = "gemma3",
)
from unsloth import to_sharegpt
from unsloth import standardize_sharegpt
from unsloth.chat_templates import get_chat_template

def formatting_prompts_func(examples):
   convos = examples["conversations"]
   texts = [tokenizer.apply_chat_template(convo, tokenize = False,
add_generation_prompt = False, truncation=True,
       padding=True,).removeprefix('<bos>') for convo in convos]
   return { "text" : texts, }
def format_dataset(dataset): # Formats the dataset
  dataset = to_sharegpt(
    dataset,
    merged_prompt="{Question}",
    output_column_name="Response",
    conversation_extension=3,
  )
  dataset = standardize_sharegpt(dataset)
  dataset = dataset.map(formatting_prompts_func, batched = True)
  return dataset
```

Appendices

Test 1

```
d_train = format_dataset(d_train)

d_train[10]["text"]

"""# Training"""

print("---Training Model---")

from trl import SFTTrainer, SFTConfig
trainer = SFTTrainer(
    model = model,
    tokenizer = tokenizer,
    train_dataset = d_train,
    eval_dataset = None,
    args = SFTConfig(
        dataset_text_field="text",
        per_device_train_batch_size=1,
        gradient_accumulation_steps=8,
        per_device_eval_batch_size=4,
        eval_accumulation_steps=1,
        #num_train_epochs = 1,
        learning_rate = 2e-6,  # 2e-6 stable
        warmup_steps = 40,
        lr_scheduler_type = "cosine",
        max_steps=280,
        logging_steps=5,
        optim="adamw_torch_fused",
        weight_decay=0.001,
        seed=13, # Set seed
        report_to="none",
        packing=False,
        gradient_checkpointing=True,
    ),
)
tokenizer.model_max_length = 2048
```

Appendices

Test 1

```
from unsloth.chat_templates import train_on_responses_only
trainer = train_on_responses_only(
    trainer,
    instruction_part = "<start_of_turn>user\n",
    response_part = "<start_of_turn>model\n",
)
# @title Show current memory stats
gpu_stats = torch.cuda.get_device_properties(0)
start_gpu_memory = round(torch.cuda.max_memory_reserved() / 1024
/ 1024 / 1024, 3)
max_memory = round(gpu_stats.total_memory / 1024 / 1024 / 1024, 3)
print(f"GPU = {gpu_stats.name}. Max memory = {max_memory} GB.")
print(f"{start_gpu_memory} GB of memory reserved.")

print("Started Training")
trainer_stats = trainer.train()
print(trainer_stats)
# @title Show final memory and time stats
used_memory = round(torch.cuda.max_memory_reserved() / 1024 /
1024 / 1024, 3)
used_memory_for_lora = round(used_memory - start_gpu_memory, 3)
used_percentage = round(used_memory / max_memory * 100, 3)
lora_percentage = round(used_memory_for_lora / max_memory * 100,
3)
print(f"{trainer_stats.metrics['train_runtime']} seconds used for
training.")
print(
    f"{round(trainer_stats.metrics['train_runtime']/60, 2)} minutes used
for training."
)
print(f"Peak reserved memory = {used_memory} GB.")
print(f"Peak reserved memory for training = {used_memory_for_lora}
GB.")
print(f"Peak reserved memory % of max memory = {used_percentage}
%.")
print(f"Peak reserved memory for training % of max memory =
{lora_percentage} %.")
```

Appendices

Test 1

```
"""# Evaluation and saving"""

from trl import SFTTrainer, SFTConfig
from unsloth.chat_templates import train_on_responses_only
from transformers import DataCollatorForLanguageModeling
from transformers import default_data_collator
from transformers import Trainer, TrainingArguments


def silly_eval(dataset): # Eval fucntion
    dataset = format_dataset(dataset)
    silly = SFTTrainer(
        model = model,
        tokenizer = tokenizer,
        train_dataset = d_train,
        eval_dataset = dataset,
        data_collator=default_data_collator,
        args=SFTConfig(
            packing=False,  # <-- crucial
            report_to="none",
            eval_steps = None,
            eval_strategy = "no"
        )
    )

    hf_trainer = Trainer(
        model=model,
        tokenizer=tokenizer,
        data_collator=silly.data_collator,
        eval_dataset=silly.eval_dataset,
        args=TrainingArguments(
            report_to="none",
            fp16=True,
            per_device_eval_batch_size = 1,
            eval_accumulation_steps = 1,
        )
    )
```

Appendices

Test 1

```
    stats = hf_trainer.evaluate()
    print(stats)

print("Starting Evaluation")
silly_eval(d_test_NOTsubtopic)
silly_eval(d_test_subtopic)
```

Appendicies

Test 2

Summarisation prompt = “You are a C programming expert and know everything about the language. Summarize the main points, structures and ideas and keep it concise. Add filler now and then as well. The input was taken from a SFT fine tuning database. ONLY SUMMARISE THE SECTION FROM THE ASSISTANT INDICATED BY: 'role': 'assistant', DO NOT REFERENCE OR MENTION AND USER-ASSISTANT INTERACTIONS NOR INCLUDE THEM IN YOUR SUMMARY. The summary should simply summarize the assistant explanation
”

Generation prompt = “*You are a pro C coder who knows everything about the C programming language. Create {num_pairs} QA pairs about the given text.*
Express the key points and info in the QA pair. Keep it concise while still retaining the main info. Add filler if necessary to add to your QA pairs
”

Appendices

Test 2 Data Generator

```
# Data Genrator for test 2

turn_no = 4
input_path = "/content/train3_summary.txt"
input_name = "C"
train_out_path = f"/content/output/train{turn_no}"

"""### Installation"""


%%capture
import os
!pip install --upgrade -qqq uv
if "COLAB_" not in "".join(os.environ.keys()):
    # If you're not in Colab, just use pip install!
    !pip install unsloth vllm synthetic-data-kit==0.0.3
else:
    try: import numpy, PIL; get_numpy =
f"numpy=={numpy.__version__}"; get_pil = f"pillow=={PIL.__version__}"
    except: get_numpy = "numpy"; get_pil = "pillow"
    try: import subprocess; is_t4 = "Tesla T4" in
str(subprocess.check_output(["nvidia-smi"]))
    except: is_t4 = False
    get_vllm, get_triton = ("vllm==0.9.2", "triton==3.2.0") if is_t4 else
("vllm==0.10.2", "triton")
    !uv pip install -qqq --upgrade        unsloth {get_vllm} {get_numpy}
{get_pil} torchvision bitsandbytes xformers
    !uv pip install -qqq {get_triton}
    !uv pip install synthetic-data-kit==0.0.3
!uv pip install transformers==4.56.2
!uv pip install --no-deps trl==0.22.2
```

Appendices

```
#@title Colab Extra Install { display-mode: "form" }
%%capture
import os
!pip install --upgrade -qqq uv
if "COLAB_" not in "".join(os.environ.keys()):
    # If you're not in Colab, just use pip install!
    !pip install unsloth vllm
else:
    try: import numpy, PIL; get_numpy =
f"numpy==={numpy.__version__}"; get_pil = f"pillow==={PIL.__version__}"
    except: get_numpy = "numpy"; get_pil = "pillow"
    try: import subprocess; is_t4 = "Tesla T4" in
str(subprocess.check_output(["nvidia-smi"]))
    except: is_t4 = False
    get_vllm, get_triton = ("vllm==0.9.2", "triton==3.2.0") if is_t4 else
("vllm==0.10.2", "triton")
    !uv pip install -qqq --upgrade \
    unsloth {get_vllm} {get_numpy} {get_pil} torchvision bitsandbytes
xformers
    !uv pip install -qqq {get_triton}
    !uv pip install transformers==4.56.2
!uv pip install --no-deps trl==0.22.2

"""# Setup generator

## Load model
This model is used to take the inputed data and trun it into QA pairs we
can use if SFT
"""

from unsloth.dataprep import SyntheticDataKit

generator = SyntheticDataKit.from_pretrained(
    # Choose any model from https://huggingface.co/unsloth
    model_name = "unsloth/Llama-3.2-3B-Instruct",
    max_seq_length = 2048, # Longer sequence lengths will be slower!
)
```

Appendices

```
"""## Generate QA Pairs + Auto clean data
We now use synthetic data kit for question answer pair generation:
"""

generator.prepare_qa_generation(
    output_folder = "data", # Output location of synthetic data
    temperature = 0.7, # Higher temp makes more diverse datases
    top_p = 0.95,
    overlap = 64, # Overlap portion during chunking
    max_generation_tokens = 600, # Can increase for longer QA pairs
)
"""Check if it succeeded:"""

!synthetic-data-kit system-check

"""# Document Parsing
We then take the C standard documentation and turn it into chunks.
These smaller portions are then processed
"""
!synthetic-data-kit \
    -c synthetic_data_kit_config.yaml \
    ingest "/content/train3_summary.txt"

# Truncate document
filenames =
generator.chunk_data(f"data/output/train3_summary.txt")
print(len(filenames), filenames[:3])

"""We get 53 chunks. We then convert these chunks into 25 QA pairs per
chunk. Their size is dependent on the max lentght set above"""

import time
# Process 3 chunks for now -> can increase but slower!
for filename in filenames:
    !synthetic-data-kit \
        -c synthetic_data_kit_config.yaml \
        create {filename} \
        --num-pairs 25 \
        --type "qa"
    time.sleep(2) # Sleep some time to leave some room for processing
```

Appendices

```
"""We now convert the generated datasets into QA formats so we can
load it for finetuning:"""

qa_pairs_filenames = [
    f"data/generated/train3_summary_{i}_qa_pairs.json"
    for i in range(len(filenames))
]
for filename in qa_pairs_filenames:
    !synthetic-data-kit \
        -c synthetic_data_kit_config.yaml \
        save-as {filename} -f ft

"""Let's load up the data and see what the synthetic data looks like!"""

from datasets import Dataset
import pandas as pd
final_filenames = [
    f"data/final/train3_summary_{i}_qa_pairs_ft.json"
    for i in range(len(filenames))
]
conversations = pd.concat([
    pd.read_json(name) for name in final_filenames
]).reset_index(drop = True)

dataset = Dataset.from_pandas(conversations)
dataset[0]

dataset[1]

dataset[350]

"""# Shuffle and Split dataset"""

dataset = dataset.shuffle(seed=13)
"""## Export dataset"""

dataset.save_to_disk("/content/output/datasets/4/train")

generator.cleanup()
```

Appendices

Test 2 Tuner & Tester

```
train_path = /content/train
test_path = /content/test


%%capture
import os, re
if "COLAB_" not in "".join(os.environ.keys()):
    !pip install unsloth
else:
    # Do this only in Colab notebooks! Otherwise use pip install unsloth
    import torch; v = re.match(r"[0-9]{1,}\.[0-9]{1,}",
str(torch.__version__)).group(0)
    xformers = "xformers==" + ("0.0.33.post1" if v=="2.9" else "0.0.32.post2"
if v=="2.8" else "0.0.29.post3")
    !pip install --no-deps bitsandbytes accelerate {xformers} peft trl
triton cut_cross_entropy unsloth_zoo
    !pip install sentencepiece protobuf "datasets==4.3.0"
"huggingface_hub>=0.34.0" hf_transfer
    !pip install --no-deps unsloth
!pip install transformers==4.56.2
!pip install --no-deps trl==0.22.2

from unsloth import FastModel
import torch

model, tokenizer = FastModel.from_pretrained(
    model_name = "unsloth/gemma-3-12b-it-unsloth-bnb-4bit",
    max_seq_length = 2048,
    load_in_4bit = True,
    #full_finetuning = False,
)
```

Appendices

```
model = FastModel.get_peft_model(
    model,
    finetune_language_layers   = True,  # Should leave on!
    finetune_attention_modules = True,  # Attention good for GRPO
    finetune_mlp_modules       = True,  # SHould leave on always!

    r = 16,           # Larger = higher accuracy, but might overfit
    lora_alpha = 32,  # Recommended alpha == r at least
    lora_dropout = 0,
    bias = "none",
    random_state = 13,
)

from unsloth.chat_templates import get_chat_template

tokenizer = get_chat_template(
    tokenizer,
    chat_template = "gemma-3",
)

from datasets import load_from_disk

train = load_from_disk(train_path)
test = load_from_disk(test_path)

print(train)
print(test)

from unsloth.chat_templates import standardize_data_formats

train = standardize_data_formats(train)
test = standardize_data_formats(test)

train[1]
test[1]
```

Appendices

```
def formatting_prompts_func(examples):
   convos = examples["messages"]
   texts = [tokenizer.apply_chat_template(convo, tokenize = False,
add_generation_prompt = False).removeprefix('<bos>') for convo in
convos]
   return { "text" : texts, }

train = train.map(formatting_prompts_func, batched = True)
test = test.map(formatting_prompts_func, batched = True)
train[1]["text"]
test[1]["text"]
from trl import SFTTrainer, SFTConfig
trainer = SFTTrainer(
    model = model,
    tokenizer = tokenizer,
    train_dataset = train,
    eval_dataset = test, # Can set up evaluation!
    args = SFTConfig(
        output_dir = "checkpoints",
      dataset_text_field = "text",
      per_device_train_batch_size = 2,
      gradient_accumulation_steps = 4,
      num_train_epochs = 4,
      per_device_eval_batch_size = 1,
      eval_accumulation_steps = 1,
      learning_rate = 8e-6,
      warmup_steps = 40,
      lr_scheduler_type = "cosine",
      optim = "adamw_torch_fused",
      weight_decay = 0.001,
      logging_steps = 10,
      save_strategy = "steps",
      save_steps = 35,
      gradient_checkpointing = True,
      seed = 13,
      report_to = "none",
      max_seq_length = 2048
    ),
)
```

Appendices

```
from unsloth.chat_templates import train_on_responses_only

trainer = train_on_responses_only(
    trainer,
    instruction_part = "<start_of_turn>user\n",
    response_part = "<start_of_turn>model\n",
)

# @title Show current memory stats
gpu_stats = torch.cuda.get_device_properties(0)
start_gpu_memory = round(torch.cuda.max_memory_reserved() / 1024 / 1024 / 1024, 3)
max_memory = round(gpu_stats.total_memory / 1024 / 1024 / 1024, 3)
print(f"GPU = {gpu_stats.name}. Max memory = {max_memory} GB.")
print(f"{start_gpu_memory} GB of memory reserved.")

stats = trainer.train(resume_from_checkpoint= True)
print(stats)

# @title Show final memory and time stats
used_memory = round(torch.cuda.max_memory_reserved() / 1024 / 1024 / 1024, 3)
used_memory_for_lora = round(used_memory - start_gpu_memory, 3)
used_percentage = round(used_memory / max_memory * 100, 3)
lora_percentage = round(used_memory_for_lora / max_memory * 100, 3)
print(f"{stats.metrics['train_runtime']} seconds used for training.")
print(
    f"{round(stats.metrics['train_runtime']/60, 2)} minutes used for training."
)
print(f"Peak reserved memory = {used_memory} GB.")
print(f"Peak reserved memory for training = {used_memory_for_lora} GB.")
print(f"Peak reserved memory % of max memory = {used_percentage} %.")
print(f"Peak reserved memory for training % of max memory = {lora_percentage} %.")
```

Appendices

```
model.save_pretrained("model/4ep")

trainer.evaluate()
```

Appendices

Test 2 Data Clumper

```
input_path = "/content/train3"
output_path = "/content/train3_summary.txt"

"""## Installation"""


%%capture
import os
!pip install --upgrade -qqq uv
if "COLAB_" not in "".join(os.environ.keys()):
    # If you're not in Colab, just use pip install!
    !pip install unsloth vllm synthetic-data-kit==0.0.3
else:
    try: import numpy, PIL; get_numpy =
f"numpy==={numpy.__version__}"; get_pil = f"pillow==={PIL.__version__}"
    except: get_numpy = "numpy"; get_pil = "pillow"
    try: import subprocess; is_t4 = "Tesla T4" in
str(subprocess.check_output(["nvidia-smi"]))
    except: is_t4 = False
    get_vllm, get_triton = ("vllm==0.9.2", "triton==3.2.0") if is_t4 else
("vllm==0.10.2", "triton")
    !uv pip install -qqq --upgrade        unsloth {get_vllm} {get_numpy}
{get_pil} torchvision bitsandbytes xformers
    !uv pip install -qqq {get_triton}
    !uv pip install synthetic-data-kit==0.0.3
!uv pip install --no-deps trl==0.22.2


%%capture
import os
!pip install --upgrade -qqq uv
if "COLAB_" not in "".join(os.environ.keys()):
    # If you're not in Colab, just use pip install!
    !pip install unsloth vllm
```

Appendices

```
else:
    try: import numpy, PIL; get_numpy =
f"numpy=={numpy.__version__}"; get_pil = f"pillow=={PIL.__version__}"
    except: get_numpy = "numpy"; get_pil = "pillow"
    try: import subprocess; is_t4 = "Tesla T4" in
str(subprocess.check_output(["nvidia-smi"]))
    except: is_t4 = False
    get_vllm, get_triton = ("vllm==0.9.2", "triton==3.2.0") if is_t4 else
("vllm==0.10.2", "triton")
    !uv pip install -qqq --upgrade \
        unsloth {get_vllm} {get_numpy} {get_pil} torchvision bitsandbytes
xformers
    !uv pip install -qqq {get_triton}
!uv pip install --no-deps trl==0.22.2
!uv pip install "transformers<4.54.0"
!uv pip install torch

"""## Summarize"""

alpaca_prompt = """Below is an instruction that describes a task,
paired with an input that provides further context. Write a response
that appropriately completes the request.

### Instruction:
{}

### Input:
{}

### Response:
{}"""

from datasets import load_from_disk

train = load_from_disk(input_path)

print(train['messages'][1])
```

Appendices

```
instruction = "You are a C programming expert and know everything about the language. Summarize the input provided while keeping in the main points, structures and ideas. The input was taken from a SFT fine tuning database The summary should be at least 3  lines long. Add fluff to the summary to keep it consise. ONLY SUMMARISE THE SECTION FROM THE ASSISTANT INDICATED BY: 'role': 'assistant', DO NOT REFERNCE OR MENTION AND USER-ASSISTANT INTERACTIONS NOR INCLUDE THEM IN YOUR SUMMARY. The summary should simply summarize the assistant explanantion"
input = train['messages'][1]

from vllm import LLM, SamplingParams

# Load vLLM model (fast!)
llm = LLM(
    model="unsloth/Llama-3.2-3B-Instruct",
    max_model_len=4096,
    gpu_memory_utilization=0.9       # use GPU more efficiently
    )
params = SamplingParams(max_tokens=64, temperature=0.1)

messages = train["messages"]
N = train.num_rows
BATCH = 32                  # increase if you have strong GPU

with open(output_path, "a") as f:
    for i in range(0, N, BATCH):
        batch_msgs = messages[i : i + BATCH]
        # Build prompts
        prompts = [
            alpaca_prompt.format(instruction, msg, "")
            for msg in batch_msgs
        ]
        # vLLM generates the batch
        outputs = llm.generate(prompts, params)
        # Extract text from vLLM result format
        for out in outputs:
            response = out.outputs[0].text.strip()
            f.write(response + "\n")
```

stripe

Stripe Young Scientist
& Technology Exhibition

DAVIS COLLEGE

Stand No. 5223
Project No. 9003