Better Retrieval for Generation

CS6803 - Topics in NLP
Dr. Maunendra Sankar Desarkar

Group 19
Aaryan - CO21BTECH11001
Abhishek Kumar - AI21BTECH11003

Code is available here: https://github.com/aaryan200/Topics-in-NLP-Project

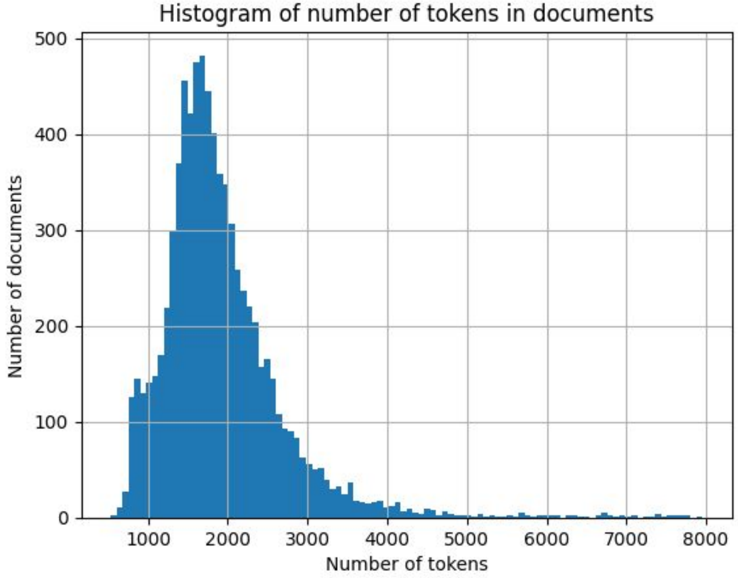
Problem Statement

- Experiments on fine-tuning embedding models on domain specific dataset to improve retrieval
- Multimodal RAG (uniMUR) - Experimentation

Embedding fine-tuning experiments

Dataset and EDA

- MedQuAD dataset:
  - **7873** reachable document urls
  - **36925** questions
- Scraping and cleaning of webpages
  - Using **BeautifulSoup**
  - Removed extra line characters and whitespaces
- Number of tokens in documents (nomic tokenizer):
  - Min: **531**, Max: **7952**
  - Mean: **1918**, Median: **1770**



- MedQuAD: Ben Abacha, A., Demner-Fushman, D. A question-entailment approach to question answering. BMC Bioinformatics 20, 511 (2019). https://doi.org/10.1186/s12859-019-3119-4
- Zach Nussbaum, John X. Morris, Brandon Duderstadt, and Andriy Mulyar. Nomic embed: Training a reproducible long context text embedder, 2024

Evaluation Setup

- If required, documents are **chunked**
- Embeddings of documents are **pre-computed** and stored
- Chunks are retrieved using **cosine-similarity**
- Metrics
  - Recall @ k

$$\text{Recall}-k = \frac{\text{Number of relevant chunks in top } k}{\text{Number of relevant chunks}}$$

  - MRR @ k

$$\text{MRR}-k = \sum_{\text{chunk} \in \{\text{relevant chunks}\}} \frac{1}{\text{Index of chunk in top k retrieved chunks}}$$

- Average number of relevant chunks per question is 3.7, we have used K $\in$ {1, 3, 10}

Final dataset

	question	relevant_docs_urls	num_rel_chunks
0	What is (are) keratoderma with woolly hair ?	[https://ghr.nlm.nih.gov/condition/keratoderma...	5
1	How many people are affected by keratoderma wi...	[https://ghr.nlm.nih.gov/condition/keratoderma...	5
2	What are the genetic changes related to kerato...	[https://ghr.nlm.nih.gov/condition/keratoderma...	5

Dataframe containing question, urls of relevant documents

	doc_url	chunk_content	embedding
0	https://ghr.nlm.nih.gov/condition/keratoderma-...	keratoderma with woolly hair : medlineplus gen...	[-0.0039987266, 0.08037464, 0.049785912, -0.12...
1	https://ghr.nlm.nih.gov/condition/keratoderma-...	##ma, woolly hair, and a form of cardiomyopath...	[-0.09539697, -0.09132044, 0.0027289127, 0.005...
2	https://ghr.nlm.nih.gov/condition/keratoderma-...	##pathy in people with this group of condition...	[0.026278932, 0.060939535, 0.031438153, -0.044...

Dataframe containing document url, chunk content and embedding

Evaluation of pre-trained models

- paraphrase-mpnet-base:
  - Context Window: 512, 109M parameters
  - 33545 chunks formed
- nomic-embed-text , bge-m3
  - Context Window: 8192
  - No need of chunking

Embedding Model	MRR@1	MRR@3	MRR@10	R@1	R@3	R@10
paraphrase-mpnet-base	78.83	84.79	85.57	19.55	38.02	52.22
nomic-embed-text-v1	89.02	92.98	93.18	88.60	97.76	**99.19**
bge-m3	**90.02**	**93.64**	**93.78**	**89.58**	**97.99**	99.00

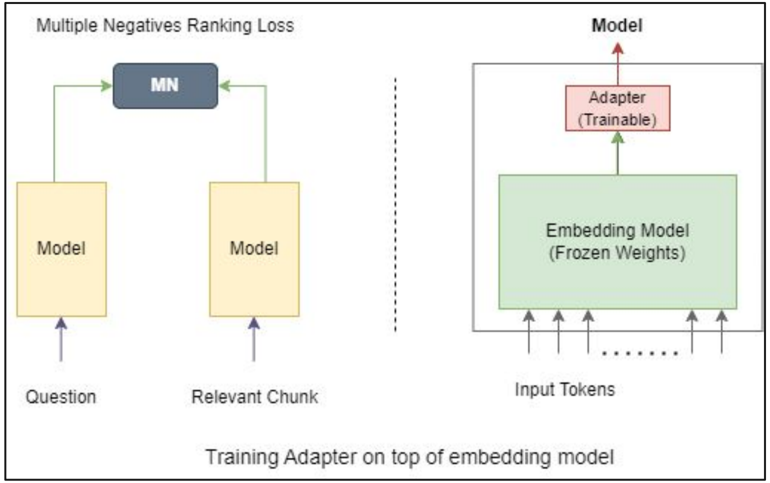
Table 1: Performance of pre-trained embedding models on MedQuAD

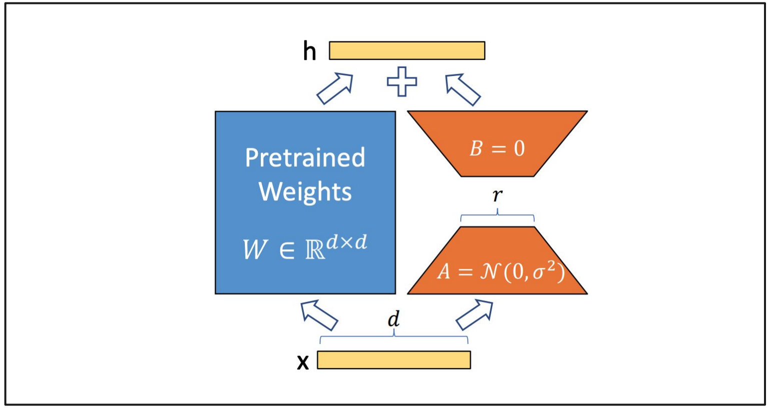
- Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, 11 2019.
- Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation, 2024

Fine-tuning strategies

- Used paraphrase-mpnet-base for fine-tuning
- Adapter fine-tuning
  - Freeze the model weights
  - Train a light-weight adapter on top
- LoRA
  - Freeze the model weights
  - Add matrices of low rank

Split: **20k** (query, chunks) for training and **5k** for testing.





LoRA Diagram Source: https://heidloff.net/article/efficient-fine-tuning-lora/

Adapter Fine-tuning

- Two Layer NN on top
- Only positive pairs
  - *(query, one relevant chunk)*
- Fine-tuned for **8** epochs
- Loss function: MultipleNegativesRankingLoss
- **0.92 GB** of GPU memory
- **10m 38s** per epoch
- Small improvements in performance

Method	Training Set					
	MRR@1	**MRR@3**	**MRR@10**	**R@1**	**R@3**	**R@10**
Two Layer Adapter, 8 epochs	79.50	85.12	85.90	19.67	38.17	54.03
Two Layer Adapter, 32 epochs	81.56	86.87	87.53	20.22	39.25	**55.05**
Three Layer Adapter	82.03	87.18	87.79	20.32	39.23	54.85
LoRA	1.82	2.65	3.39	0.46	0.96	2.11
LoRA + Contrastive	69.77	75.67	76.71	17.31	27.69	34.88
LoRA + Contrastive + CoSENT	**88.58**	**91.54**	**91.83**	**22.06**	**41.72**	54.06
	Test Set					
Two Layer Adapter, 8 epochs	78.04	84.34	85.31	20.18	36.86	50.54
Two Layer Adapter, 32 epochs	80.74	86.52	87.31	20.89	37.90	**51.32**
Three Layer Adapter	81.36	86.99	87.68	21.04	38.08	51.22
LoRA	1.74	2.67	3.38	0.47	1.04	2.27
LoRA + Contrastive	60.44	65.11	66.57	15.98	22.70	27.69
LoRA + Contrastive + CoSENT	**85.32**	**88.89**	**89.64**	**22.07**	**39.31**	49.02

Table 2: Results for different fine-tuning strategies

Adapter fine-tuning documentation: https://docs.llamaindex.ai/en/stable/examples/finetuning/embeddings/finetune_embedding_adapter/

Adapter Fine-tuning

- To check generalization, fine-tuned for **32** epochs
- Increase in performance

Method	Training Set					
	MRR@1	**MRR@3**	**MRR@10**	**R@1**	**R@3**	**R@10**
Two Layer Adapter, 8 epochs	79.50	85.12	85.90	19.67	38.17	54.03
Two Layer Adapter, 32 epochs	81.56	86.87	87.53	20.22	39.25	**55.05**
Three Layer Adapter	82.03	87.18	87.79	20.32	39.23	54.85
LoRA	1.82	2.65	3.39	0.46	0.96	2.11
LoRA + Contrastive	69.77	75.67	76.71	17.31	27.69	34.88
LoRA + Contrastive + CoSENT	**88.58**	**91.54**	**91.83**	**22.06**	**41.72**	54.06
	Test Set					
Two Layer Adapter, 8 epochs	78.04	84.34	85.31	20.18	36.86	50.54
Two Layer Adapter, 32 epochs	80.74	86.52	87.31	20.89	37.90	**51.32**
Three Layer Adapter	81.36	86.99	87.68	21.04	38.08	51.22
LoRA	1.74	2.67	3.38	0.47	1.04	2.27
LoRA + Contrastive	60.44	65.11	66.57	15.98	22.70	27.69
LoRA + Contrastive + CoSENT	**85.32**	**88.89**	**89.64**	**22.07**	**39.31**	49.02

Table 2: Results for different fine-tuning strategies

Adapter Fine-tuning

- Fine-tuned a three layer NN to check for further generalization
- 32 epochs
- **1.04 GB** of GPU memory
- **12m 13s** per epoch
- Performance further increased

Method	MRR@1	MRR@3	MRR@10	R@1	R@3	R@10
	Training Set					
Two Layer Adapter, 8 epochs	79.50	85.12	85.90	19.67	38.17	54.03
Two Layer Adapter, 32 epochs	81.56	86.87	87.53	20.22	39.25	**55.05**
Three Layer Adapter	82.03	87.18	87.79	20.32	39.23	54.85
LoRA	1.82	2.65	3.39	0.46	0.96	2.11
LoRA + Contrastive	69.77	75.67	76.71	17.31	27.69	34.88
LoRA + Contrastive + CoSENT	**88.58**	**91.54**	**91.83**	**22.06**	**41.72**	54.06
	Test Set					
Two Layer Adapter, 8 epochs	78.04	84.34	85.31	20.18	36.86	50.54
Two Layer Adapter, 32 epochs	80.74	86.52	87.31	20.89	37.90	**51.32**
Three Layer Adapter	81.36	86.99	87.68	21.04	38.08	51.22
LoRA	1.74	2.67	3.38	0.47	1.04	2.27
LoRA + Contrastive	60.44	65.11	66.57	15.98	22.70	27.69
LoRA + Contrastive + CoSENT	**85.32**	**88.89**	**89.64**	**22.07**	**39.31**	49.02

Table 2: Results for different fine-tuning strategies

LoRA

- **12k** positive (*query*, *chunk*) pairs
- CosineSimilarity Loss
- Rank of LoRA: $r = 8$
- For batch size of 8:
  - **10 GB** GPU memory
  - **14 minutes** per epoch
- **Catastrophic Forgetting**
- Possible reason
  - Only positive samples
  - Inadequate loss function

Method	Training Set					
	MRR@1	MRR@3	MRR@10	R@1	R@3	R@10
Two Layer Adapter, 8 epochs	79.50	85.12	85.90	19.67	38.17	54.03
Two Layer Adapter, 32 epochs	81.56	86.87	87.53	20.22	39.25	**55.05**
Three Layer Adapter	82.03	87.18	87.79	20.32	39.23	54.85
LoRA	1.82	2.65	3.39	0.46	0.96	2.11
LoRA + Contrastive	69.77	75.67	76.71	17.31	27.69	34.88
LoRA + Contrastive + CoSENT	**88.58**	**91.54**	**91.83**	**22.06**	**41.72**	54.06
	Test Set					
Two Layer Adapter, 8 epochs	78.04	84.34	85.31	20.18	36.86	50.54
Two Layer Adapter, 32 epochs	80.74	86.52	87.31	20.89	37.90	**51.32**
Three Layer Adapter	81.36	86.99	87.68	21.04	38.08	51.22
LoRA	1.74	2.67	3.38	0.47	1.04	2.27
LoRA + Contrastive	60.44	65.11	66.57	15.98	22.70	27.69
LoRA + Contrastive + CoSENT	**85.32**	**88.89**	**89.64**	**22.07**	**39.31**	49.02

Table 2: Results for different fine-tuning strategies

LoRA + Contrastive learning

- **12k** positive (*query, chunk*) pairs
- **12k** *hard* negative pairs
  - Pick the top irrelevant chunk among the retrieved chunks
- Solved the catastrophic forgetting problem
- No performance gains
  - Implies inadequate loss function

Method	Training Set					
	MRR@1	**MRR@3**	**MRR@10**	**R@1**	**R@3**	**R@10**
Two Layer Adapter, 8 epochs	79.50	85.12	85.90	19.67	38.17	54.03
Two Layer Adapter, 32 epochs	81.56	86.87	87.53	20.22	39.25	**55.05**
Three Layer Adapter	82.03	87.18	87.79	20.32	39.23	54.85
LoRA	1.82	2.65	3.39	0.46	0.96	2.11
LoRA + Contrastive	69.77	75.67	76.71	17.31	27.69	34.88
LoRA + Contrastive + CoSENT	**88.58**	**91.54**	**91.83**	**22.06**	**41.72**	54.06
	Test Set					
Two Layer Adapter, 8 epochs	78.04	84.34	85.31	20.18	36.86	50.54
Two Layer Adapter, 32 epochs	80.74	86.52	87.31	20.89	37.90	**51.32**
Three Layer Adapter	81.36	86.99	87.68	21.04	38.08	51.22
LoRA	1.74	2.67	3.38	0.47	1.04	2.27
LoRA + Contrastive	60.44	65.11	66.57	15.98	22.70	27.69
LoRA + Contrastive + CoSENT	**85.32**	**88.89**	**89.64**	**22.07**	**39.31**	49.02

Table 2: Results for different fine-tuning strategies

Jun Lu, David Li, Bill Ding, and Yu Kang. Improving embedding with contrastive fine-tuning on small datasets with expert-augmented scores, 2024

LoRA + Contrastive learning + CoSENT Loss

- **12k** positive + **12k** *hard* negative pairs
- CoSENT Loss

$$\text{loss} = \log(1 + \exp(s(k,l)) - \exp(s(i,j)) + \ldots)$$

- For all input pairs in a batch where *s(k, l)* is more than *s(i, j)*
- Great boost in performance for both training as well as test sets

Method	MRR@1	MRR@3	MRR@10	R@1	R@3	R@10
Training Set						
Two Layer Adapter, 8 epochs	79.50	85.12	85.90	19.67	38.17	54.03
Two Layer Adapter, 32 epochs	81.56	86.87	87.53	20.22	39.25	**55.05**
Three Layer Adapter	82.03	87.18	87.79	20.32	39.23	54.85
LoRA	1.82	2.65	3.39	0.46	0.96	2.11
LoRA + Contrastive	69.77	75.67	76.71	17.31	27.69	34.88
LoRA + Contrastive + CoSENT	**88.58**	**91.54**	**91.83**	**22.06**	**41.72**	54.06
Test Set						
Two Layer Adapter, 8 epochs	78.04	84.34	85.31	20.18	36.86	50.54
Two Layer Adapter, 32 epochs	80.74	86.52	87.31	20.89	37.90	**51.32**
Three Layer Adapter	81.36	86.99	87.68	21.04	38.08	51.22
LoRA	1.74	2.67	3.38	0.47	1.04	2.27
LoRA + Contrastive	60.44	65.11	66.57	15.98	22.70	27.69
LoRA + Contrastive + CoSENT	**85.32**	**88.89**	**89.64**	**22.07**	**39.31**	49.02

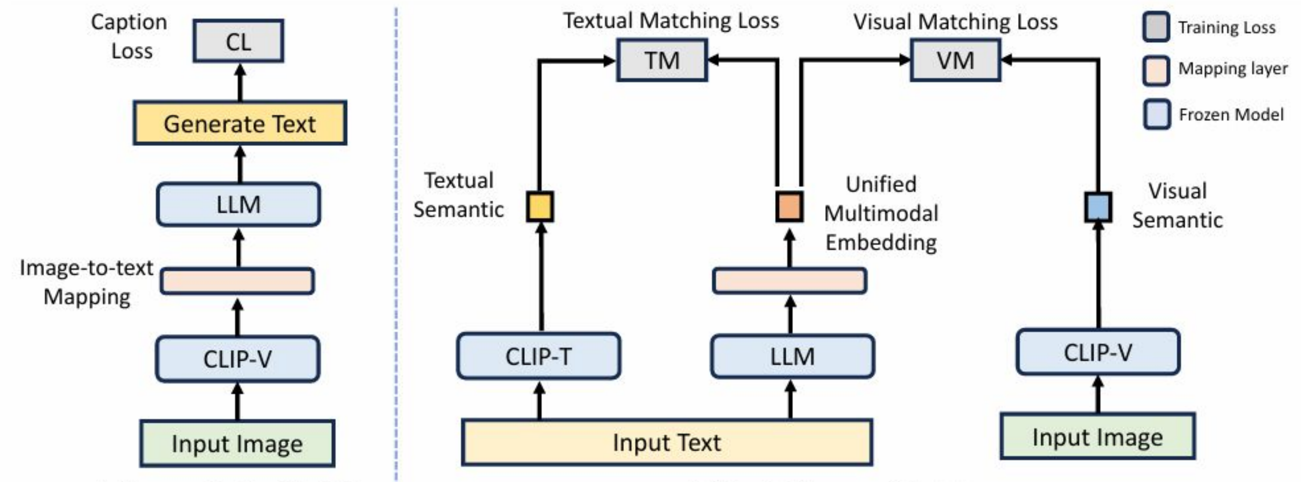
Table 2: Results for different fine-tuning strategies

Multimodal RAG Implementation

Sources :-
https://arxiv.org/pdf/2301.13823 (fromage)
https://aclanthology.org/2024.findings-eacl.105/ (uniMUR)

Bridging Gap between pre-trained(LM and vis_enc)

Unified Embeddings for Multimodal Retrieval



(a) Image-to-text Training

(b) Dual Alignment Training

Losses

- Cross Entropy for Image captionary (next token prediction task)
- MSE error loss between for Textual Matching between unified embedding and text_features of caption_text .
- Contrastive loss function for visual Matching between over a batch N image and unified embeddings . we maximise similarity between relevant pairs and minimize similarity between irrelevant pairs.

$$\mathcal{L} = \mathcal{L}_{cap} + \lambda_1 \mathcal{L}_{vm} + \lambda_2 \mathcal{L}_{tm},$$

Implementation

- To implement uniMUR from scratch based on whatever details given in the uniMUR - not easy . It lacks several Implementation details like techniques involved in training , given compute cost is for what batch_size , epochs .

- Coming to the Image captioning task (next token prediction task) . The most natural way here is teacher forcing . In our case , it turns out that with teacher forcing on single image-text example , the training becomes a bit easy and Testing of the model becomes a bit harsh (sequential)

More effective Teacher Forcing

Random concatenation of tuples (language like image token , caption) makes training a bit more harsh. Model learns to attend whom to score better . It improves image captioning task (basically CIDEr score and more meaningful captions

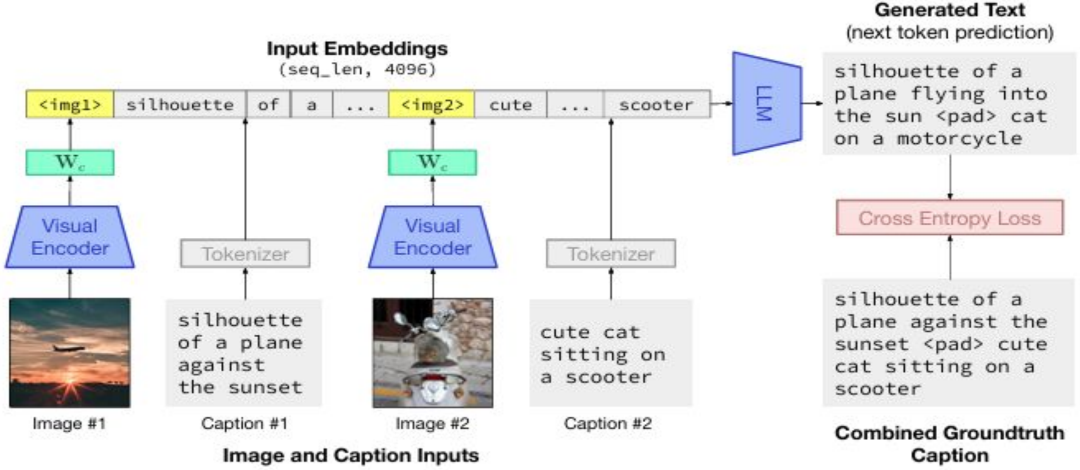


Image Captioning

FROMAGe - code open source

- Building uniMUR from scratch without proper implementation details was hard . we tried some approaches , but ran into issues several issues .
- Now building uniMUR from FROMAGe - from its code base
- Most Implementation details we get to know from FROMAGe paper and code

Training Intensive task - for us

The details mentioned in the FROMAGe paper are really training intensive :-

- Batch_size = 180 (needs good amount of memory)
- Training samples = 3M
- OPT- 6.7B parameters
- Training for 24 hours on 1 A6000 GPU

Implementation details

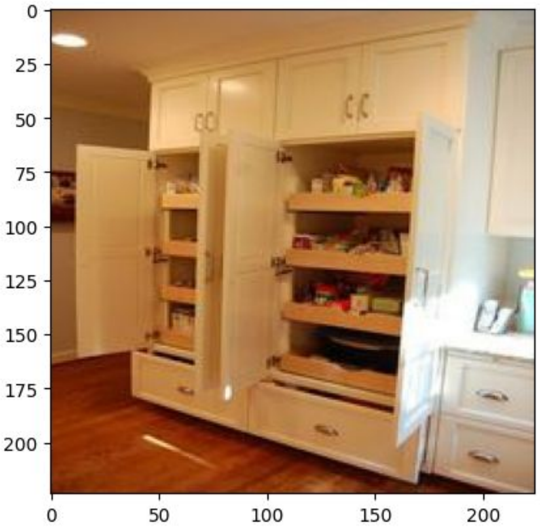
- Batch_size = 16 (needs good amount of memory , hard to increase)
- Training samples = 20k (also memory issue)
- OPT- 125 million parameters
- Training on ~ 30 GB - GPU v100 (memory problem)

Results

(i) somewhat meaning full results on image captioning task
(ii) recall image-text from image is very bad.
(iii) recall image -text from text is very bad.
(iv) complete implementation

Captioning results

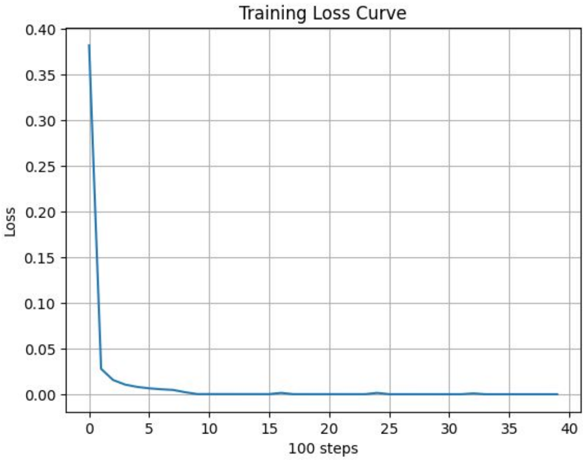
['happy newlyweds striking a romantic pose out on the red carpet at the entrance']
['a photo of a wedding dress is a wedding cake for a bride and groom [RET] </s>']



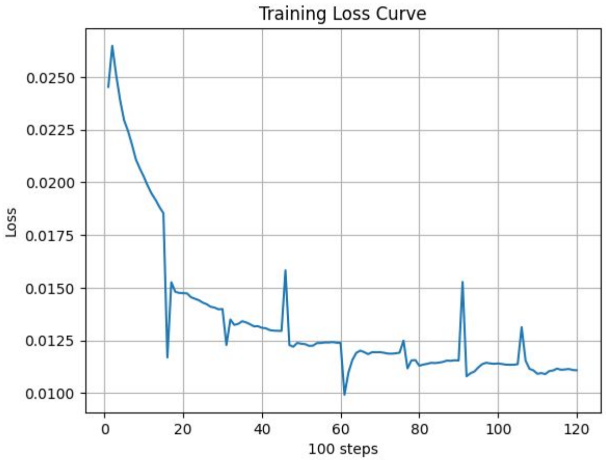
['pull out shelves in kitchen cabinets ... this would be great in a craft room !']
['a photo of the kitchen countertops are set up for a large island . [RET] </s>']

Appendix

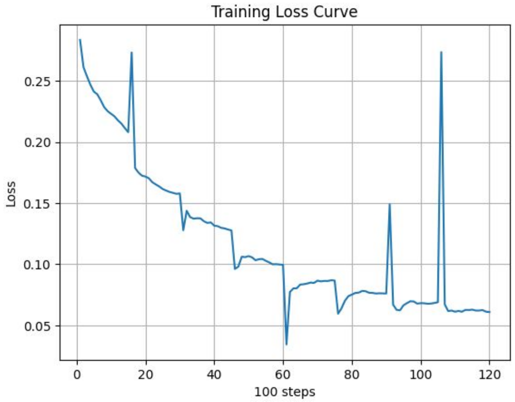
Loss Curves



LoRA fine-tuning



LoRA + Contrastive Learning



LoRA + Contrastive + CoSENT Loss