GenAI Final Project Report

Kristin Henderson

Part 1: Pre-training Embedding Models

Training Configuration

I trained two embedding models from scratch using a two-stage approach (MLM pre-training followed by contrastive fine-tuning). Both models use DistilBERT as the base architecture (approximately 66 million parameters), but they differ in training data and epochs.

General-Purpose Wikipedia Model

- Learning rate: 5e-5
- Optimizer: AdamW (weight decay 0.01 for MLM, 0.0 for contrastive)
- LR schedule: Linear warmup (10%) + cosine decay (MLM), linear warmup only (contrastive)
- Batch size: 32
- Training epochs: 20 MLM epochs (11 with early stopping, 2000 articles from Wikitext-103), 5 contrastive epochs
- Contrastive temperature: 0.1
- Positive samples: Neighboring sentences
- Negative samples: Hard negatives (sentences offset by 12 sentences) + in-batch negatives
- Max sequence length: 256
- Pooling: Mean pooling with L2 normalization
- Projection: 256-dimensional projection layer

Paul Graham-Specific Model

- Learning rate: 5e-5
- Optimizer: AdamW (weight decay 0.01 for MLM, 0.0 for contrastive)
- LR schedule: Linear warmup (10%) + cosine decay (MLM), linear warmup only (contrastive)
- Batch size: 32
- Training epochs: 20 MLM epochs, 8 contrastive epochs
- Contrastive temperature: 0.1
- Positive samples: Neighboring sentences
- Negative samples: Hard negatives (sentences offset by 12 sentences) + in-batch negatives
- Max sequence length: 256
- Pooling: Mean pooling with L2 normalization
- Projection: 256-dimensional projection layer

I stuck with the common BERT-style defaults because they trained fine on my runs: AdamW as the optimizer, a 5e-5 learning rate for both MLM pre-training and contrastive fine-tuning (common when training from scratch), batch size 32 to fit on a Colab GPU and provide a reasonable number of in-batch negatives, and temperature 0.1 to help the model learn clear distinctions between positive and negative pairs. The Wiki MLM stopped at 11 epochs because early stopping kicked in on the 2,000 Wikitext-103 articles, while the PG MLM kept improving and used the full 20 epochs on the essay corpus. I turned off weight decay during contrastive so the L2 penalty wouldn't keep shrinking weights; that way the adapter could move to whatever values the contrastive loss needed instead of being pulled back toward zero.

Architecture Comparison and Design Decisions

Both models use a two-stage training pipeline: first MLM pre-training from scratch, then contrastive fine-tuning with hard negatives. I chose this approach because it allows the model to first learn general language representations through MLM, then specialize those representations for semantic similarity tasks through contrastive learning.

Two-Stage Training Approach

The first stage (MLM pre-training) trains the model to predict bi-directional masked tokens. This helps the model develop a general understanding of language, which I thought was important when training from scratch, as it provides a strong initialization before contrastive learning. The second stage (contrastive fine-tuning) uses an InfoNCE-style loss with hard negatives, where each anchor sentence is paired with a positive (neighboring sentence) and a hard negative (sentence 12 positions away). This teaches the model to distinguish between semantically similar and dissimilar text pairs.

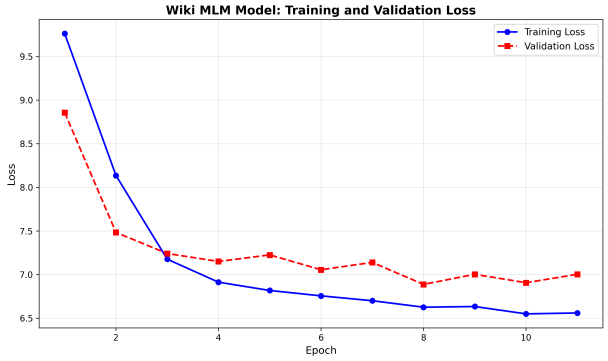
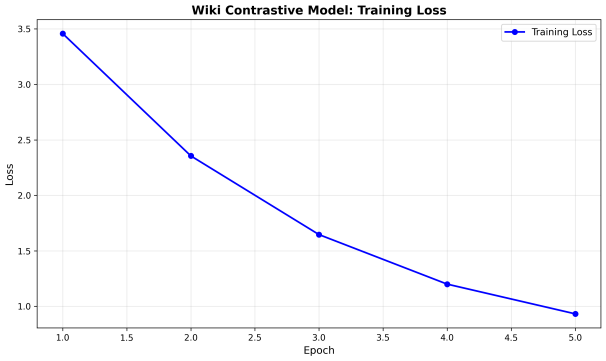
I chose DistilBERT as the base architecture because it balances size (66M params) and speed while keeping most of BERT's capability. For embedding extraction, I use mean pooling over the token embeddings, which averages all token representations while masking out padding tokens. This creates sentence-level embeddings for variable-length sequences. After mean pooling, I apply a 256-dimensional projection layer, which projects the 768-dimensional embeddings to a lower-dimensional space so the model focuses on the most useful features for contrastive learning. The final embeddings are L2-normalized to unit length, so distances are consistent when comparing embeddings.

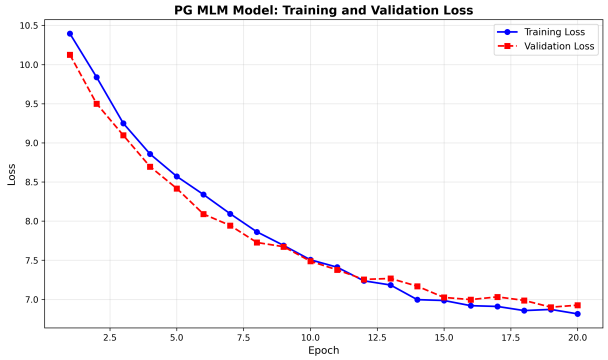
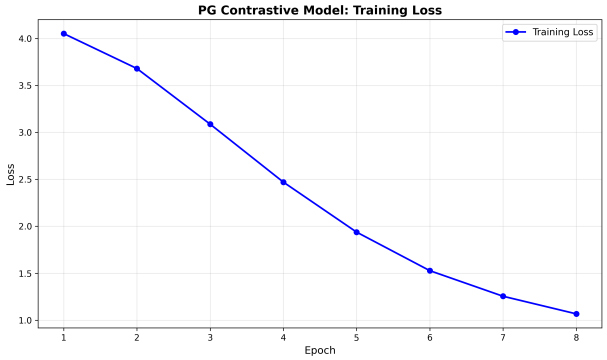
Hard Negative Mining

The contrastive training uses hard negatives rather than random negatives. Hard negatives are sentences that are close (12 sentences away), within the same essay, and assumed not to be semantically related to the anchor. This is more challenging than random negatives and helps the model learn finer-grained distinctions, so it can tell apart chunks from similarly themed essays. The model also uses in-batch negatives, where other positive pairs in the same batch serve as additional negatives, increasing the number of negative examples per batch.

Loss Plots

The training and validation loss curves for both models are shown below:

Wiki MLM Loss	Wiki Contrastive Loss
	

PG MLM Loss	PG Contrastive Loss
	

Embedding-Specific Metrics

Contrastive Loss

Both models learned during contrastive fine-tuning:

- The Wikipedia model's training loss decreased from approximately 3.46 to 0.93 over 5 epochs, showing steady improvement in distinguishing positive and negative pairs. The loss decreased smoothly across epochs.
- The PG model's training loss decreased from approximately 4.05 to 1.07 over 8 epochs, showing similar learning patterns.

The contrastive loss measures how well the model distinguishes between positive pairs (anchor and neighboring sentence) and negative pairs (anchor and hard negative). Lower loss indicates better separation in the embedding space, which should translate to better retrieval performance.

Retrieval Metrics

To evaluate the models on a practical retrieval task, I tested both on a question-answering retrieval task where queries are matched against chunks from Paul Graham essays, using the questions from the validation split of my combined QnA dataset. The models were evaluated using Mean Reciprocal Rank (MRR) and Normalized Discounted Cumulative Gain at rank 10 (nDCG@10). These metrics were computed during the RAG system evaluation in Part 3:

Original contrastive models (after Part 1 training):

- **Wikipedia encoder**:
  - MRR = 1.0000
  - nDCG@10 = 0.0321
- **PG encoder**:
  - MRR = 0.8978
  - nDCG@10 = 0.2515

After post-training on question-chunk pairs (Part 3 fine-tuning):

- **Wikipedia encoder**:
  - MRR = 0.9520
  - nDCG@10 = 0.2843
- **PG encoder**:
  - MRR = 0.9009
  - nDCG@10 = 0.4124

All retrieval metrics above use reranking: top-20 chunks are first retrieved with the embedding model, then reranked with a cross-encoder to select the final top-k.

MRR measures how high the first relevant chunk ranks; 1.0 means it's always ranked first. Values around 0.9 mean the right chunk is usually within the top 1–2, and dropping toward 0.5 means it often falls to the top 2-3. However, MRR only considers questions where a relevant chunk was successfully retrieved, so a high MRR with a low retrieval success rate indicates the model performs well when it finds relevant chunks, but fails to find them for most questions. nDCG@10 evaluates the quality of the top-10 ranked results, giving higher weight to documents ranked near the top. Higher nDCG@10 values indicate that relevant documents are consistently placed near the top of the ranking.

Among the two contrastive models trained in Part 1, the PG model shows better retrieval performance, with MRR of 0.8978 and nDCG@10 of 0.2515. The Wikipedia model scored a perfect MRR on the cases where it found something, but its nDCG@10 was lower, meaning it surfaced fewer or lower-ranked relevant chunks across the top of the list. This suggests that domain-specific training on Paul Graham essays helps the model better match queries to relevant essay chunks, even though the Wikipedia model has broader general knowledge.

Comparison Analysis

Both models trained successfully, but they specialize in different ways.

Based on the results, the Wikipedia model likely learned broad semantic patterns from diverse topics. That general knowledge should help it match queries to documents even when they use different vocabulary or phrasing.

The PG model should be better at understanding relationships between sentences within essays and recognizing the author's phrasing and style, because it likely picked up domain-specific patterns from Paul Graham's essays.

For RAG applications, the choice between models depends on the specific retrieval needs. If the task involves finding relevant chunks within Paul Graham essays based on semantic similarity, the PG model may provide better results. If the task involves matching user queries (which may use different vocabulary) to essay content, or if queries are very short (like titles), the Wikipedia model's general semantic understanding may be more helpful.

Architecture Choices / Limitations / Future Work

I used DistilBERT for both models, though I originally intended to use BERT-base for the Wikipedia model to take advantage of its larger capacity. In future work, I would use BERT-base for the Wikipedia model instead of DistilBERT to better capture general semantic patterns.

Part 2: Post-Training a Language Model for Paul Graham QnA

Dataset Construction

For this part, I built two types of QnA datasets based on Paul Graham's essays.

Base QnA set (manual or curated).

I created a small set of 50 QnA pairs by writing them myself using a Gradio interface. These questions focus on the main ideas in the essays and have short, factual answers (1-2 sentences) taken from a single chunk. This gives the model a small but clean set of examples to learn from. I used odd-indexed chunks from the essays to ensure no overlap with the synthetic dataset.

Synthetic QnA set (LLM-generated).

I generated a larger synthetic dataset of 1,308 QnA pairs using Gemma-3-4b-it. I prompted the model to write short questions and answers for individual essay chunks, focusing on main ideas and key concepts. This broadened topic coverage, but the LLM-written questions and answers vary more in phrasing and tone than the hand-written set. I used even-indexed chunks to keep a clean split from the base dataset.

I saved each dataset as JSON and created separate 80/20 train-validation splits for the base-only, synthetic-only, and combined setups.

Fine-Tuning Strategy (LoRA)

I used LoRA to fine-tune Gemma-3-1b-it. LoRA is a parameter-efficient method that trains only a small set of low-rank adapter weights while the base model stays frozen. I chose it because it ran reliably in Colab (QLoRA kept giving me issues), was fast, and didn't require storing or updating the full 1B parameters. This approach is commonly used for instruction-tuned models like Gemma-3-1b-it on moderate-sized datasets.

I used typical small-model settings: rank 8, alpha 16, and dropout 0.1. LoRA updated about 1.49 million parameters out of ~1.0 billion (about 0.15%), with everything else frozen, which kept training light. It only modifies the attention projection matrices (query, key, value, output), so the model can learn the QnA task without changing the original weights. That's helpful here because the dataset is small, and I just need to nudge the model toward the essays' style and content instead of updating all its parameters.

To match my RAG prompt, I updated the generation prompt to explicitly request concise 1–2 sentence answers with no bullets. I also filtered synthetic QnA answers to <=80 tokens (model tokenizer), removing 2 of 1,308 items. After this change, the base and synthetic models improved. The oversampled combined model dipped, likely because the 5x base:synthetic weighting is less optimal under the shorter-answer constraint.

Training Comparison

I trained three separate LoRA models:

1. **Base-only model:** trained only on the curated QnA set (50 pairs).
2. **Synthetic-only model:** trained only on the larger synthetic dataset (1,308 pairs).
3. **Combined model:** trained on both datasets together with 5x oversampling of the base pairs (1,558 total pairs: 1,308 synthetic + 250 base).

All models were trained using the same configuration:

- Learning rate: 5e-4
- AdamW optimizer
- Cosine schedule with warmup (100 steps)
- Evaluation on a 20 percent validation split
- Batch size: 8 (per-device) with gradient accumulation of 4
- Mixed precision (fp16) enabled

I used the same settings so the comparison would reflect differences in the training data rather than differences in hyperparameters. The number of epochs varied: 3 epochs for synthetic, 15 epochs for base (due to small dataset size), and 4 epochs for combined. I used AdamW and a cosine schedule with warmup because these are standard choices for small fine-tuning runs.

Validation loss decreased across all three models, with the combined model reaching the lowest loss overall.

Evaluation Metrics

I evaluated the three fine-tuned models on a held-out validation split (20 percent of the data) using four metrics:

- Loss (from training logs)
- BERT-F1 (semantic similarity between predicted and reference answers)
- ROUGE-L (overlap of longest common subsequences)
- Token-level F1 (precision/recall over tokens)

Here are the main results:

- **Base QnA only**
  - BERT-F1: 0.6471
  - ROUGE-L: 0.2154
  - Token F1: 0.2723
  - Validation Loss: 2.9381
- **Synthetic QnA only**
  - BERT-F1: 0.7015
  - ROUGE-L: 0.2141
  - Token F1: 0.2598
  - Validation Loss: 2.9867
- **Combined base + synthetic**
  - BERT-F1: 0.5991
  - ROUGE-L: 0.1768
  - Token F1: 0.2280
  - Validation Loss: 2.8220

BERT-F1 measures how semantically similar the model's answer is to the reference answer, even if the wording is different. ROUGE-L and token-level F1 focus more on exact or near-exact word overlap. Higher values are better for all three metrics.

After updating the generation prompt to match the RAG prompt format (requesting concise 1–2 sentence answers with no bullets) and filtering synthetic answers to <=80 tokens, the synthetic-only model now achieves the highest BERT-F1, indicating the strongest semantic alignment with reference answers. The base-only model leads ROUGE-L and Token F1, likely because its curated answers are short and closely match the references. The combined model still has the lowest loss but no longer leads the lexical metrics after these changes.

Best Model Selection

All three models learned the QnA task, but they behave differently.

- The base-only model benefits from clean, hand-curated pairs and now leads ROUGE-L and Token F1 (0.2154 / 0.2723). It required 15 epochs to converge, reflecting the small dataset size (50 pairs).
- The synthetic-only model achieves the highest BERT-F1 (0.7015) after the concise prompt and 80-token filter. The larger synthetic set (1,306 usable pairs after filtering) helps cover more essay topics and styles. Its validation loss is highest (2.9867), but semantically it is strongest.
- The combined model has the lowest validation loss (2.8220) and balances both datasets, but after the prompt and length updates, its lexical metrics fall below the single-dataset models.

Given these results, I would pick synthetic-only for semantic alignment, base-only for lexical closeness, and combined as a middle-ground with the best loss.

Limitations / Future Work

My final fine-tuned language model (as well as my reranker from Part 3) uses the training split of the combined QnA dataset. However, given the updated results after modifying the prompt, I would switch to the synthetic-only dataset and retrain both models in future work.

Part 3: RAG System Benchmark

RAG Pipeline Implementation

I built a complete RAG system that combines semantic retrieval with language model generation. The system uses my pre-trained embedding models (from Part 1) for retrieval and my post-trained language model (from Part 2) for generation.

Chunking Strategy

I chunked Paul Graham essays using sentence-aware chunking with a target size of 256 tokens and 50 tokens of overlap between chunks. This approach preserves sentence boundaries, which helps maintain semantic coherence within chunks. The overlap ensures that context near chunk boundaries is not lost. I used a BERT-base-uncased tokenizer for chunking to ensure consistent tokenization with the embedding models.

Retrieval Component

The retrieval system uses cosine similarity search over chunk embeddings. I built separate indices for each embedding model (Wiki contrastive, PG contrastive, and their post-trained variants). The system first retrieves the top-20 candidate chunks using the embedding model, then applies a reranker to reorder and select the final top-k chunks. The reranker is a cross-encoder that processes each query-chunk pair together to score relevance, giving more precise ranking than embedding similarity alone. I use k=5 for generation, but evaluated retrieval performance at k=1, 3, 5, and 10. I used FAISS for efficient similarity search when available, falling back to numpy-based search otherwise.

Generation Component

The generation component uses my fine-tuned Gemma-3-1b-it model (combined dataset from Part 2) with LoRA adapters. For each question, the system combines the top-k retrieved chunks into a single context string and generates an answer. The prompt includes the question and the retrieved context, instructing the model to answer concisely in 1-2 sentences without bullet points or numbered lists. I set max_new_tokens to 80 to enforce shorter, more direct answers that match the reference style.

Embedding Model Fine-Tuning Strategy

To improve retrieval performance for question-answering, I post-trained both embedding models (Wiki and PG) on question-chunk pairs from the combined training set (synthetic + 5x oversampled base QnA pairs). This fine-tuning step teaches the models to maximize similarity between questions and their relevant chunks, which is more aligned with the RAG retrieval task than the general contrastive training from Part 1.

I fine-tuned both models starting from their contrastive checkpoints, preserving the 256-dimensional projection layer (from Part 1) so embeddings stay in the same space. I used a symmetric contrastive loss (InfoNCE-style) with in-batch negatives, pairing each question with its relevant chunk as positive and other batch chunks as negatives. The training configuration used:

- Learning rate: 3e-5
- Batch size: 32
- Training epochs: 5
- Temperature: 0.05
- Optimizer: AdamW with weight decay 0.01
- LR schedule: Linear decay

Reranker Fine-Tuning Strategy

To further improve retrieval precision, I fine-tuned a cross-encoder reranker model. The reranker takes query-chunk pairs as input and outputs a relevance score, allowing for more precise ranking than embedding-based similarity alone. I fine-tuned it on the same combined training set (synthetic + 5x oversampled base QnA pairs) used for embedding fine-tuning.

The reranker training used positive pairs (question, relevant chunk) and negative pairs (question, irrelevant chunk) sampled from the essay corpus. The training configuration used:

- Base model: cross-encoder/ms-marco-MiniLM-L-6-v2
- Learning rate: 2e-5
- Batch size: 16
- Training epochs: 3
- Optimizer: AdamW with weight decay 0.01

- LR schedule: Linear decay with warmup (10%)
- Problem type: Regression (outputs a single relevance score)

The reranker is integrated into the RAG pipeline by first retrieving top-20 candidates using the embedding model, then reranking them to select the final top-k chunks for generation.

Embedding Model Comparison

I compared retrieval performance between my general-purpose (Wikipedia) and Paul Graham-specific embedding models, both in their original contrastive-trained form and after post-training on question-chunk pairs.

Retrieval Metrics

I evaluated retrieval on the combined validation set (synthetic + 5x oversampled base) for all models to ensure fair comparison and avoid data leakage, since the LLM was trained on the training split of this dataset. I determined ground-truth relevant chunks by matching each QnA pair's 'context' field to the corresponding chunk in the chunked corpus. All models were evaluated using Precision@k, Recall@k, MRR, and nDCG@10.

Results:

Model	P@3	R@3	P@5	R@5	MRR	nDCG@10
Wiki contrastive	0.0107	0.0321	0.0064	0.0321	1.0000	0.0321
Wiki post-trained	0.0972	0.2917	0.0590	0.2949	0.9520	0.2843
PG contrastive	0.0855	0.2564	0.0545	0.2724	0.8978	0.2515
PG post-trained	0.1410	0.4231	0.0885	0.4423	0.9009	0.4124

Note: P@k = Precision@k, R@k = Recall@k. All models evaluated on combined validation set.

Independent Test Set Retrieval Metrics

I also evaluated retrieval performance on an independent test set of 55 questions from `examples/qa_pairs.json`, which comes from the Tonic Validate GitHub repository (the same ground-truth set used for Tonic Validate evaluation). This provides an additional perspective on model performance on held-out data that was not used during training or validation.

Results:

Model	P@3	R@3	P@5	R@5	MRR	nDCG@10
Wiki contrastive	0.0242	0.0545	0.0182	0.0636	1.0000	0.0635
Wiki post-trained	0.0848	0.1773	0.0509	0.1773	0.8611	0.1661
PG contrastive	0.1455	0.3061	0.0909	0.3152	0.8730	0.2969
PG post-trained	0.1212	0.2455	0.0764	0.2545	0.8444	0.2445

Note: P@k = Precision@k, R@k = Recall@k. All models evaluated on independent test set (55 questions from examples/qa_pairs.json, from Tonic Validate GitHub).

Model Comparison

Precision@k measures the fraction of retrieved chunks that are relevant; Recall@k measures the fraction of relevant chunks that are retrieved. MRR measures the average rank of the first relevant chunk (higher is better), but only considers questions where a relevant chunk was retrieved. nDCG@10 evaluates the top-10 ranking quality, giving higher weight to relevant documents ranked near the top.

As shown in the table above, the Wiki contrastive model shows perfect MRR but very low precision and recall, indicating that while it ranks correctly when it finds relevant chunks, it struggles to retrieve relevant chunks for most questions. The PG contrastive model performs better, demonstrating that domain-specific pre-training helps with retrieval.

Post-training on question-chunk pairs provides substantial improvements for both models. The Wiki post-trained model shows large gains across all metrics compared to the original contrastive model, while the PG post-trained model achieves the best overall performance. These results suggest that task-specific fine-tuning on question-chunk pairs is very effective for RAG retrieval, helping both general-purpose and domain-specific models.

One interesting observation: the PG post-trained model performs best on the combined validation set, but the PG contrastive model (before fine-tuning) has higher metrics on the independent test set. This could be due to overfitting, though the small test set size (55 questions) might also play a role.

Generation Quality Evaluation

I evaluated the complete RAG system on 55 benchmark questions from `examples/qa_pairs.json` (the same ground-truth set from Tonic Validate GitHub) using the Tonic Validate framework with GPT-4.1-mini as the evaluator. The evaluation computed five metrics:

- **Answer Similarity** (0.0-5.0): How well the reference answer matches the LLM answer
- **Retrieval Precision** (0.0-1.0): Whether the context retrieved is relevant to answer the given question
- **Augmentation Precision** (0.0-1.0): Whether the relevant context is in the LLM answer
- **Augmentation Accuracy** (0.0-1.0): Whether all the context is in the LLM answer
- **Answer Consistency** (0.0-1.0): Whether there is information in the LLM answer that does not come from the context

Results:

- Answer Similarity: 2.3818
- Retrieval Precision: 0.3964
- Augmentation Precision: 0.5173
- Augmentation Accuracy: 0.5127
- Answer Consistency: 0.7288

Answer Consistency (0.73) is the strongest metric, indicating that most information in the generated answers comes from the retrieved context. Answer Similarity (2.38 out of 5.0) is moderate, which suggests that generated answers differ from reference answers in style, content, or level of detail. The system tends to generate longer, more detailed answers compared to the concise reference answers (generated answers average 325 characters vs. 148 characters for reference answers), which may contribute to the stylistic differences reflected in similarity scores.

Generation Metrics Comparison

I compared the RAG system's generation performance when using different embedding models for retrieval (Wiki contrastive, Wiki post-trained, PG contrastive, PG post-trained) on the combined validation set. All systems use the same language model (combined dataset from Part 2) to ensure fair comparison, so differences in generation quality reflect how well each embedding model retrieves relevant chunks for the language model to use.

Results:

Metric	Wiki (Orig)	Wiki (Post)	PG (Orig)	PG (Post)
BERT-F1	0.5936	0.6259	0.6233	0.6334
ROUGE-L	0.1780	0.1868	0.1810	0.1948
Token F1	0.2151	0.2256	0.2237	0.2363

Note: Generation metrics after retrieval using each of the 4 embedding models, evaluated on combined validation set.

The RAG system using the PG post-trained embedding model achieves the best generation performance across all three metrics (BERT-F1: 0.6334, ROUGE-L: 0.1948, Token F1: 0.2363), followed closely by the Wiki post-trained model and the PG original model.

Example Outputs

To illustrate the system's performance, below are two examples from the evaluation on the ground-truth set from the Tonic Validate GitHub:

Example 1: High Performance

Question: What often happens to startups that turn down acquisition offers?

Reference Answer: Startups that turn down acquisition offers usually end up doing better, often receiving a bigger offer later or going for an IPO.

Generated Answer: What often happens to startups that turn down acquisition offers is that they end up doing better.

Metrics: Answer Consistency = 1.0, Augmentation Accuracy = 1.0, Answer Similarity = 4.0

This example demonstrates strong performance across all metrics. The generated answer accurately reflects the reference answer, is well-supported by the retrieved context, and matches the reference in both content and style. The answer is concise, which aligns with the reference answer's style.

Example 2: Median Performance

Question: What is the main reason for the increasing number of startups according to Paul Graham?

Reference Answer: The main reason for the increasing number of startups is that it has become cheaper to start web startups.

Generated Answer: According to Paul Graham, the main reason for the increasing number of startups is the increasing number of startup people around you.

Metrics: Answer Consistency = 1.0, Augmentation Accuracy = 0.4, Answer Similarity = 2.0

This example shows a common pattern: the system generates an answer that is consistent with the retrieved context, but it doesn't match the reference answer. The generated answer mentions "increasing number of startup people around you," which is related but not the same as the reference answer about cost. This explains why Augmentation Accuracy (0.4) and Answer Similarity (2.0) are lower.

Analysis

Which embedding model performed better for retrieval and why

Based on validation set performance, the PG post-trained model showed the best retrieval performance, followed by the Wiki post-trained model. The PG model's superior precision and recall suggest that combining domain-specific pre-training with task-specific fine-tuning is effective for retrieval. The Wiki model showed dramatic improvement over its original contrastive version, indicating that question-chunk fine-tuning effectively adapts general-purpose models to the retrieval task.

How the system compares to frontier models

The system shows promise in Answer Consistency (0.73), indicating that when relevant context is retrieved, the generation model can produce answers consistent with that context. However, Answer Similarity (2.38) is below benchmark systems. OpenAI's RAG system averages 3.47 on the same metric (based on the Tonic Validate examples). The moderate Answer Similarity indicates a style mismatch, with the system generating longer, more detailed answers compared to concise reference answers. This suggests that both retrieval and generation components could be improved to better match reference answer style and content.

Limitations and potential improvements

The primary limitations are low retrieval performance and generation quality. For retrieval, potential improvements include testing different top-k values for the reranker's initial retrieval stage (currently set to 20) to optimize the balance between recall and computational cost, using essay-level and chunk-level metadata to better match questions to relevant content, experimenting with different chunk sizes, more thoroughly exploring SimCSE for the Wikipedia model (initial experiments could be promising), and implementing hybrid search combining semantic and keyword-based retrieval.

For generation, I have already implemented several improvements, including modified generation prompts to emphasize brevity and simple sentence formatting (no bullets or numbering), added answer length constraints (max_new_tokens=80), and filtered the fine-tuning dataset to ensure concise answers. However, the model still sometimes produces wordy answers. Future improvements could include more testing of generation parameters (top-k, top-p, temperature) to find optimal settings for concise answers.

The system would also benefit from larger training datasets. Overall, the RAG system performs modestly with room for improvement. It demonstrates that domain-specific embedding training and task-specific fine-tuning can improve retrieval performance.