Bible RAG (Retrieval-Augmented Generation) - Experiments in Bible Question Answering

Zuhair Farhan (27100) [1] **Sahil Kumar (27149)** [1]

Abstract

This project evaluates the effectiveness of Retrieval-Augmented Generation (RAG) systems on the Bible as a corpus, using various document chunking strategies, retrieval techniques, embedding models, and open-access language models. We benchmark different configurations of language models (Qwen2.5 1.5B and 3B), retrieval methods (BM25, MMR, Semantic, and Hybrid RRF), and embedding strategies (including BAAI's `bge-small-en`). Evaluation metrics include cosine similarity-based measures of faithfulness, relevance, and similarity to manually curated ground truth answers. Our findings highlight how chunk size, embedding type, and retrieval diversity significantly influence final answer quality. The complete source code, corpus, and generated results are available at the GitHub Repository.

1. Introduction

RAG is a framework for improving model performance by augmenting prompts with relevant data outside the foundational model, grounding LLM responses on real, trustworthy information. RAG models can easily be adapted to work on any type of corpus, where it is entered into a vector database after being broken into "chunks", then these chunks are retrieved based on a retrieval method, enabling a LLM to answer questions about these documents efficiently.

In this report, we explain the development of the **Bible RAG** model, where we attempted to create an efficient RAG system with the Bible as a corpus, to enable efficient question-answering centered toward the Christian holy text. We outline the several different parameters, LLM models, and embedding types used, and share the results of each, reaching a conclusion as to what worked best for us.

2. Platform and Environment

The project was entirely developed on Google Colab and Kaggle, two online code platforms that grant limited but free access to high-end GPUs, which can be used for running projects with large data requirements. Given that we are working with LLMs and embeddings, it made sense for us to work with these platforms, given both of us authors do not have access to GPUs of our own. However, the total GPU RAM provided did not prove sufficient for our initial tests (will be discussed later), it did prove to be enough for later devised tests.

Additionally, the project was coded in **Python 3**, with the following packages being used:

- `time`, `warnings`
- `numpy`, `pandas`
- `scikit-learn`
- `sentence-transformers`
- `langchain`
- `transformers`

The entire code can be viewed through the github repository.

3. Dataset

The corpus for our RAG was the Bible. Specifically, we used the **King James Version** of the Bible, formatted in .txt.

- This was available via the OpenBible website online.
- **Domain:** The dataset belongs to the *religious literature* domain, specifically focused on Christian theological texts.
- No preprocessing was done. We simply imported the text file as is, then read the entirety of it as a **single** document, then applied chunking to that document of varying sizes.

– An alternative approach could've been to read multiple verses as a single document, hence you would have multiple documents to apply chunking on.
– What wouldn't work is to have each verse as a document, as the verses are generally too small to have chunking applied to it.

4. System Architecture

4.1. Chunking

We implemented chunk sizes of 256, 512, and 1024, with an overlap of 50. The various sizes were used in order to test whether small, more concise chunks would result in more accurate results, or large, more broad chunks. Smaller chunks means that fewer verses are in individual chunks together, in some cases only one verse being a chunk. In larger chunks, more verses are grouped together. Therefore, for certain questions which contain a lot of context within the bible, the larger chunk sizes may prove better. We intended to try chunk sizes of 2048 as well, but due to time constraints decided to opt out of that.

4.2. Retrieval Methods

We implemented the following retrieval methods:

- BM25
- MMR
- Semantic
- Hybrid with RRF

The Hybrid with RRF method is just the precomputed retrievals from the previous three methods, with RRF applied to it. The implementation of four methods can be found in the python notebook included in the github repository.

4.3. Embeddings

We used the following embedding types to create our vector databases with:

- `BAAI/bge-small-en`
- `sentence-transformers/paraphrase-MiniLM-L6-v2`
- `sentence-transformers/all-MiniLM-L6-v2`

All of these can be found on the HuggingFace website, and imported through the Transformers package, or with LangChain.

4.4. LLMs

We used two LLMs:

- `Qwen/Qwen2.5-1.5B-Instruct`
- `Qwen/Qwen2.5-3B-Instruct`

We did attempt to use models with greater number of parameters, such as with 7B, however due to GPU constraints, we could only work with models with 3B parameters or fewer. We attempted to work with a couple of other LLMs as well, namely `Deepseek's 1.5B model` and `tiiuae/Falcon3-1B-Instruct`, but when we generated responses with it, we found it performed poorly, so we scrapped these two models.

4.5. Ground Truth Generation

The ground truth was generated through a custom built ChatGPT model, titled **Bible**, created by Dan An. The model can be accessed here.

5. Running the Model

5.1. Questions

We asked the model four questions, which are the following:

- What does the Bible say about forgiveness?
- What is the greatest commandment?
- What is the Bible's view on wealth and poverty?
- Explain the concept of grace in the Bible

When the ground truths were being generated by the custom ChatGPT model, we asked these questions as is, followed by the words "in paragraphs", just to constraint the ground truth to be in paragraphs.

5.2. The Execution Flow

We basically decided to run all possible combinations of the parameters in one go, as a result we used nested for loops.

- The outer most for loop was for controlling the chunking sizes to be applied to the bible document. As mentioned above, we tried four different chunking sizes.
- The next loop was the embedding type used. Based on the embedding type, the vector database could be built, alongwith the chunks from the previous for loop. We tried three different embedding types.

- The next for loop was for the LLM model used for answer generation. Although we did try two separate models, we unfortunately could not run them in the same run, due to storage constraints of the GPU RAM. Hence, we ran the Falcon 1B model in our first run, and the Qwen2.5 1.5B model in the second run.
- The next for loop is simply for the number of documents to be retrieved from the retreivel methods. We tried three different numbers.
- The next for was for the questions and answers. For each question and answer, we ran three retrieval methods (within their own, final nested for loop) and than the hybrid retrieval method which uses RRF. The final nested for loop ensures that for the hybrid method, it can simply use the precomputed results from the previous methods, rather than recomputing them, taking more time and space.

All this can perhaps be better understood from the following code snippet, outline this flow:

```
# This is a simple Python code example
for chunk_size in chunk_sizes:
    # chunking applied

    for embedding_type in
        embedding_methods:
        # vector store built

        for model_name, pipeline_model
            in llm_models.items():
            # LLM loaded

            for k_docs in
                doc_retrieval_counts:
                # Number of documents
                    to retrieve set

                for q_idx, qa in
                    enumerate(
                    questions_and_answers
                    , 1):
                    # Begin processing
                        each question
                        and answer

                    precomputed = {
                    "BM25": bm25_search
                        (question, docs
                        , k_docs),
                    "Semantic":
                        semantic_search
                        (question,
                        vectorstore,
                        k_docs),
                    "MMR": mmr_search(
                        question,
                        vectorstore,
                        k_docs)
                    }

                    for method_name in
                        ["BM25", "
                        Semantic", "MMR
                        "]:
                        # Apply and
                            generate
                            response
                            from each
                            of these
                            retrieval
                            methods

                    # Hybrid after
                        precomputation
                    # Apply Hybrid
                        method and
                        generate
                        response
```

6. Evaluation

The evaluation metrics were computed using a custom similarity-based scoring function. This function, `evaluate_response`, encodes the generated response, ground truth answer, question, and the retrieved context using the selected embedding model. Cosine similarity is then computed between the encoded response and each of the following:

- The concatenated retrieved documents (for **faithfulness**),
- The original question (for **relevance**),
- And the manually curated ground truth (for **similarity**)

We did try to implement the popular `ragas` package for evaluation, however we encountered issues with it so decided to adopt this approach instead.

7. Results

This section outlines the results of our experiments.

7.1. Quantitative Results

We list four tables, numbered 1-4 in the pages 5-6, where each table is for one of the four questions we tested with. Each table includes the average faithfulness, relevancy, and similarity scores per parameter combination, as well as the average time taken for retrieval and response generation per method.

We have not listed the number of documents, but instead incorporated them in the averages, as we did not see any significant differences in the values with different number of documents being retrieved. We have also incorporated the embedding types in the averages, although the `BAAI/bge-small-en` embedding type was superior from the two other types. We did this for sake of brevity in the tables.

Additionally, the values for model `Qwen/Qwen2.5-3B-Instruct` with chunks of sizes 1024 have not been included, because unfortunately during running, the execution unexpectedly closed, causing a significant number of parameter test cases to be missed. We have included the csv file for it and the rest of the tests in the github repository.

7.2. Insights Per Question

7.2.1. Question 1: What does the Bible say about forgiveness?

Across all retrieval strategies, MMR and Semantic Search yielded the highest performance, particularly in faithfulness and similarity. The Qwen2.5 3B model with chunk size 256 paired with MMR retrieval produced the strongest results, achieving a faithfulness score above 0.80 and relevance above 0.82. BM25 consistently underperformed in comparison. Interestingly, increasing the chunk size to 512 led to a slight drop in relevance for some combinations, suggesting that overly large chunks may dilute the focused evidence required for this specific question.

7.2.2. Question 2: What is the greatest commandment?

Performance remained strong across models, though relevance slightly dipped compared to Question 1. Semantic and MMR retrievals still led in faithfulness and similarity, particularly when used with Qwen2.5 3B. BM25 retrieval, although less competitive in other metrics, surprisingly produced some of the highest similarity scores, suggesting it occasionally returned highly lexical matching passages. Chunk size 256 again emerged as an ideal size, especially for deep, theologically nuanced queries like this one.

7.2.3. Question 3: What is the Bible's view on wealth and poverty?

The best results were achieved using Qwen2.5 3B with Semantic retrieval and a chunk size of 256, obtaining a faithfulness score above 0.86. Retrieval methods mattered greatly for this question; BM25 generally struggled due to its reliance on surface-level keyword matching. In contrast, MMR and Semantic Search excelled at retrieving precise and contextually rich verses. Larger chunks did not improve performance, reinforcing that shorter, sharper contexts are more beneficial for direct doctrinal questions.

7.2.4. Question 4: Explain the concept of grace in the Bible.

This question saw strong and consistent performance across all settings, with high scores in faithfulness and similarity. Qwen2.5 1.5B with Semantic retrieval at chunk size 512 achieved the top score (faithfulness: 0.88). Unlike previous questions, larger chunks appeared to help here, likely because the topic is more dispersed across different scriptures, and it also wasn't a direct question, but rather an prompt inquiring more about a specific topic within the Bible. The Hybrid RRF method also held up well, maintaining high similarity and relevance with lower variance across runs. This suggests that nuanced themes with broad scriptural distribution benefit from longer context and retrieval fusion.

7.2.5. Overall Trends

Semantic and MMR retrieval consistently outperformed BM25 in faithfulness and relevance. The Qwen2.5 3B model showed superior performance over 1.5B in nearly all settings. Smaller chunk sizes (256 or 512) were generally more effective, while 1024 often led to reduced relevance. Hybrid RRF remained a stable, high-performing fallback, offering robust performance when individual methods varied.

8. Challenges Faced and Mistakes Made

Note: These are in no particular order.

8.1. Hardware constraints

Neither of us had any solid hardware to work on, and crucially, neither of us had any GPUs of our own. Hence, we had to work with online third parties, Kaggle and Google Colab. Although both of these did provide us with GPUs, there were still two issues:

- The GPUs were only given for a limited time. Although Kaggle did grant free access for a lot longer, Google Colab didn't, and in one instance we were

Table 1. Performance On Question 1

Model	Chunk Size	Retriever	Faithfulness	Relevancy	Similarity	Time (s)
Qwen2.5 1.5B	256	BM25	0.516	0.8282	0.881233333	7.009
Qwen2.5 1.5B	256	MMR	0.821911111	0.846044444	0.868455556	7.556080129
Qwen2.5 1.5B	256	Semantic	0.860111111	0.842822222	0.869355556	8.267180469
Qwen2.5 1.5B	256	Hybrid with RRF	0.785077778	0.845922222	0.876177778	6.964363072
-	-	-	-	-	-	-
Qwen2.5 1.5B	512	BM25	0.68085	0.8447375	0.8699375	9.628755689
Qwen2.5 1.5B	512	MMR	0.8273125	0.847625	0.886725	5.606778353
Qwen2.5 1.5B	512	Semantic	0.7894	0.834625	0.875075	5.666351348
Qwen2.5 1.5B	512	Hybrid with RRF	0.771425	0.8462125	0.864125	7.201993048
-	-	-	-	-	-	-
Qwen2.5 3B	256	BM25	0.557433333	0.848	0.859	11.55359504
Qwen2.5 3B	256	MMR	0.806088889	0.826611111	0.866355556	13.63438535
Qwen2.5 3B	256	Semantic	0.841911111	0.8418	0.862877778	11.86453162
Qwen2.5 3B	256	Hybrid with RRF	0.776255556	0.847933333	0.875277778	13.12142801
-	-	-	-	-	-	-
Qwen2.5 3B	512	BM25	0.642422222	0.802911111	0.797977778	23.66356378
Qwen2.5 3B	512	MMR	0.819544444	0.847266667	0.889855556	12.20475067
Qwen2.5 3B	512	Semantic	0.777711111	0.842522222	0.874211111	12.64793571
Qwen2.5 3B	512	Hybrid with RRF	0.762788889	0.839888889	0.871455556	15.21990712
-	-	-	-	-	-	-
Qwen2.5 3B	1024	BM25	0.594044444	0.831511111	0.814722222	8.94006896
Qwen2.5 3B	1024	MMR	0.755866667	0.818844444	0.853655556	15.64353
Qwen2.5 3B	1024	Semantic	0.782488889	0.837477778	0.842433333	12.98390577
Qwen2.5 3B	1024	Hybrid with RRF	0.791722222	0.831566667	0.847033333	66.12649915
-	-	-	-	-	-	-

Table 2. Performance On Question 2

Model	Chunk Size	Retriever	Faithfulness	Relevancy	Similarity	Time (s)
Qwen2.5 1.5B	256	BM25	0.588944444	0.724233333	0.886544444	256.2775557
Qwen2.5 1.5B	256	MMR	0.8801	0.761577778	0.889933333	200.5302901
Qwen2.5 1.5B	256	Semantic	0.792777778	0.717544444	0.854155556	229.0375716
Qwen2.5 1.5B	256	Hybrid with RRF	0.815955556	0.746933333	0.867488889	260.913319
-	-	-	-	-	-	-
Qwen2.5 1.5B	512	BM25	0.6584625	0.6972375	0.8859875	134.8973578
Qwen2.5 1.5B	512	MMR	0.7903	0.710142857	0.830685714	266.5621638
Qwen2.5 1.5B	512	Semantic	0.861814286	0.685228571	0.857185714	194.1158133
Qwen2.5 1.5B	512	Hybrid with RRF	0.8369	0.65765	0.841216667	185.1990354
-	-	-	-	-	-	-
Qwen2.5 3B	256	BM25	0.599144444	0.741166667	0.909755556	18.55407683
Qwen2.5 3B	256	MMR	0.877411111	0.763522222	0.89320	8.929626465
Qwen2.5 3B	256	Semantic	0.854866667	0.740133333	0.909188889	7.27322793
Qwen2.5 3B	256	Hybrid with RRF	0.810444444	0.760388889	0.897766667	7.194968939
-	-	-	-	-	-	-
Qwen2.5 3B	512	BM25	0.551222222	0.768911111	0.901	10.83972081
Qwen2.5 3B	512	MMR	0.800811111	0.742522222	0.868366667	10.30893016
Qwen2.5 3B	512	Semantic	0.838588889	0.721566667	0.874944444	12.3094959
Qwen2.5 3B	512	Hybrid with RRF	0.741166667	0.707422222	0.872677778	11.61436587
-	-	-	-	-	-	-
Qwen2.5 3B	1024	BM25	0.517577778	0.731977778	0.906933333	20.06977529
Qwen2.5 3B	1024	MMR	0.800244444	0.759055556	0.883477778	12.21029594
Qwen2.5 3B	1024	Semantic	0.792144444	0.749277778	0.870911111	11.86988333
Qwen2.5 3B	1024	Hybrid with RRF	0.812855556	0.750455556	0.866833333	13.11012401
-	-	-	-	-	-	-

Table 3. Performance On Question 3

Model	Chunk Size	Retriever	Faithfulness	Relevancy	Similarity	Time (s)
Qwen2.5 1.5B	256	BM25	0.587822222	0.735111111	0.790822222	5.00329362
Qwen2.5 1.5B	256	MMR	0.8427	0.8102625	0.8587625	69.42085502
Qwen2.5 1.5B	256	Semantic	0.8462125	0.814825	0.853	7.104887336
Qwen2.5 1.5B	256	Hybrid with RRF	0.801175	0.783175	0.830825	69.50324094
-	-	-	-	-	-	-
Qwen2.5 1.5B	512	BM25	0.640566667	0.784733333	0.82555	7.110291322
Qwen2.5 1.5B	512	MMR	0.74685	0.850716667	0.89455	7.090352774
Qwen2.5 1.5B	512	Semantic	0.7389	0.84015	0.893666667	95.70887677
Qwen2.5 1.5B	512	Hybrid with RRF	0.735266667	0.7975	0.86795	50.2231009
-	-	-	-	-	-	-
Qwen2.5 3B	256	BM25	0.622855556	0.822222222	0.848377778	13.15052083
Qwen2.5 3B	256	MMR	0.845033333	0.832444444	0.881433333	16.25419328
Qwen2.5 3B	256	Semantic	0.861644444	0.834366667	0.863255556	15.72629828
Qwen2.5 3B	256	Hybrid with RRF	0.818144444	0.805555556	0.843044444	15.10788512
-	-	-	-	-	-	-
Qwen2.5 3B	512	BM25	0.571911111	0.773266667	0.821011111	13.71665147
Qwen2.5 3B	512	MMR	0.783955556	0.829366667	0.8655	16.64021034
Qwen2.5 3B	512	Semantic	0.801811111	0.801477778	0.815233333	16.16150088
Qwen2.5 3B	512	Hybrid with RRF	0.867177778	0.8334	0.756888889	17.93483173
-	-	-	-	-	-	-
Qwen2.5 3B	1024	BM25	0.591533333	0.811122222	0.830566667	12.91108489
Qwen2.5 3B	1024	MMR	0.828666667	0.791255556	0.8324	20.28929639
Qwen2.5 3B	1024	Semantic	0.8108	0.770811111	0.813777778	14.49453878
Qwen2.5 3B	1024	Hybrid with RRF	0.793877778	0.812744444	0.839955556	12.23761272
-	-	-	-	-	-	-

Table 4. Performance On Question 4

Model	Chunk Size	Retriever	Faithfulness	Relevancy	Similarity	Time (s)
Qwen2.5 1.5B	256	BM25	0.7442125	0.83935	0.898	7.623432785
Qwen2.5 1.5B	256	MMR	0.7684125	0.857025	0.9118125	6.731403142
Qwen2.5 1.5B	256	Semantic	0.8022875	0.8549375	0.8998	7.138631135
Qwen2.5 1.5B	256	Hybrid with RRF	0.7928	0.8272	0.8897	6.518519551
-	-	-	-	-	-	-
Qwen2.5 1.5B	512	BM25	0.76465	0.839683333	0.913783333	7.045374235
Qwen2.5 1.5B	512	MMR	0.860883333	0.774633333	0.803	10.76220032
Qwen2.5 1.5B	512	Semantic	0.887616667	0.805033333	0.829983333	10.52837678
Qwen2.5 1.5B	512	Hybrid with RRF	0.85545	0.81545	0.851666667	8.794382215
-	-	-	-	-	-	-
Qwen2.5 3B	256	BM25	0.760888889	0.840566667	0.876788889	11.61493738
Qwen2.5 3B	256	MMR	0.823966667	0.840577778	0.890388889	14.89945576
Qwen2.5 3B	256	Semantic	0.8199	0.8433	0.882444444	23.35836649
Qwen2.5 3B	256	Hybrid with RRF	0.813966667	0.834322222	0.882911111	14.92152969
-	-	-	-	-	-	-
Qwen2.5 3B	512	BM25	0.723455556	0.812611111	0.897977778	11.28019402
Qwen2.5 3B	512	MMR	0.822788889	0.843677778	0.872033333	14.61298119
Qwen2.5 3B	512	Semantic	0.879777778	0.836855556	0.808644444	14.63691754
Qwen2.5 3B	512	Hybrid with RRF	0.810066667	0.8114	0.875811111	15.13038251
-	-	-	-	-	-	-
Qwen2.5 3B	1024	BM25	0.711488889	0.829111111	0.896133333	14.76184861
Qwen2.5 3B	1024	MMR	0.748966667	0.835777778	0.872255556	15.45722135
Qwen2.5 3B	1024	Semantic	0.730022222	0.860244444	0.888822222	15.0244311
Qwen2.5 3B	1024	Hybrid with RRF	0.733166667	0.8497	0.921066667	19.06064394
-	-	-	-	-	-	-

forced to stop using Colab for a while as we had exhausted the entire alotted free time for the GPU.

- The GPUs still weren't enough to run bigger LLM models, which perhaps may have resulted in better results.

8.2. Annoyance of Working with Third Party Sites

We are appreciative of working with Kaggle and Google Colab, but it would be a lie if we said we having smooth sailings with it. On the contrary, we faced quite a bit of additional stress due to "small annoyances."

- An issue that occurred often was that, due to inactivity, the platforms would stop executing our code, causing us to lose a lot of time spent on running a model.
- At times, both platforms were excruciatingly slow, which only increased the overall time running the model. Paired with the above nuisance, we ended having to sit at our screens waiting for the models to run to completion and quickly save the results.

8.3. Generation of the Ground Truth

We originally intended to generate ground truth through a separate LLM via the code itself. However, due to GPU constraints, we could only load one LLM in a run, and hence, we had to back out of that idea.

We then decided to use a custom built GPT agent, as mentioned earlier, that was custom made solely for the Bible. Hence, we believed it would be a good baseline to judge our responses with.

We believe we did make an error, in that we did not restrict its response length. As a result, it produced long answers, about 3-5 paragraphs for each question. Contrast this with our generated responses, which can hardly produce a paragraph, let alone 3-5, and it ended up resulting in poor semantic scores.

8.4. Inability to Work with Ragas

For the evaluation of the generated responses of the LLMs, we initially had tried to work with Ragas, which is a standard package to use for this task. However, we struggled with setting it up. In particular, it required us to work with an OpenAI API. Although we did get that, even then we could not jump over the hurdle.

We are confident that, with no time constraints (and especially with no other substantial workload), we would've been able to implement Ragas. However, we later switched to finding the cosine similarities instead.

9. Best Model Configuration

After extensive evaluation across multiple configurations, the combination that consistently outperformed others was the **Qwen2.5 3B** model paired with the **MMR retriever**, using a **chunk size of 256**, and **BAAI/bge-small-en** embeddings. This setup achieved the highest faithfulness and similarity scores across most questions while maintaining strong relevance and reasonable generation times. The fine-grained chunking allowed the model to focus on more coherent and precise contexts, and MMR retrieval ensured a diverse yet relevant selection of documents that captured both lexical and semantic variety.

The BAAI embedding model (`bge-small-en`) consistently outshined other embedding types, such as `MiniLM`, particularly in scenarios involving nuanced or theologically dense queries. It enabled better semantic representation of biblical text, which was crucial for both retrieval relevance and evaluation via cosine similarity. Furthermore, models using BAAI embeddings displayed more stable performance across all chunk sizes and questions, further affirming its robustness for Bible-based retrieval-augmented generation tasks.

10. Additional Figures

Here we attach a few images highlighting the work of the project.

10.1. Main Execution Flow Output

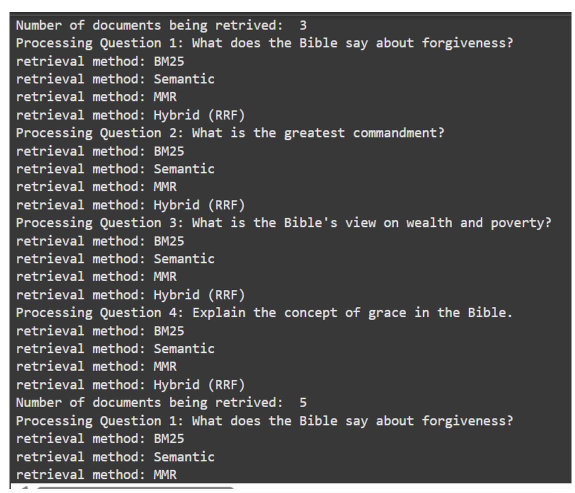

Figure 1. An Example of Output for Main Execution Flow

10.2. Results From CSV Files

All the CSV files can be found in the github repository, alongwith with the code and the corpus - namely, the KJV Bible.

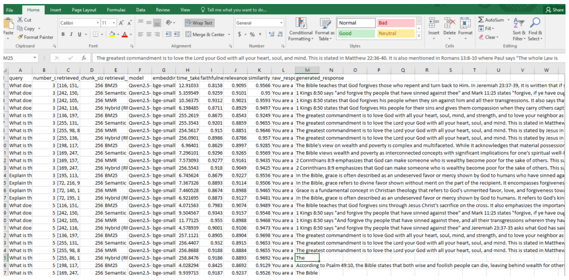

Figure 2. qwen 1.5B Sample Output

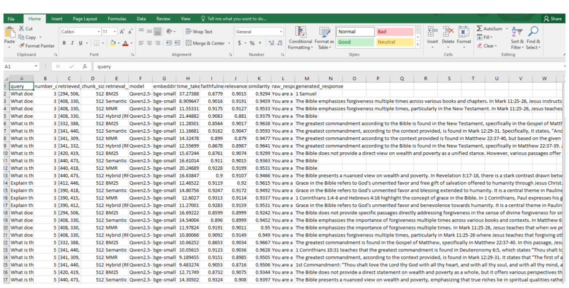

Figure 3. Qwen 3B Sample Output