Automated Fact-Checking System: Bi-Encoder + FAISS Retrieval and BERT Classification

Group 121

Abstract

Climate change is a significant threat to society, a challenge that is speeding up along with misinformation around the internet. This research aims to develop an automated fact-checking system to verify climate-related claims by retrieving relevant evidences and classifying the claim's factuality. It breaks down into two main phases, initially, by retrieving the top N most relevant evidences from a knowledge source, and then it classifies the claim's correctness based on the retrieved evidences. Our findings fights to climate science's misinformation and promote reliable public statements through scalable, automated verification. On the leaderboard, our system performed second place with a harmonic score of 0.3.

1 Introduction

As digital content has grown exponentially, so too has the spread of misinformation and distorted facts (Rodríguez-Ferrándiz, 2023). In particular, the proliferation of unverified claims related to climate science presents a major challenge.Traditionally, the fact-checking process has been conducted manually (Guo et al., 2022), which is highly resource-intensive, demanding significant time, labor, and effort (Atanasova et al., 2020). This highlights the urgent need for an automated fact-checking solution (Hanselowski et al., 2019). This project addresses this problem by developing a system designed to automate the process of verifying climate-related claims. Our system implements a three-stage pipeline consisting of an evidence retrieval system followed by a claim-evidence classification system. The retrieval section utilises a dense retrieval approach, influenced by the framework of (Karpukhin et al., 2020), leveraging bi-encoder embeddings and FAISS indexing (Johnson et al., 2019) to efficiently find relevant evidences from a large corpus, with a potential enhancement like cross-encoder re-ranking. Next side, classification module utilises a Transformer-based model to determine the relationship `SUPPORTS`, `REFUTES`, `NOT_ENOUGH_INFO`, or `DISPUTED` between a claim and retrieved evidences, with a final claim label derived from aggregating evidence-level predictions through majority voting. System development and evaluation is done using existing sets of tests provided from the University of Melbourne's dataset. Performance is analysed using three key metrics: evidence Retrieval F-score, Claim Classification Accuracy, and their Harmonic Mean. This report details our system's architecture, justifies key design choices with references, presents quantitative results and analysis on the development set, and discusses system strengths, limitations and insights gained.

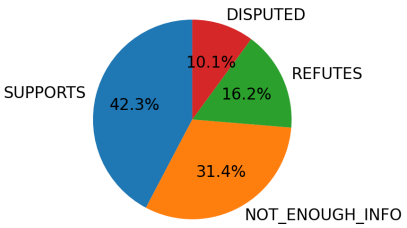

Figure 1: Distribution of Train Dataset

2 Approach

The automated fact-checking system follows a two-stage pipeline: evidence retrieval and claim classification. We did not apply explicit preprocessing such as lowercasing, stopword removal, or lemmatisation, as modern tokenizers and Transformers handle tokenization and normalization internally. This design allows for independent optimization of retrieving relevant information and classifying claim veracity. Given a claim, the system retrieves supporting evidence from a large corpus and labels it as `SUPPORTS`, `REFUTES`, `NOT_ENOUGH_INFO`, or `DISPUTED`. As a note, the training data exhibits

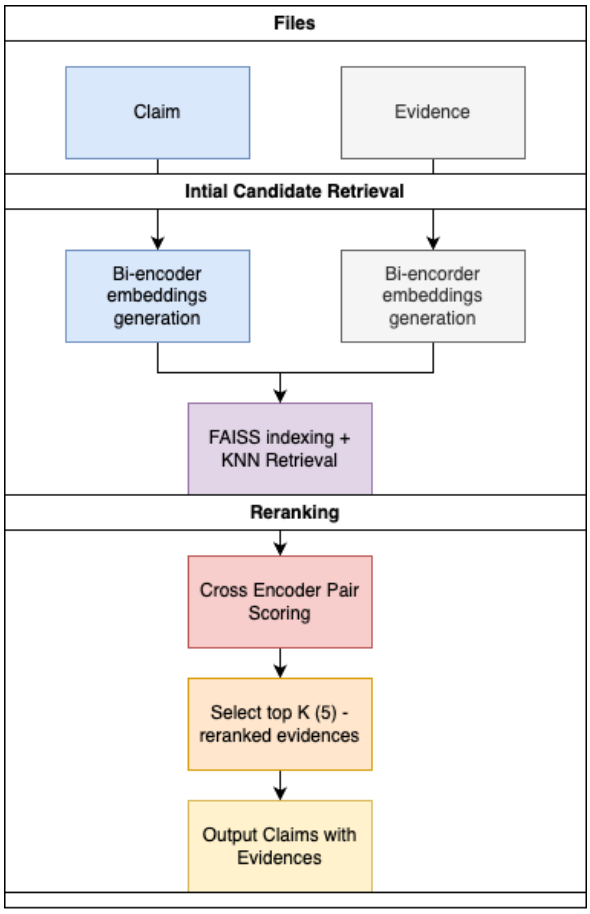

Figure 2: High-level pipeline for evidences retrieval (bi-encoder + FAISS → cross-encoder reranking)

label imbalance, noting this is a decision factor for heuristic labelling for equal majority vote for classification, and bringing bias towards support seen this diagram 1.

2.1 Phase 1: Evidence Retrieval

This phase is responsible for efficiently searching the large corpus of evidence sentences (`evidence.json`) provided from the University of Melbourne, to identify a relevant subset for a given claim. It is implemented as a two-stage process which are Initial candidate retrieval followed by reranking, refer to 2.

2.1.1 Initial Candidate Retrieval

Architecture/Method We use a bi-encoder architecture based on SentenceTransformer (all-MiniLM-L6-v2), which independently encodes claims and evidence sentences into fixed-size embeddings. The model builds on the Sentence-BERT framework (Reimers and Gurevych, 2019), leveraging a Siamese BERT structure to produce semantically rich sentence representations. We selected the MiniLM variant for its strong trade-off between efficiency and performance, especially in resource-constrained settings (Wang et al., 2020).

Rationale Bi-encoders are selected for their efficiency in handling over 120,000 evidences. Evidence embeddings can be precomputed, enabling fast similarity search at query time. The `all-MiniLM-L6-v2` model balances performance and computational cost, making it suitable for resource-limited environments such as Google Colab.

Training The bi-encoder is fine-tuned on `train-claims.json` using `MultipleNegativesRankingLoss`, which pulls true (claim, evidence) pairs closer in embedding space while pushing apart in-batch negatives. Positive pairs are generated via `load_positive_train_data`.

Similarity Search Evidence embeddings are L2-normalized and indexed using FAISS (`faiss.IndexFlatL2`). Querying with normalized claim embeddings approximates cosine similarity. FAISS efficiently handles large-scale vector search (over 1.2M entries), offering speed and memory benefits suitable for environments like Google Colab (Johnson et al., 2019).

$$\text{sim}(\mathbf{a}, \mathbf{b}) = \widehat{\mathbf{a}}^\top \widehat{\mathbf{b}} = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \, \|\mathbf{b}\|} = \cos(\mathbf{a}, \mathbf{b})$$

Action For each claim, the `retrieve` function performs a KNN search using the FAISS index to find the embeddings of retrieval-top-k most similar evidence. The code indicates `retrieval_top_k` is set to a larger value (e.g. 150) to maximise recall at this initial stage.

2.1.2 Re-ranking

Architecture/Method We use a cross-encoder model (`ms-marco-MiniLM-L-6-v2`) during the reranking stage. Unlike bi-encoders, cross-encoders take a (claim, evidence) pair as joint input and compute a relevance score, enabling richer interaction between texts, refer to 2.

Rationale Cross-encoders, though slower, yield higher accuracy by applying full attention over (claim, evidence) pairs. Our two-stage pipeline combines a fast bi-encoder for high recall with a precise cross-encoder for reranking. Since we only have positive (claim, evidence) pairs (typically 5

per claim), manually creating negatives introduces extreme imbalance (e.g., 5 positives vs. 1.2M negatives), as seen figure 1. MultipleNegativesRankingLoss handles this by treating other positives in the batch as negatives, making it ideal for our setup (Reimers and Gurevych, 2019).

Training The cross-encoder is fine-tuned. Training data is prepared using load_positive_train_ds and includes mined hard negatives using .util.mine_hard_negatives. Hard negatives are crucial for improving the model's ability to distinguish between relevant and irrelevant passages, especially those that are semantically similar to the claim but do not actually supporting. The CrossEncoderTrainer is used, configured for evaluation with CrossEncoderRerankingEvaluator. The code specifies MultipleNegativesRankingLoss, though cross-entropy on explicit pairs is more typical for cross-encoders.

Action For each claim, the `retrieval_top_k` candidate evidenced identified in the first stage is paired with the claim text. These pairs are then passed through the fine-tuned cross-encoder to obtain a relevance score for each pair. The claim-evidence pairs are sorted in descending order based on their cross-encoder scores. The top `rerank_top_k` evidencess are selected as the final retrieved set. An optional `score_threshold` can be applied, though at least one evidences is always kept if the threshold is active. However, we did not use the threshold at the end. Because we find that no threshold always gives better F-scores. The code suggests `rerank_top_k` is set to 10 for the classification phase. The output of this phase for the test set is saved to an intermediate JSON file (e.g., `test-output.json`). Which now consists of evidences that is related to the claims.

2.2 Phase 2: Claim Classification

To make the final prediction, we have the next step of claim classification, which the phase takes the claim and evidences retrieved by Phase 1 and predicts the final status of the claim. We used several techniques such as RNN, LSTM, BiLSTM and Transformer, but eventually we used Transformer, which gave the best classification prediction.

Architecture/Method For the sequence classification, we have chosen the model of distilbert-base-uncased Transformer. This is a distilled, smaller version of BERT along with no casing for the alphabet (Sanh et al., 2019). An `AutoTokenizer` is loaded for preprocessing. Refer to 3

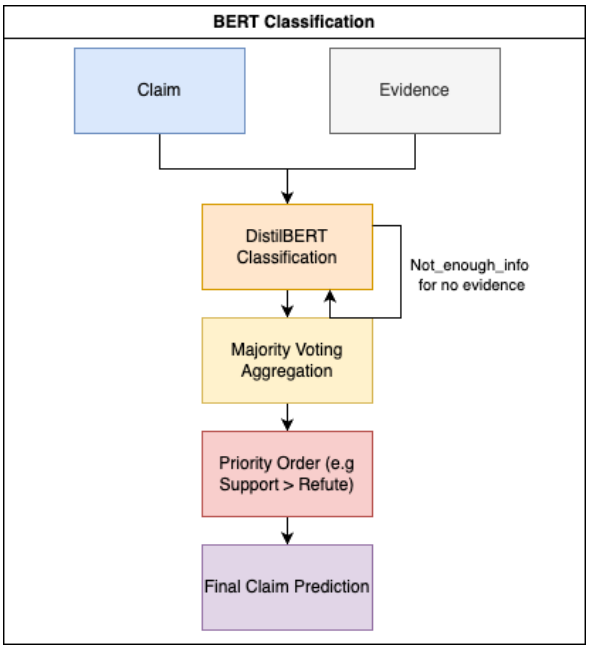

Figure 3: Transformer claim classification architecture

Rationale Transformer models like DistilBERT are state-of-the-art for various NLP classification tasks. DistilBERT offers a decent performance with less computational efficiency, making it suitable for training within Google Colab's free tier limitations. Using an "uncased" model simplifies text processing.

Data Preparation Training instances for the classifier are created from the provided labelled data (`train-claims.json`, `dev-claims.json`). For every claim in the training set and each of its evidences, a training instance is generated by concatenating the claim text and the evidence text using a special separator token (Devlin et al., 2019).

$$\underbrace{\texttt{"Claim text"}\ [\texttt{SEP}]\ \texttt{"evidences text"}}$$

Input to classifier, where [SEP] is tokenizer's seperator for ambiguity

A custom PyTorch `ClaimEvidenceDataset` is used to handle tokenisation, padding, and truncation of these combined texts to a maximum length of 128 tokens (`max_len=128`). This teaches the model to classify the relationship between a single claim and a single piece of evidence. Claims labelled as `NOT_ENOUGH_INFO` in the training data will generate training instances labelled specifically. If they have no associated evidence, they do not contribute training instances to the classifier. Considering there was always K evidences per claim,

there was no need to combine multiple evidences as it doesn't get skewed.

Training The `distilbert-base-uncased` model is fine-tuned on the prepared training data. A manual PyTorch training loop is implemented, running for 3 epochs. The AdamW optimiser is used with a learning rate of `1e-5`. Training progress is monitored by printing the running loss and evaluating on the development set each epoch, reporting accuracy and a classification report. The limited number of epochs is likely chosen considering training time constraints and to potentially prevent overfitting.

2.2.1 Prediction and Aggregation

After training, the model's state dictionary is saved. For the test set (loaded from the intermediate retrieval output `test-output.json`), predictions are made for each claim. For a given test claim, the `prepare_test_data` function creates input items by pairing the claim text with each of its retrieved evidences. The `predict_claim_label` function processes these items. 1. The trained classifier predicts a label `SUPPORTS`, `REFUTES`, `NOT_ENOUGH_INFO`, or `DISPUTED` for each (claim, retrieved_evidence) pair. 2. The final predicted label for the claim is determined by a **majority vote** among the labels predicted for its retrieved evidences. 3. A crucial heuristic is applied for tie-breaking:. If there is a tie in the counts of predicted labels, the label is chosen based on a predefined priority order. `SUPPORTS` > `NOT_ENOUGH_INFO`> `REFUTES` > `DISPUTED` in the order of the training dataset's label distribution, as seen here 1 The final predicted labels are written to a new JSON file (e.g. `test-output-updated.json`).

3 Experiments

3.1 Evaluation Method

For retrieval, we evaluate by computing precision, recall and F1 score. On the other hand, the classification component was done via accuracy and validation F1 score to grab the best model.

3.2 Experimental Details

Default setting: retrieve top-30 evidences via bi-encoder and rerank top-5 via cross-encoder (threshold $= 0$ on normalized $[0, 1]$ scores). Models are trained on `train-claim.json` and evaluated on `dev-claim.json`. Below are the experiments.

1) Pretrained models and fine-tuning combinations

We tried different combinations with bi-encoder and cross-encoder and whether they are fine-tuned. Results show that fine-tuned bi-encoder and fine-tuned cross-encoder outperform, results in this figure 4.

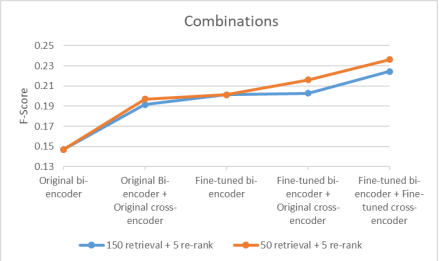

Figure 4: Performance comparison of model combinations: bi-encoder vs. bi-encoder + cross-encoder

2) Loss function for cross-encoder

We set a different loss function of `MultipleNegativesRankingLoss`, instead of the default loss function, cross-entropy loss, for the cross-encoder. MultipleNegativesRankingLoss has better performance than cross-entropy loss. The fine-tuned cross-encoder performs extremely badly with cross-entropy loss. The reason is that cross-entropy loss expects negative examples from the training dataset, which we do not have unless we create negative examples manually. MultipleNegativesRankingLoss is great because it only receives positive examples that we have and creates negative examples automatically. As a result, MultipleNegativesRankingLoss can help fine-tune the cross-encoder to get F-scores over 0.2, while the default cross-entropy loss only gives F-scores below 0.1.

3) Different number of retrieved evidences from the bi-encoder

Our retrieval system has two parts: Bi-encoder retrieves a relatively large amount (10–500) of evidences from the evidence dataset, as seen in Figure 5. Cross-encoder re-ranks this retrieved evidence and returns the top-k relevant evidences. For the first part, prior work has shown that retrieval size affects reranking performance (Guo et al., 2022) and we tried different numbers of retrieved evidence. Results show that retrieving 30 evidences at the first stage outperforms. Although fewer retrieved evidences decreases the recall, it narrows down the scope of evidences for our cross-encoder to re-rank, which increases the precision

and F-score.

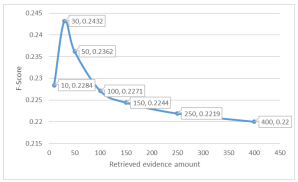

Figure 5: Retrieval F1 score vs Evidence Amount

4) Score thresholds for the cross-encoder's predictions

Cross-encoder receives two texts and predicts a score to represent their similarity. However, scores are not bound, which is inconvenient for us to set a threshold. So we normalised scores to [0,1]. We set thresholds that only evidence with a normalised score predicted by cross-encoder higher than the threshold can be returned as our predicted evidence for a claim. We hoped that thresholds can help us to predict relevant evidences for claims more precisely, rather than simply returning a fixed number of k re-ranked top evidences. Results show that thresholds drop the performance in Figure 6. Therefore, we do not adopt a threshold for our cross-encoder.This result is inconsistent with the finding of (Zhang et al., 2020) that thresholding helps improve reranking accuracy.

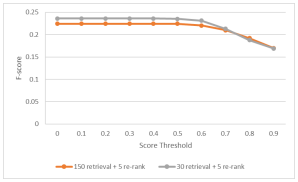

Figure 6: Showing the Worse Effect of cross-encoder score thresholds on retrieval precision and recall

3.3 Results

Table 1 reports our system's performance on the development set, alongside the provided baseline. While our classifier obtains substantially higher accuracy, our retrieval component lags behind.

	F-score	Accuracy	Harmonic Mean
Baseline	0.338	0.351	0.344
Ours	0.238	0.442	0.31

Table 1: Development set evidences-retrieval F-score, claim classification accuracy, and their harmonic mean.

Table 2 and Figure 7 show how the classification model's training loss, validation accuracy, and F1

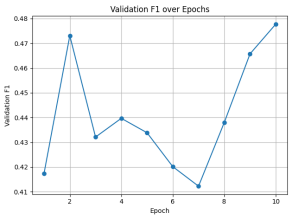

Figure 7: Validation accuracy and F1 over epochs during Transformer training

evolve over 10 epochs. Performance peaks at epoch 5 (F1=0.450, Acc=0.464) before a slight decline, suggesting mild overfitting afterwards.

Epoch	Train loss	Val Acc	Val F1
1	1.1222	0.5010	0.4174
2	0.7466	0.5092	0.4731
3	0.3868	0.4481	0.4321
4	0.1935	0.4481	0.4397
5	0.1035	0.4420	0.4339
6	0.0543	0.4073	0.4202
7	0.0347	0.4196	0.4123
8	0.0229	0.4562	0.4380
9	0.0160	0.4644	0.4657
10	0.0131	0.4786	0.4778

Table 2: Loss and validation performance by epoch for Transformer

Discussion with Result Our classification component outperforms the random-baseline's accuracy (0.442 vs. 0.351) and achieves a strong weighted F1 (0.450). However, evidences retrieval remains the bottleneck (F=0.238 vs. baseline's 0.338), pulling down the overall harmonic mean (0.296). The learning curves show rapid convergence by epoch 10 seen 7, with slight overfitting. The leaderboard score was surprising as we achieved top 2 (8).

4 Conclusion

In this project. We found that pretrained models are powerful for NLP tasks. Fine-tuning plays an significant role in helping models focus on our own tasks. We have learnt how to preprocess data to satisfy requirements of pretrained models. We have learnt how hyperparameters like learning rate, epochs amount, batch size can influence our models. We have learnt how to choose suitable loss functions based on what format of datasets we have. Limitation: A small training dataset results in early overfitting and low performance. In the future 1.

We will preprocess the datasets more carefully. We will check the formats of texts. We will remove duplicates or nearly Near-Duplicates on the dataset because we adopts MultipleNegativesRankingLoss. Suppose there are two duplicated evidences, one is set to ground truth while the other is not. MultipleNegativesRankingLoss will treat the other as negative, which introduces bias to our model. 2. We will try other pretrained model and test their performance.

Team Contributions

- **John Kim**:
  - Analysed dataset splits and identified label distribution bias
  - Implemented the claim classifier and evaluation pipeline
  - Fine-tuned the Transformer classification model
  - Debugging and code fixes for whole code
  - Ran `eval.py` for metric calculation for dev
  - Performed error analysis and curve-tracking F_1 reporting for our classification
  - Generated result tables and figures and uploaded to OverLeaf
  - Experimented with and optimised hyperparameters to suit Google Colab (2-hour) constraints (learning rate, batch size, warm-up ratio, weight decay, epochs) and made google colab runnable with imports and file downloads
  - Organised the LaTeX manuscript and Colab notebooks
  - Set up Git version control and Colab environment
  - Wrote report (abstract, introduction, approach, result) , formulas, and figures
- **Guwei Ke**:
  - Implemented data preprocessing, which we eventually deleted
  - Built retrieval models (bi-encoder + FAISS)
  - Also built re-ranking (cross-encoder)
  - Updated the report with experiments and conclusiondraw figures, update the report with passages about evidence retrieval.

Task:					Results Public Evaluation Development Task		
#	Participant	Entries	Date	ID	Harmonic Mean of F and A	Evidence Retrieval F-score	Claim Classification Accuracy
1	kersapphire	1	2025-05-11 20:27	285300	0.33	0.25	0.49
2	guweik	1	2025-05-12 03:44	285572	0.3	0.23	0.43

Figure 8: Guwei Achieving top 2 - harmonic mean 0.3

  - Draw figures for experiments
  - Conducted experiments to compare:
    1. original bi-encoder
    2. original bi-encoder + original cross-encoder
    3. fine-tuned bi-encoder
    4. fine-tuned bi-encoder + original cross-encoder
    5. fine-tuned bi-encoder + fine-tuned cross-encoder
    6. CosineSimilarityLoss vs. MultipleNegativesRankingLoss
    7. different numbers of retrieved evidence from the bi-encoder
    8. score thresholds for the cross-encoder
- **Lanye Shao**:
  - Implemented and integrated an alternative LSTM-based claim classification model (BiLSTM + Attention)
  - Designed a modular architecture to support switching between LSTM and Transformer classifiers using a single flag
  - Added relevant references and BibTeX entries for cited retrieval and classification models
  - Adjust report text, formulas, and figures

References

Pepa Atanasova, Jekaterina Simonsen, Christina Lioma, and Isabelle Augenstein. 2020. Generating fact checking explanations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 7352–7364. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186.

Zhijiang Guo, Michael Schlichtkrull, and Andreas Vlachos. 2022. A survey on automated

fact-checking. *Transactions of the Association for Computational Linguistics*, 10:178–206.

Andreas Hanselowski, Christian Stab, Claudia Schulz, Zhilin Li, and Iryna Gurevych. 2019. A richly annotated corpus for different tasks in automated fact-checking. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 493–503. Association for Computational Linguistics.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3982–3992.

Raúl Rodríguez-Ferrándiz. 2023. An overview of the fake news phenomenon: From untruth-driven to post-truth-driven approaches. *Media and Communication*, 11(2):15–29.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Weijia Wang, Furu Wei, Li Dong, Haifeng Bao, Nan Yang, and Ming Zhou. 2020. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*, pages 5774–5786.

Shuohang Zhang, Meng Qu, Furu Wei, Haocheng Sun, Minlie Huang, and Ming Zhou. 2020. Reconsider: Re-ranking using contrastive learning for passage retrieval. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4090–4100. Association for Computational Linguistics.