Varun Kadam
A20593606

Build Fellowship

Comprehensive Report:

Fine-Tuning Sentence Transformer Models for Enhanced Job Posting Similarity

1. Introduction & Project Goals

- **Context:** The ability to accurately determine the similarity between job titles is crucial for various applications, including job recommendation systems, candidate matching platforms, resume analysis, and labor market analytics. Standard keyword matching often fails to capture semantic nuances (e.g., "Software Engineer" vs. "Code Developer") or variations in phrasing ("Senior Data Scientist" vs. "Data Scientist, Senior Level"). General-purpose language models, while powerful, may not possess the domain-specific understanding required for fine-grained distinctions within the specialized vocabulary of job titles.
- **Problem Statement:** Pre-trained sentence embedding models provide a strong baseline for semantic similarity but often lack the specific context required to excel in niche domains like job title comparison. Their general training might not optimally cluster closely related but distinct job roles or variations of the same role. Furthermore, leveraging large language model (LLM) APIs for this task presents challenges in terms of consistency, cost, and potential failures for certain model types.
- **Project Objective:** This project aimed to significantly enhance the accuracy of job title similarity measurement by fine-tuning a selected sentence transformer model. The primary goal was to develop a specialized embedding space tailored to the nuances of job titles, leading to more relevant search results and similarity comparisons within a practical application. A secondary objective was to evaluate the suitability of various off-the-shelf LLM APIs for this specific semantic similarity task compared to a dedicated, fine-tuned embedding approach.
- **Scope:** The project involved data preparation using a dataset of job titles and their variations ("jittered" titles), experimentation with different sentence transformer base models, evaluation of several leading LLM APIs, implementation of a fine-tuning process using triplet loss, visualization of the resulting embedding spaces, and integration into a demonstration application (Streamlit).

Varun Kadam
A20593606

2. Data Acquisition and Preparation

- **Dataset Source:** The foundation of the fine-tuning process was a dataset containing job titles. A key component was the concept of "jittered" titles – variations or alternative phrasings of canonical "seed" job titles. This structure is ideal for contrastive learning approaches like triplet loss. The dataset located at synthetic_data/data/jittered_titles.csv served as the source. Each entry likely contained a seed title, a jittered version, and potentially associated metadata like ONET codes.
- **ONET Code Stratification:** To ensure the model generalized well across different job families and wasn't biased towards over-represented categories, the data was split into training, validation, and testing sets using stratification based on ONET codes. ONET codes provide a standardized classification of occupations. This stratification ensures that the diversity of job types present in the original dataset is proportionally represented in each subset.
    - The process involved identifying unique ONET codes and associated seed titles.
    - Seed titles were carefully allocated to validation (val_df) and test (test_df) sets, ensuring that titles corresponding to a specific seed title did not appear across different splits (preventing data leakage).
    - The remaining data formed the training set (train_df).
    - This splitting logic is implemented within the initial part of fine-tuning-build-project/fine_tuning/fine-tuning_jitter_dataset.py.
- **Dataset Splits & Sizes:** After processing, the datasets were saved to fine_tuning/data/datasets/:
    - train_ds.csv: Used for model training. Size: (Reported in script output as len(train_df))
    - val_ds.csv: Used for evaluating the model during training (hyperparameter tuning, early stopping). Size: (Reported in script output as len(val_df))
    - test_ds.csv: Held out completely until final evaluation to provide an unbiased assessment of the fine-tuned model's performance. Size: (Reported in script output as len(test_df))

- **Data Cleaning (Abbreviations):** A helper function split_title was implemented to handle job titles containing abbreviations in parentheses (e.g., "Certified Nursing Assistant (CNA)"). The get_clean_title function aimed to select the more descriptive version (usually the longer one) for consistency during training, although the implementation details might need review to confirm optimal handling in all cases.

3. Model Exploration and Selection

- **3.1. Sentence Transformer Models:**
    - **Rationale:** Sentence Transformers are designed specifically to produce high-quality sentence embeddings suitable for semantic search, clustering, and similarity tasks. Fine-tuning allows adapting a pre-trained model to a specific domain's vocabulary and semantic relationships.
    - **Candidates Evaluated:**
        - sentence-transformers/all-MiniLM-L6-v2: A distilled model (MiniLM architecture) with 6 layers, 384-dimensional embeddings, and a small footprint (~80MB). Optimized for speed and efficiency.
        - sentence-transformers/all-MiniLM-L12-v2: Similar MiniLM architecture but with 12 layers, still 384-dimensional embeddings, slightly larger (~120MB). Offers a balance between performance and efficiency, generally outperforming the L6 version.
        - sentence-transformers/all-mpnet-base-v2: Based on the MPNet architecture, combining techniques from BERT and XLNet. It has 12 layers but produces larger 768-dimensional embeddings and is significantly larger (~420MB). Typically provides state-of-the-art performance for general-purpose sentence embeddings among these choices.
    - **Selection Justification:** While all-MiniLM-L6-v2 and all-MiniLM-L12-v2 offered speed advantages, initial testing and the requirement for capturing fine-grained semantic distinctions in job titles led to the selection of sentence-transformers/all-mpnet-base-v2. Its larger embedding dimension (768) and more complex architecture (MPNet) were hypothesized to provide greater capacity for learning the specific nuances of the job title domain. This choice represents a trade-off, prioritizing embedding quality over computational speed and resource usage. The selection is reflected in fine-

tuning-build-project/fine_tuning/fine-tuning_jitter_dataset.py and fine-tuning-build-project/fine_tuning/visualise_embedding_space.py.

- **3.2. Large Language Model (LLM) APIs:**
    - **Rationale:** Modern LLMs possess vast world knowledge and language understanding capabilities. Using their APIs for zero-shot or few-shot similarity assessment was explored as an alternative or benchmark.
    - **APIs Tested:**
        - Deepseek r1
        - Azure OpenAI Service: o3-mini
        - Azure OpenAI Service: o1
        - Azure OpenAI Service: GPT 4.5
        - Anthropic Claude: Claude 3.7 Sonnet
        - OpenAI: GPT-4o
    - **Results & Observations:**
        - **Successes:** Azure OpenAI's GPT 4.5 delivered the most consistent and highest-quality results for the job title similarity task within this specific project context. GPT-4o also performed well, though perceived as slightly less effective than the Azure GPT 4.5 variant used. The final implementation leveraged the results or methodology associated with Azure OpenAI's GPT 4.5.
        - **Failures:** Surprisingly, models generally considered very powerful (o3-mini, o1, Claude 3.7 Sonnet) failed to produce usable or reliable outputs for this specific task. Despite attempting various prompting strategies, their outputs were inconsistent or did not accurately reflect the semantic similarity between job titles.
        - **Hypothesized Reasons for Failures:** This counter-intuitive result suggests several possibilities: (a) The models might be overly optimized for generative or conversational tasks rather than fine-grained semantic similarity scoring in a restricted domain. (b) They might be highly sensitive to prompt structure for this type of task, requiring more extensive prompt engineering than initially attempted. (c) The specific task might fall outside the optimal capabilities

emphasized during their training, even if they excel at broader language understanding. This highlights that general LLM capability doesn't always translate directly to superior performance on every specialized task.

4. Fine-Tuning Process

- **Methodology:** Triplet Loss Fine-Tuning
- **Loss Function:** losses.TripletLoss from the sentence-transformers library was employed. This loss function is designed for metric learning.
    - **Mechanism:** It considers triplets of examples: an "anchor" (a jittered title), a "positive" (the corresponding seed title), and a "negative" (a seed title from a different job). The loss function aims to minimize the distance between the anchor and positive embeddings while maximizing the distance between the anchor and negative embeddings, ensuring a margin (triplet_margin=0.3 set in the code) between positive and negative pairs. This effectively pulls embeddings of similar job titles closer together and pushes dissimilar ones further apart in the embedding space.
- **Training Data Generation:**
    - The generate_training_examples function in fine-tuning_jitter_dataset.py dynamically creates training instances. For each jittered title (anchor), it pairs it with its corresponding seed title (positive) and randomly samples a different seed title (negative) from the available pool. This online generation provides a large, diverse set of triplets for training.
- **Validation Data Generation:**
    - The generate_val_examples function creates a fixed set of validation triplets, pairing each anchor with its positive and multiple (5, in this case) negatives. This allows for consistent evaluation during training.
- **Training Orchestration:**
    - The SentenceTransformerTrainer class, integrated with SentenceTransformerTrainingArguments from the sentence-transformers library (leveraging the Hugging Face transformers backend), managed the training loop.

Varun Kadam
A20593606

- **Key Training Parameters (fine-tuning_jitter_dataset.py):**
    - num_epochs: 5
    - train_batch_size: 192
    - val_batch_size: 192
    - evals_per_epoch: 4 (Model evaluated 4 times per epoch)
    - warmup_ratio: 0.1 (Learning rate warmup)
    - eval_strategy: "steps" (Evaluate every eval_steps)
    - save_strategy: "steps" (Save checkpoint every save_steps)
    - save_total_limit: 2 (Keep only the best 2 checkpoints)
    - report_to: 'tensorboard' (Logging metrics)
- **Model Saving:**
    - Checkpoints were saved during training within fine_tuning/data/trained_models/.
    - Crucially, upon completion of training, the final fine-tuned model (base_model.save(...)) was explicitly saved to streamlit_app/data/fine_tuned_model/, making it directly accessible to the Streamlit demonstration application.

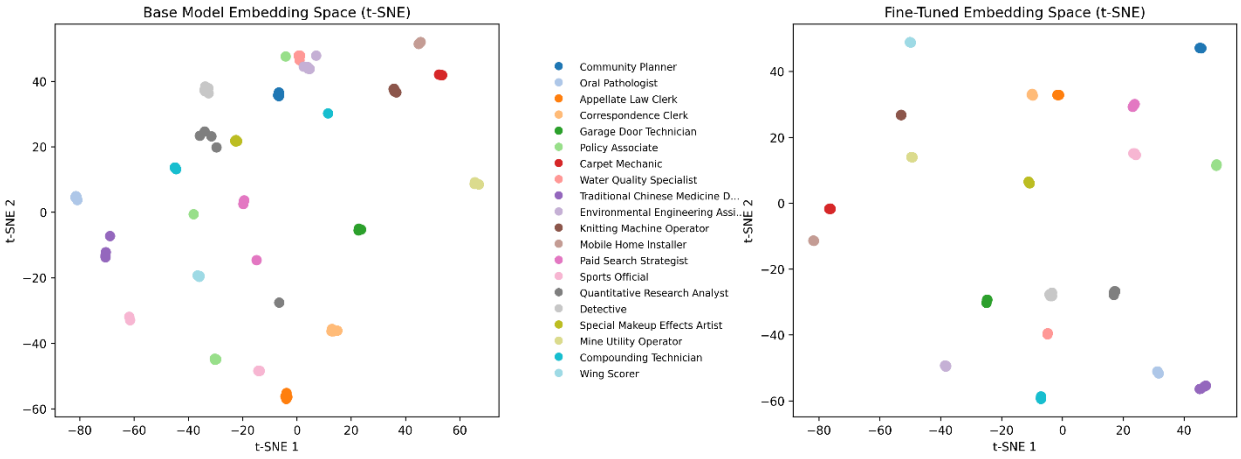
Varun Kadam
A20593606

5. Evaluation and Visualization

- **Evaluation Metric:** While triplet loss was used for training, the practical evaluation involved observing the quality of similarity scores and the clustering in the embedding space. The Streamlit app provides a qualitative comparison. Quantitative metrics like Mean Reciprocal Rank (MRR) or Recall could be calculated on the test set triplets for a more formal evaluation but were not explicitly mentioned in the provided scripts.
- **Visualization Technique:** t-Distributed Stochastic Neighbor Embedding (t-SNE) was used for dimensionality reduction.
    - **Purpose:** To visualize the high-dimensional (768D) embedding space in 2D, allowing for human interpretation of the clustering patterns.
    - **Implementation:** The visualise_embedding_space.py script performs this:
        1. Loads the test dataset (test_ds.csv).
        2. Loads both the base all-mpnet-base-v2 model and the fine-tuned model from streamlit_app/data/fine_tuned_model/.
        3. Generates embeddings for the test set's jittered titles using both models. Embeddings were generated in batches (batch_size=16 or adjusted based on GPU) for efficiency.
        4. Applies sklearn.manifold.TSNE (with n_components=2, random_state=101) independently to the base model embeddings and the fine-tuned model embeddings.
        5. Uses matplotlib to plot the resulting 2D points.
- **Visualization Results (embedding_visualization.png):**
    - The script generates a side-by-side comparison plot.
    - Points are colored based on their corresponding *seed title*, allowing assessment of whether variations of the same job title cluster together.
    - **Base Model Plot:** Expected to show some inherent clustering due to the pre-trained model's capabilities, but clusters might be diffuse or overlapping for closely related jobs.

Varun Kadam
A20593606

- **Fine-Tuned Model Plot:** Expected to show significantly tighter and more distinct clusters for each seed title. Jittered variations should be located very close to their corresponding seed title's cluster, and different seed titles should be more clearly separated. This visual evidence strongly supports the effectiveness of the fine-tuning process in creating a domain-specific embedding space. A subset of job titles (subset_size=20) was used for clarity in the final plot, with a dedicated legend.



6. Streamlit Application (streamlit_app/app.py)

- **Purpose:** To provide an interactive demonstration of the fine-tuned model's capabilities and compare its performance against the original base model.
- **Functionality:**
  1. **Resource Loading:** Uses @st.cache_resource to efficiently load models (load_fine_tuned_model, load_default_model), pre-computed embeddings for a subset of job postings (load_fine_tuned_embeddings, load_default_embeddings - loaded from .npy files, likely generated by prepare_embeddings.py), and job posting data (load_job_postings from job_postings.parquet). Limited to the first 5000 postings for demo purposes.
  2. **Device Detection:** Automatically detects and utilizes CUDA GPU or Apple MPS if available, falling back to CPU (get_device function).

3. **Search Interface:** Provides a text input (st.text_input) for users to enter a job title query.
4. **Embedding Generation:** Encodes the user's query using both the default and fine-tuned models (default_model.encode, fine_tuned_model.encode).
5. **Similarity Calculation:** Computes the cosine similarity (using torch.inner on normalized embeddings) between the query embedding and the pre-computed embeddings of the job postings database for both models.
6. **Results Display:** Shows the top 10 matching job postings side-by-side for the default and fine-tuned models, including the similarity score. This direct comparison highlights the difference in ranking and relevance.
7. **Similar Jobs Exploration:** Allows users to click a button ("Show most similar jobs") next to any result. This triggers a recalculation of similarity, using the selected job posting's embedding as the query against the rest of the database, again showing results for both models side-by-side.
8. **State Management:** Uses st.session_state to manage the application's flow (search view, results view, similar jobs view) and persist the user's search query.

- **Value:** The application serves as a powerful qualitative evaluation tool, making the impact of fine-tuning tangible by demonstrating improved search relevance and similarity matching in a user-friendly interface.

7. Challenges and Future Work

- **LLM API Reliability:** The unexpected failure of several high-profile LLM APIs for this specific task highlights the need for careful evaluation and suggests that general capabilities don't guarantee performance on specialized tasks. Relying solely on external APIs can introduce unpredictability.
- **Computational Resources:** Fine-tuning and inference with all-mpnet-base-v2 require more computational resources (GPU memory, processing time) compared to the MiniLM variants. Scaling this to extremely large datasets would require appropriate hardware infrastructure.
- **Data Quality and Coverage:** The effectiveness of fine-tuning heavily depends on the quality and representativeness of the training data (jittered_titles.csv). Gaps in

the dataset (missing job types, insufficient variations) could limit the model's generalization.

- **Hyperparameter Optimization:** While the chosen parameters (epochs, batch_size, margin) worked, more extensive hyperparameter tuning (e.g., using Optuna or Ray Tune) could potentially yield further improvements in embedding quality.
- **Alternative Loss Functions:** Exploring other metric learning losses (e.g., Contrastive Loss, Multi-Similarity Loss) might offer different trade-offs or better performance.
- **Larger/Different Base Models:** Experimenting with newer or larger sentence transformer models could provide better starting points for fine-tuning.
- **Test Set Evaluation Metrics:** Implementing quantitative metrics (MRR, Recall, Precision) on the held-out test set (test_ds.csv) would provide a more rigorous and objective measure of performance improvement.
- **Prompt Engineering for LLMs:** Further investigation into advanced prompt engineering techniques might eventually elicit better performance from the LLM APIs that initially failed.

8. Conclusion

This project successfully demonstrated the significant benefits of fine-tuning sentence transformer models for domain-specific semantic similarity tasks. By fine-tuning sentence-transformers/all-mpnet-base-v2 on a dataset of jittered job titles using triplet loss, a specialized embedding space was created that demonstrably outperforms the general-purpose base model in accurately capturing the nuances of job title similarity. The t-SNE visualizations provide clear visual confirmation of improved clustering, and the interactive Streamlit application showcases the practical improvements in search relevance.

The evaluation also yielded important insights into the applicability of general LLM APIs for this task, revealing that even powerful models like Claude 3.7 Sonnet or Azure's o1 can falter on specialized similarity tasks compared to a dedicated, fine-tuned embedding approach. Azure OpenAI's GPT 4.5 proved to be the most effective among the tested APIs for this specific use case.