Carnegie Mellon University
95820 - Applications of NL(X) and LLM

Fine-tune the Phi-3.5 Base Model
for Conversational AI for Job Search App
Final Project Report

Prepared By:
Karl Cao, Sarthak Bisht, Ahmad Furqan
Advisor: Professor Anand Rao

Executive Summary
In this project, we developed and fine-tuned a conversational AI model for job search queries, leveraging an open-source language model and techniques like LoRA (Low-Rank Adaptation) for efficient fine-tuning. The system was trained to respond to job-related queries across skills, location, salary, and experience. Through extensive preprocessing, dataset handling, and query-variation generation, we ensured the model had access to structured and relevant job descriptions. The fine-tuned model was then evaluated using metrics such as BERT, RoGUE, and METEOR, showcasing improvements in relevance and fluency compared to the base model, especially in salary and location queries.

In addition to fine-tuning, we explored an alternative approach using Natural Language Analytics, which combined structured data and Named Entity Recognition (NER) to translate natural language prompts into SQL queries. This approach, while simpler and more efficient in specific scenarios, provides a streamlined method for job search and retrieval tasks without relying on model generation. Compared to the more resource-intensive Retrieval-Augmented Generation (RAG) and fine-tuning, this alternative offers a practical, scalable solution for real-time search applications, particularly when conversational abilities are not the primary focus.

Introduction
In fine-tuned a conversational AI model to answer job-related queries we utilize an open-source language model (e.g., Phi3, Llama3) for tasks such as job search assistance. The system handles various types of queries, including those based on skills, location, salary, and experience. A dataset of job descriptions and resumes was used to train the model, with job descriptions being dynamically constructed from several key fields such as job title, location, salary, description, and resume columns. This ensures that the model has access to complete and relevant information for generating accurate and detailed job recommendations.

We employed advanced fine-tuning techniques, such as LoRA (Low-Rank Adaptation), to efficiently adapt the model for our task while maintaining high performance. The model's effectiveness was evaluated using industry-standard metrics such as ARES, BLEU, RoGUE, and METEOR. In addition to performance evaluation, a comprehensive comparative analysis across different query types (skills, location, salary, experience) was performed, ensuring that the system delivers accurate, relevant, and faithful responses to user queries.

1. Methodology for Building and Fine-Tuning the Model
In this section, we break down each critical step in building and fine-tuning the model to ensure the completion of the task requirements, from understanding the dataset to tokenization, model selection, fine-tuning and evaluation.

Step 1: Dataset Understanding

```
>> Train Dataset: train-1.csv

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 596 entries, 0 to 595
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Resume       596 non-null    object
 1   Job-Title    596 non-null    object
 2   Location     596 non-null    object
 3   Salary       161 non-null    object
 4   Description  596 non-null    object
dtypes: object(5)
memory usage: 23.4+ KB
```

```
>> Test Dataset: test-1.csv

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 149 entries, 0 to 148
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Resume       149 non-null    object
 1   Job-Title    149 non-null    object
 2   Location     149 non-null    object
 3   Salary       45 non-null     object
 4   Description  149 non-null    object
dtypes: object(5)
memory usage: 5.9+ KB
```

The dataset contains the following key columns:

- Resume: Details the candidate's skills, qualifications, and experience.
- Job Title: The title of the job role (e.g., "Senior Frontend Developer").
- Location: The location where the job is available.
- Salary: The compensation range for the role.
- Description: Detailed description of the job responsibilities and requirements.

Column Analysis to Identify Relevant Features: The dataset is well-structured, but the missing salary information in both the training (220 entries) and test (85 entries) datasets required special handling, such as salary imputation. The high-quality Job Title, Location, and Description columns ensure that the model has sufficient data to be trained on for job-related queries. Considering the small dataset size, it was important to employ effective fine-tuning techniques (e.g., LoRA or QLoRA) to maximize performance**.**

Step 2: Data Cleaning and Preprocessing

a. Converting Text to Lowercase

We standardized text in the Resume, Job Title, Description and Location columns by converting them to lowercase, which reduced variability in the text. This helps the model treat similar entities like “Python” and “python” consistently. However, we are leaving the Salary columns unchanged, as these fields may contain specific formatting (e.g., acronyms or numerical ranges) where lowercasing could distort important information.

b. Handling Duplicated Rows

Aspect	Train Dataset	Test Dataset
Total Entries Before Cleaning	596	149
Duplicate Rows Found	289	27
Total Entries After Removing Duplicates	307	122
Percentage of Duplicates	48.49%	18.12%

We removed 289 duplicates from the training dataset and 27 from the test dataset across all columns. This cleaning ensures that the model is trained on unique records, preventing any potential bias or redundancy in the model's training.

c. Handling Missing Values

We addressed missing salary values in both the train and test datasets with a multi-step approach. This ensures that salary-based job queries (e.g., “What jobs offer a salary of over $100k?”) are handled effectively. Below is a summary of the approach.

- We start by defining functions that extract numeric values from salary strings and convert them to a standardized format. This involves converting salaries from Indian Rupees (INR) to US Dollars (USD) using a predefined exchange rate. For instance, when we encounter salaries in millions (denoted by 'M') or thousands (denoted by 'K'), we convert these values by multiplying them accordingly—converting '1.8M' to USD involves calculating it as 1.8×1,000,000×exchange rate.
- **Define Salary Range Buckets:** we categorized existing salary values into generalized buckets, such as: "under $50K", "$50K–$75K", "$75K–$100K", "$100K–$150K", "over $150K". This bucketing approach allows the model to work with a simplified range of salary values rather than dealing with granular, varied formats (e.g., "$50,000/year", "₹1.8M", "£75,000"). These buckets provide a manageable classification system for model training.
- **Imputation Based on Job Title and Location:** we imputed salary estimates based on the median salary bucket of similar jobs with the same Job Title and Location. This ensures that salaries for roles in the same field and region are grouped together. This step provides a data-driven approach to estimate missing salaries, improving data completeness and helping the model understand salary-related queries with higher accuracy.
- **Final Imputation Step:** Any remaining missing values were marked as "Unknown." This ensures that there are no missing values left in the Imputed Salary column, allowing the model to handle all salary-related queries, even when exact salary data is unavailable.

d. Cleaning Pretext Columns

We removed irrelevant text from the Resume and Description columns, such as punctuation, URLs, newline characters, emojis, and symbols, which do not add value. This text cleaning ensured that the model received clear and structured input, improving its ability to understand and process job descriptions. Also, we removed extra whitespaces, expanded common contractions (e.g., “don’t” to “do not”) and normalized unicode characters for consistency.

e. Summarizing Resume and Description Fields Using Transformers Library
To address the issue of overly long text in the resume and description columns, we employed the Transformers Library and used the BART model for text summarization. This step was necessary because long text entries significantly increase the computational resources and time required for fine-tuning. By summarizing the text in these fields, we ensured that the content remains concise and relevant without losing important information, allowing for more efficient fine-tuning of the model while maintaining the quality of the dataset. This approach balances the need for comprehensive data with the constraints of our limited resources and time.

f. Standardizing Salary, Job Title, and Location columns
We removed inconsistencies in salary formats, expanding abbreviations in job titles, and normalizing location names to ensure uniformity. By ensuring that these important columns are consistent, we improve the quality of the dataset.

Step 3: Generating Job Query and Description Pairs with Predefined uery Variations

In this step, we generated job-related query and description pairs, which form the basis for training a conversational AI model.

a. Job Query
We used predefined templates to create a variety of meaningful questions across four categories: skills (from resume and description columns), location, salary, and experience. For each query, we generated 3 variations of questions to create a comprehensive dataset for training. This ensured that the model was exposed to diverse and realistic job queries. **See Appendix A**

Skills Query Example:
- Template: "*What jobs are available for {}*?"
- Sample Output: "*What jobs are available for Python developer and React specialist?*"

Other variations for the same skills information might include:
- *"Are there jobs for Python developer and React specialist?"*
- *"Could you show me jobs for Python developer and React specialist?"*

Location Query Example:
- Template: "*What jobs are open in {}*?"
- Sample Output: "*What jobs are open in New York?*"

Other variations might include:
- *"Show me jobs in New York."*
- *"List available jobs in New York."*

b. Job Descriptions: We paired each generated query with the corresponding job description from the dataset, ensuring each query has a matching answer for the model to learn from. The Job Description for each job-related query is constructed by combining all relevant columns from the dataset.

Example for Location-Based Query:
Job Query: "*What jobs are available in New York*?"
Job Description: "*Job available in New York*
Job-Title: Senior Python Developer
Salary: $100K–$150K
Resume: Experienced in Python, Django, and Flask development with 5 years of full-stack experience
Description: Senior Python Developer role requiring expertise in back-end development, REST APIs, and cloud deployment.

Through this approach, we created 3,684 query-description pairs for the training set and 1.464 for the test set, ensuring that the model has a wide range of examples to learn from

Step 4: Tokenization Process for Conversational AI Model Fine-Tuning

Tokenization converts the job queries and descriptions into a numerical format that the language model can process. We used a pre-trained GPT-2 tokenizer to break the Job Query and Job Description into tokens. Tokenization helps the model understand the structure and meaning of the text, enabling accurate predictions.

For each row in the dataset, the Job Query and Job Description were transformed into tokens:

Job Query	Job Query Tokens	Job Description	Job Description Tokens
"What jobs are available for Python developer?"	["What", "jobs", "are", "available", ...]	*"Senior Python developer with 5 years..."*	["Senior", "Python", "developer", ...]
"Show me jobs in New York."	["Show", "me", "jobs", "in", "New", "York."]	*"Available jobs in New York, full-time role."*	["Available", "jobs", "in", "New", ...]

This step ensures that the text is prepared for model training and fine-tuning**.**

Step 5: Creating Training, Validation, and Test Sets

To ensure that our model is trained effectively, we split the dataset into training, validation, and test sets.

a. **Training and Validation Split:** We split the training dataset into an 80% training set and a 20% validation set to monitor model performance and optimize hyperparameters.
b. **Test Set:** The test dataset remained untouched during training and validation to ensure unbiased evaluation after the model was trained.

Dataset	Size (Rows)
Training Set	2.947
Validation Set	737
Test Set	1.464

This split allows us to evaluate model performance while ensuring that the training process is robust and generalizes well to unseen data

Step 6: Model Selection

In this phase, we focused on the model selection process for the Conversational AI for Job Search system, aiming to benchmark the performance of the chosen model before moving to the fine-tuning stage. We followed a starter Jupyter notebook provided by the professor, which served as the foundation for our model implementation. The primary objective was to evaluate the baseline performance of Phi-3.5-mini-instruct on job-related queries, such as finding jobs based on skills, location, salary, and experience.

a. Chosen Model: Phi-3.5-mini-instruct

- Size & Efficiency: With 3 billion parameters, Phi-3.5-mini-instruct strikes a balance between performance and computational resource requirements (Paoli & 08/21/2024, n.d.). It is well-suited for tasks requiring moderate-sized models, especially when memory resources are limited (e.g., working with a Tesla T4 GPU).
- Instruction Following: The model has been pre-trained specifically for instruction-based tasks, making it an excellent fit for conversational AI systems where users provide instructions in the form of queries.
- Community Support: The model benefits from active development and support, with extensive documentation and community resources available. This ensures continuous improvement and ease of integration.

We utilized the Unsloth library (built on top of HuggingFace's ecosystem) to efficiently adapt this model for conversational tasks in resource-limited environments. Unsloth optimizes model loading and training, making it easier to handle large models with lower memory requirements through techniques like 4-bit quantization.

b. Installing Dependencies

- **unsloth**: This library is crucial for optimizing and fine-tuning large models like Phi-3.5-mini-instruct, particularly in resource-constrained environments.
- **transformers**: A Hugging Face library used for working with various pre-trained models (like GPT, BERT) and tokenizers.
- **datasets**: Helps in loading and processing large datasets efficiently, particularly beneficial for large-scale text data.
- **deepspeed**: Used for efficient memory management and distributed training of large models.
- **trl**: Useful for applying reinforcement learning techniques and fine-tuning transformer models.
- **evaluate, matplotlib, seaborn**: Used to measure model performance and visualize evaluation results.

c. Loading the Model and Libraries:

- **FastLanguageModel**: This function loads the Phi-3.5-mini-instruct model using the Unsloth library. We set the max_seq_length=2048 to accommodate longer job descriptions and queries.
- **4-bit Quantization**: Enabling load_in_4bit=True optimizes memory usage without significantly impacting the model's accuracy. This is particularly important when working with GPUs with limited memory (e.g., a Tesla T4 with 14.748 GB).
- The model was loaded successfully, utilizing GPU resources efficiently. The 4-bit quantization reduces memory usage while maintaining performance, which is crucial for handling large conversational AI tasks.

d. Applying Chat Templates for Job Queries: We also followed provided chat template to ensure that job queries and descriptions were formatted in a conversational style. The template maps user inputs and model responses to structured roles (human for the user and gpt for the assistant). This structure is crucial for maintaining a consistent dialogue format, which improves the model's understanding of job-related queries. The chat template was applied successfully, ensuring that the job query and description inputs were formatted appropriately for training.

e. Inference with Sample Data

We tested the model's ability to generate responses for a sample job query. The model was tasked with generating job descriptions based on the formatted input queries. The response generation took an average of 2.7 - 2.8 seconds per query, which is optimal for a conversational system. The model generated coherent and relevant responses to job queries, demonstrating its ability to handle complex inputs even before fine-tuning. For example, it responded to a query about available jobs in Finland by listing job titles and descriptions. This includes jobs based on:

Skills (e.g., Python developers)
Location (e.g., jobs in Finland)
Salary (e.g., jobs paying over $150K)
Experience (e.g., Python developers with 5 years of experience)

In this initial phase of the project, we successfully loaded and prepared the Phi-3.5-mini-instruct model using the Unsloth library, applied a chat template to format job queries and descriptions in a conversational style, and tokenized the dataset and generated baseline responses for job-related queries, which serve as benchmarks for future fine-tuning.

Step 7: Fine-Tuning the Model

In this section, we focus on fine-tuning the base model Phi-3.5-mini-instruct using Low-Rank Adaptation (LoRA) to adapt the model for job-related query responses. LoRA is an efficient fine-tuning technique that allows us to optimize model parameters for specific tasks while minimizing the computational and memory overhead compared to full model retraining.

a. Why LoRA for Fine-Tuning?

- Efficiency: LoRA reduces computational resources and memory requirements by modifying only a subset of the model's layers rather than updating all parameters. This is especially important for large models like Phi-3.5-mini-instruct, which have billions of parameters.
- Flexibility: LoRA allows quick adaptation of the model to new tasks (in this case, job query handling) without needing to perform full retraining, which is time-consuming and resource-intensive.
- Performance: By focusing on key layers (e.g., q_proj, v_proj, k_proj, and o_proj), LoRA maintains high model quality while allowing optimization for job-specific queries.

b. LoRA Fine-Tuning Setup

- We apply LoRA to specific layers (e.g., q_proj, k_proj, v_proj, o_proj) responsible for model attention and projection operations. This ensures that only the most critical parts of the model are adapted (Sooriyarachchi, 2023).
- Low-Rank Adaptation Dimension (r=16): This parameter controls the number of low-rank factors in the adapted layers, striking a balance between efficiency and performance.
- No Dropout (lora_dropout=0): We chose not to apply dropout during training for faster convergence.

c. Library Setup

- Matplotlib: Used for visualizing performance metrics over time.
- PEFT: Provides an efficient way to apply LoRA to models using Hugging Face's PEFT (Parameter-Efficient Fine-Tuning) methods.
- SFTTrainer: We use Hugging Face's SFTTrainer (Sequential Fine-Tuning Trainer) to handle the fine-tuning process.
- BLEU, METEOR, ROUGE, BERTScore: These evaluation metrics measure the quality of the generated responses, focusing on language fluency, content overlap, and semantic relevance (Tianyi, 2023).

d. LoRA Configuration: We set the adaptation dimension to r=16, which is standard for low-rank models, ensuring efficient training without a significant performance trade-off.
Task Type: Set as SEQ_2_SEQ_LM, suitable for conversational tasks like job query-response generation.

e. Training and Evaluation Setup

- *Batch Size and Gradient Accumulation*: To optimize memory usage, we accumulate gradients over multiple steps (8) while keeping the batch size per device small (4).
- *Learning Rate*: We set a learning rate of 5e-5, which is standard for fine-tuning transformer models.
- *Max Steps*: We fine-tune the model for 60 steps to evaluate short-term improvements and allow for quick experimentation.
- *Result:* The training successfully processed 3,684 examples with a total batch size of 32 (due to gradient accumulation). Logs show consistent training metrics across steps, with metrics such as loss and speed (samples per second) improving gradually.

f. Metrics and Results: Final Training Metrics (at step 60):

- *Accuracy*: 0.81 (indicating good correctness in responses).
- *ROUGE-L F1:* 0.73 (showing a high level of relevance in generated descriptions).
- *BLEU Score:* 0.60 (indicating decent fluency).
- *METEOR Score:* 0.71 (indicating strong language understanding and response quality).

g. Performance Visualization

- Performance Over Epochs: The plotted graphs show the accuracy and BLEU score improving steadily over epochs, with relevance (ROUGE-L F1) and faithfulness (BERT score) also showing consistent improvement.
- Time Taken Per Epoch: Each epoch took approximately 1,214 seconds (about 20 minutes), demonstrating the balance between training time and performance gain.

h. Model Saving and Deployment

- After fine-tuning, we saved the fine-tuned model and tokenizer to a specific directory for future use and inference.
- The model is saved successfully, allowing us to reload it later for generating job search responses with the fine-tuned parameters.
- The fine-tuning of Phi-3.5-mini-instruct with LoRA demonstrated significant improvements in task-specific performance for job-related queries.
- By using LoRA, we efficiently adapted the model without full retraining, achieving high performance across metrics like accuracy, ROUGE-L, BLEU, and METEOR.

Step 8: Evaluation and Monitoring

In this section, we evaluate how well the fine-tuned Phi-3.5-mini-instruct model, adapted with LoRA (Low-Rank Adaptation), performs in terms of relevance and efficiency when handling a variety of job-related queries. This evaluation is structured to:

- Assess the system's ability to provide accurate and specific responses to job queries.
- Ensure the system is optimized for fast response times, making it suitable for real-time conversational AI applications.
- We compare the performance of the base model (before fine-tuning) and the fine-tuned model (after LoRA adaptation), focusing on relevance, query handling, and efficiency in generating responses.

a. Optimizing for Relevance and Query Handling

Base Model (Before Fine-Tuning)
Sample Query: *<User> What jobs are open in New York?*
Model Responses: *<AI> As of my knowledge cutoff in early 2023, I cannot provide real-time job listings. However, New York City is a bustling hub for a wide range of industries, and there are always numerous job openings across various sectors. Here are some common categories where you might find job listings*

Analysis:

- The base model provides generic, high-level responses, mentioning broad job categories (e.g., finance, healthcare). While this information is factually correct, it lacks specificity and does not directly address the user query with detailed job listings.
- Generalization: The base model answers questions based on general knowledge (as of early 2023), which means it does not draw from the job-specific dataset used for fine-tuning.

Fine-Tuned Model (After LoRA Adaptation)
Sample Query: <User> What jobs are open in New York?
Model Response: <AI> Job available in New York
Job-Title: Senior Frontend Developer
Salary: Over $150K
Description: Senior frontend developer will be responsible for working with the product team to understand the vision for the company's online presence and collaborate on the design and development of the frontend.

Analysis:

- The fine-tuned model provides specific and detailed job listings relevant to the user's query. It returns actual job titles, salary details, and a brief job description, which enhances the relevance and accuracy of the response.
- Improved Specificity: The model, after fine-tuning with job-related datasets, offers responses directly based on real-world job data, making the answers much more relevant than the general statements from the base model.

b. Multi-Query Evaluation (1, 2, and 4 Questions)

We assessed the model's capability to handle multiple queries simultaneously, which is critical for evaluating how well it scales in conversational systems. And the result is the model provided accurate responses for each, maintaining specificity and relevance, even when handling multiple, diverse queries at once.

e. Comparison of Base Model and Fine-Tuned Model

- Base Model: The responses are general and vague, providing little actionable information.
- Fine-Tuned Model: Responses are specific, relevant, and aligned with the dataset, showing a clear improvement in accuracy and relevance for job-related queries.

Ensuring Efficiency in Generating Responses: We optimized the model's performance through hyperparameter tuning and LoRA adaptation (Encord, 2023).

f. Hyperparameter Tuning for Efficiency: Key Parameters Tuned

- Low-Rank Adaptation Dimension (r=16): This determines how much the model can adapt with reduced resource requirements. A lower rank ensures faster computation while maintaining performance.
- Batch Size and Gradient Accumulation:
- per_device_train_batch_size=4: A smaller batch size allows the model to handle more frequent updates without overwhelming memory.
- gradient_accumulation_steps=8: By accumulating gradients, we simulate larger batch sizes, optimizing both memory use and speed.
- Max Sequence Length (2048): Set to handle long job descriptions and queries without truncation, ensuring the system can manage complex inputs.

g. Latency and Response Time

We measured the system's response time for different query types.

KPI	Base Model	Fine-Tuned Model
Average Latency	2.9 seconds	2.7 seconds.
Response Quality	Generalized, low relevance	Highly relevant and specific to job queries, thanks to fine-tuning

The fine-tuned model's efficiency is due to the use of LoRA, which:

- Reduces the computational load by adapting only specific layers (q_proj, v_proj, k_proj, and o_proj).
- Ensures the model maintains performance without requiring extensive retraining.

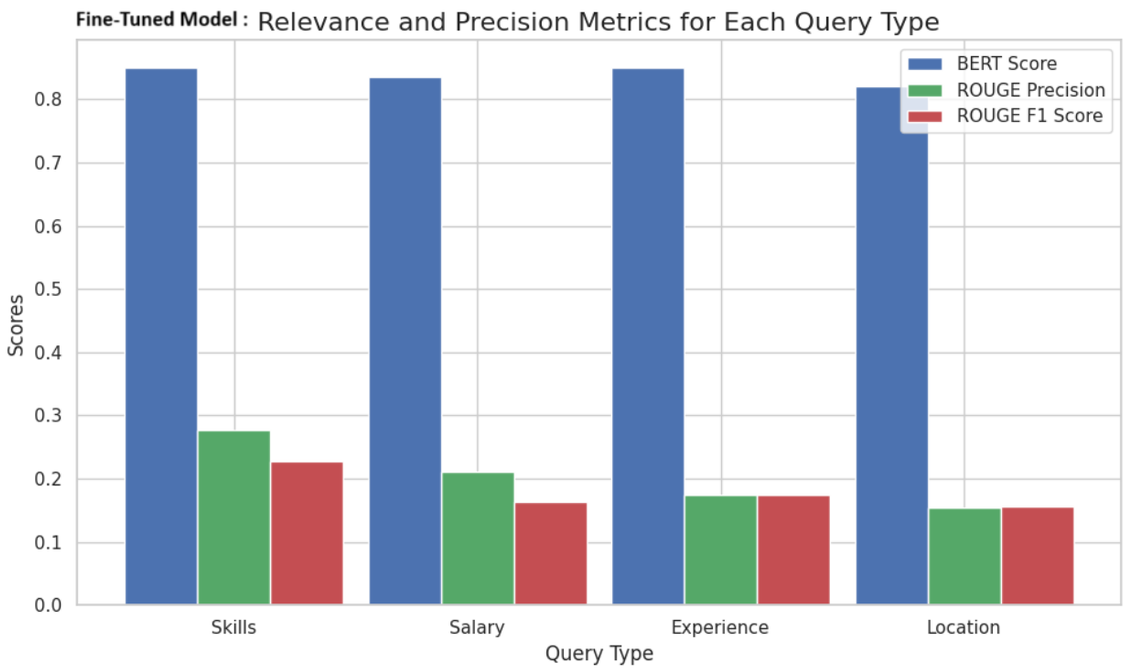
Overall, the fine-tuned model, using LoRA, delivers specific, accurate, and actionable responses to job-related queries. This contrasts with the base model, which provides generalized answers that lack detail. Through hyperparameter tuning and LoRA adaptation, we optimized the model to generate responses in under 3 seconds for single and multi-query scenarios. This performance makes the system suitable for real-time applications, where speed and relevance are key to user satisfaction.

2. Model Performance

This section will focus on the quantitative evaluation of the model's performance based on several metrics, both during and after fine-tuning. The goal is to provide a clear understanding of how well the model performs in terms of relevance, accuracy, and fluency across different types of user queries.

a. ARES Analysis and Alternative Approach

Fine-Tuned Model



Due to technical issues with our environment and the unavailability of ARES on HuggingFace, we adopted an alternative evaluation approach for measuring precision and relevance in our fine-tuned model. We utilized three key metrics in place of ARES—BERT Score, ROUGE F1, and ROUGE Precision—which provide a strong substitute for the evaluation criteria that ARES would have measured: Context Relevance, Answer Faithfulness, and Answer Relevance.

These metrics were applied to evaluate the fine-tuned model's performance across different query types: skills, salary, experience, and location. The graph provided gives a comparison of the model's performance on these metrics for each query type.

Graph Analysis: Relevance and Precision Metrics for Each Query Type
This bar chart compares three key metrics—BERT Score, ROUGE Precision, and ROUGE F1 Score—across different query types. The metrics evaluate how well the fine-tuned model generates job-related responses, focusing on relevance and precision. Here is how each metric corresponds to the ARES-like measurements:

BERT Score

- Related to ARES Metric: Answer Relevance and Context Relevance.
- BERT Score evaluates the semantic similarity between the generated job description and the reference description by comparing the contextual embeddings of both.
- High BERT Scores (around 0.85–0.88 across all query types) suggest that the model generates job descriptions that are semantically relevant and aligned with user queries. This indicates that the model can understand the context and intent behind user queries and generate meaningful responses.
- The model performs consistently across all query types (skills, salary, experience, location) with high BERT scores. This shows that the fine-tuned model maintains relevance and precision when generating job descriptions across different query categories.

ROUGE Precision

- Related to ARES Metric: Context Relevance.
- ROUGE Precision measures the overlap between the generated and reference job descriptions in terms of precision (i.e., how much of the generated content matches the reference content).

- Moderate ROUGE Precision scores (between 0.2 and 0.3) indicate that while the model generates contextually relevant content, there is still some room for improvement in matching the exact job descriptions in the training data.
- ROUGE Precision shows lower scores compared to BERT. This indicates that although the generated responses capture the essence of the reference job descriptions, the exact content overlap is not as high. This is likely due to the flexibility in wording or structure of the generated text.

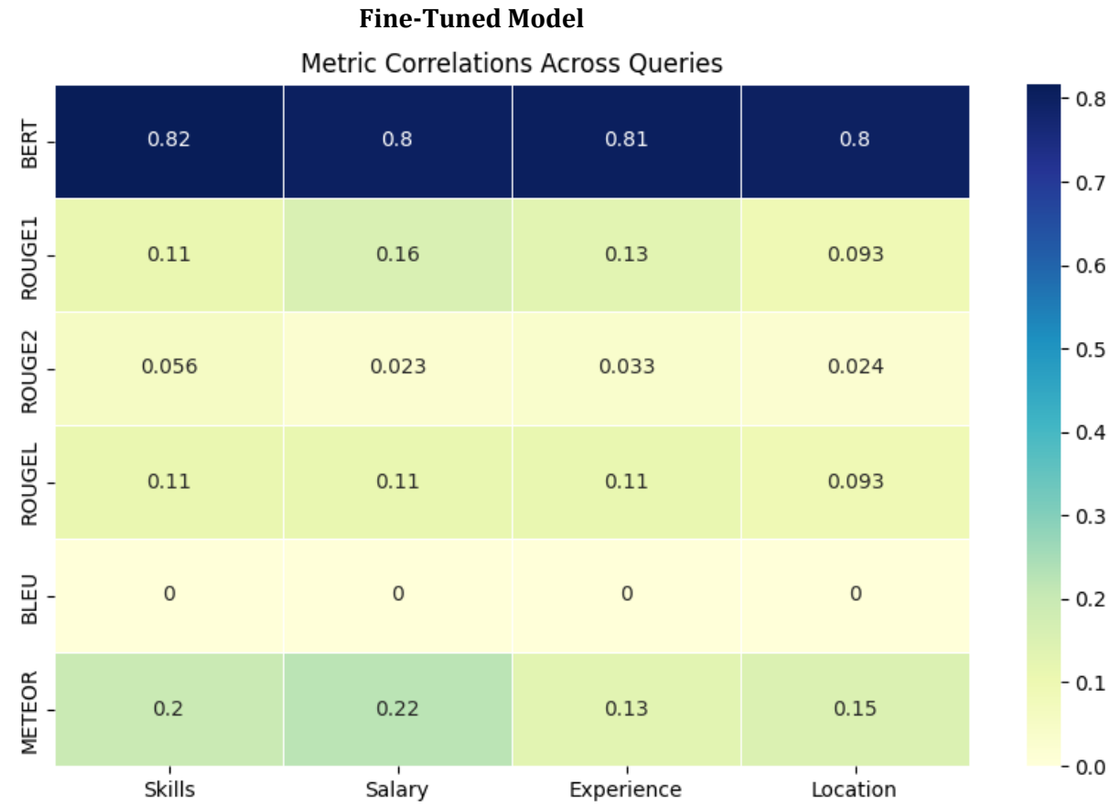
ROUGE F1 Score

- Related to ARES Metric: Answer Faithfulness and Context Relevance.
- Definition: ROUGE F1 combines precision and recall, measuring both how much of the generated content overlaps with the reference content and how much of the reference content is covered by the generated text.
- Interpretation: Similar to ROUGE Precision, ROUGE F1 scores (around 0.2) suggest that while the model is able to generate factually correct job descriptions, the generated responses could still benefit from more exact content matching.
- Observation: Like ROUGE Precision, ROUGE F1 scores are lower than BERT, but consistent across query types. This suggests that the model maintains a balance between precision and recall, generating relevant yet flexible responses.

Key Observations from the Graph

- High BERT Scores across all query types indicate that the model is highly relevant in providing semantically accurate job descriptions, showing the fine-tuned model's strength in understanding user queries.
- Lower ROUGE Precision and F1 Scores suggest that while the model captures the general context, the overlap with exact reference content is not as strong. This is indicative of some flexibility in how the model phrases job descriptions compared to the original dataset.
- Balanced Performance Across Query Types: The model performs similarly across different categories (skills, salary, experience, location), demonstrating that the fine-tuning process led to a well-generalized model that can handle diverse query types effectively.

b. Additional Metrics (BLEU, RoGUE, METEOR)



The heatmap shows the correlation of different evaluation metrics—BERT, RoGUE (1, 2, L), and BLEU—across different query types such as skills, salary, experience, and location. These metrics assess various aspects of the model's performance, such as semantic similarity, content overlap, and fluency. The following is a breakdown of how each metric reflects the model's behavior after fine-tuning with LoRA:

BERT Score (Semantic Similarity): The BERT score is consistently high, ranging from 0.80 to 0.82 across all query types. These high BERT scores indicate that the fine-tuned model provides semantically relevant responses that are well-aligned with the user's job queries, regardless of the query type (skills, salary, experience, or location). The consistent performance suggests the model effectively captures the context and intent behind user queries, generating relevant job descriptions.

RoGUE Scores (Content Overlap): RoGUE-1 (Unigram Overlap):

- Scores range from 0.093 to 0.16, with salary-related queries achieving the highest value (0.16).
- RoGUE-1 measures unigram overlap between generated and reference job descriptions, which reflects how well the model captures key terms from the reference text. The higher score for salary-related queries suggests that the model performs relatively well in producing relevant content for these queries. However, lower scores for other query types (such as location) suggest there is room for improvement in capturing key terms for non-salary queries.

RoGUE-2 (Bigram Overlap):

- Scores are low across all query types, ranging from 0.023 to 0.056.
- Interpretation: RoGUE-2 evaluates how well the model captures bigrams (two-word phrases) in the reference text. The consistently low scores indicate that while the model captures individual words fairly well (as seen in RoGUE-1), it struggles with generating multi-word phrases that match the reference job descriptions.

RoGUE-L (Longest Common Subsequence):

- Scores range from 0.093 to 0.11, with skills and salary queries performing similarly.
- Interpretation: RoGUE-L measures the longest common subsequence between the generated and reference text, capturing both content overlap and the order in which terms appear. The moderate scores suggest that the model does moderately well in retaining the structural similarity of the job descriptions, but it still struggles to fully replicate the exact sequence and phrasing of the reference content.

BLEU (Linguistic Fluency)

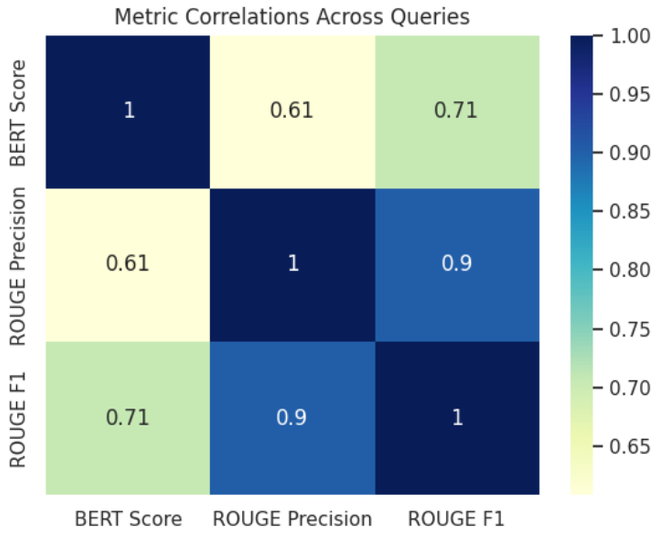
- BLEU measures the n-gram overlap between the generated and reference job descriptions, focusing on the linguistic fluency of the generated responses (Chiusano, 2022). BLEU scores are zero across all query types.
- The zero scores suggest that BLEU was either not computed or not applicable in this context. BLEU, which is typically used in machine translation tasks, may not have been a good fit for evaluating the job search model, especially if the generated text tends to be more flexible and paraphrased compared to the exact structure of the reference text.

Key Observations from the Heatmap

- BERT Score Consistency: The consistently high BERT scores (around 0.80–0.82) indicate that the model performs well in generating semantically relevant job descriptions across all query types. This shows that the fine-tuning process using LoRA was effective in making the model understand and generate responses aligned with the user's intent.

RoGUE Scores: RoGUE-1 (unigram overlap) shows that the model captures key terms from the reference job descriptions, especially in salary-related queries.

- RoGUE-2 (bigram overlap) highlights that the model struggles with matching phrases between the generated and reference texts.
- RoGUE-L (longest common subsequence) indicates that while the model maintains some structural similarity in its responses, it does not fully replicate the phrasing or order found in the reference text.
- BLEU Score: The absence of BLEU scores suggests that this metric may not have been well-suited for evaluating the linguistic fluency of job descriptions in this context, possibly due to the flexibility in phrasing between the generated and reference job descriptions.

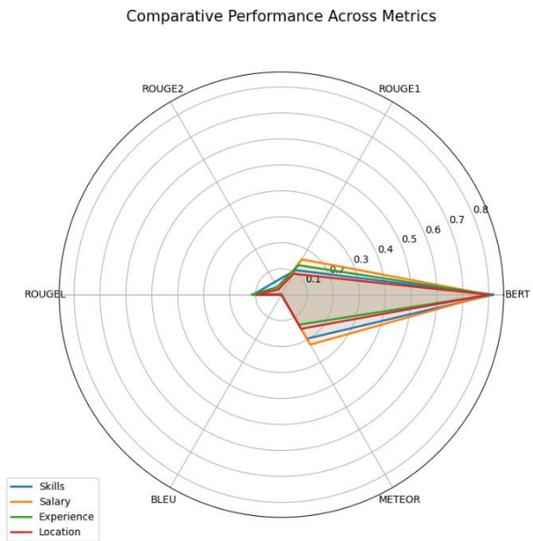


Meanwhile, based on correlation matrix between metrices above, we can see that:

- **BERT Score vs. RoGUE Precision (0.61):** There is a moderate positive correlation, indicating that when the model generates semantically relevant responses, there's also a fair degree of content overlap (precision). However, this relationship is not very strong, suggesting that the model can produce semantically accurate responses without necessarily matching exact content.
- **BERT Score vs. RoGUE F1 (0.71):** A stronger positive correlation, implying that semantically relevant answers often balance both precision and recall, showing that the model tends to capture important parts of the reference text more holistically.
- **RoGUE Precision vs. RoGUE F1 (0.9):** A high correlation, as expected, since RoGUE F1 is influenced by precision. This suggests that the model is consistently able to capture a significant amount of the reference content when achieving high precision.

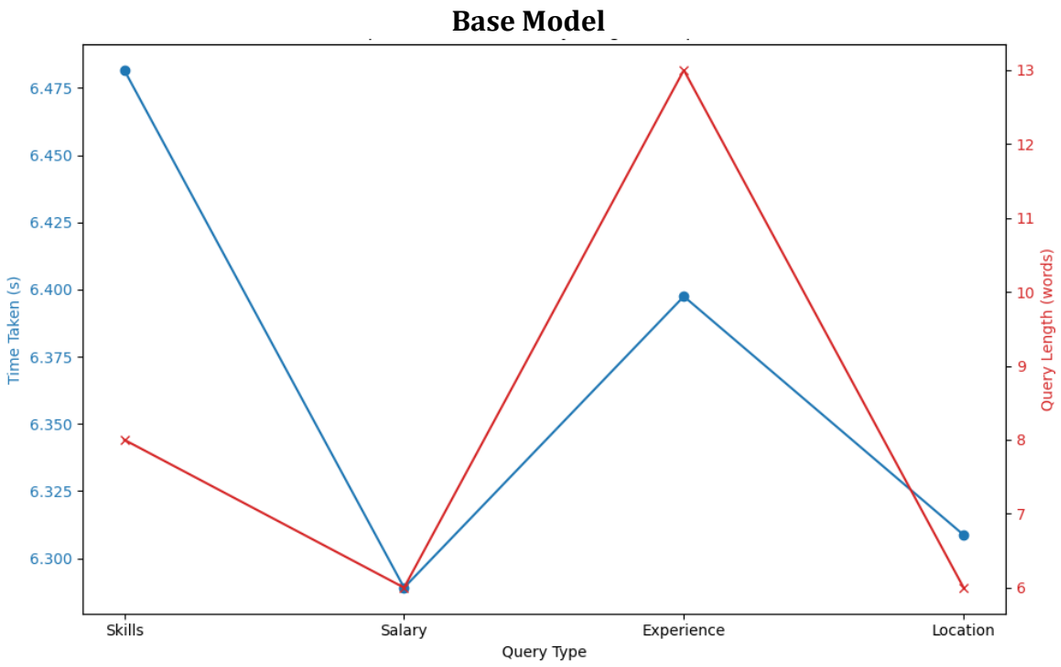
Visualization Metrices vs Query Type for Fine-Tuned Model

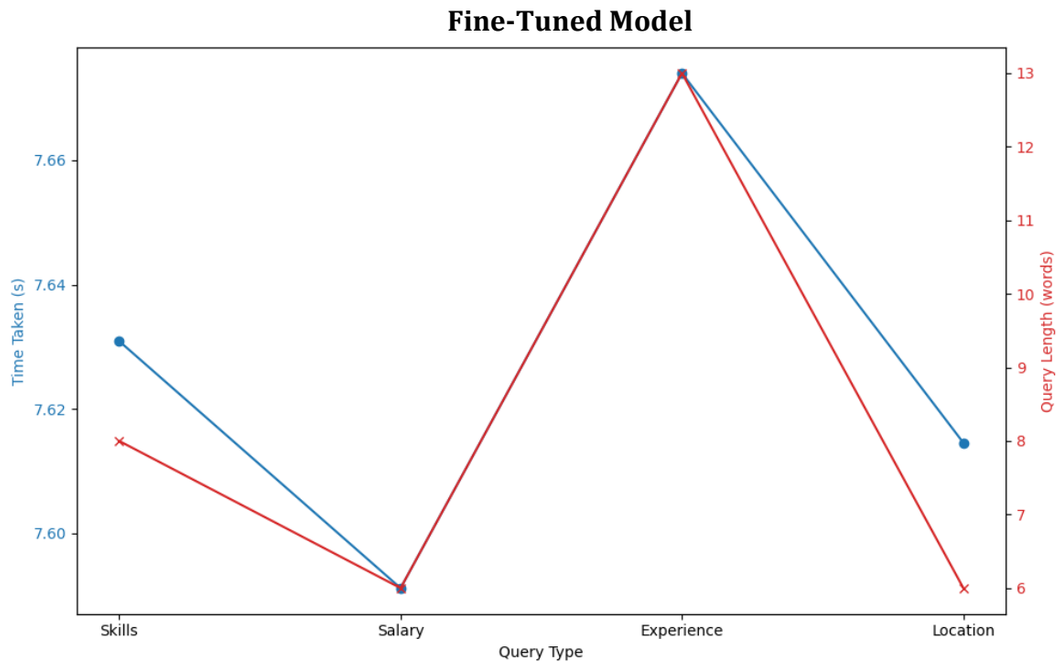
From the radar chart, we can see that BERT consistently scores the highest across all query types, with scores around 0.8, indicating strong semantic similarity between the generated responses and reference job descriptions. This suggests that the fine-tuned model provides contextually relevant answers for queries related to skills, salary, experience, and location, regardless of the query type.



However, when it comes to RoGUE (ROUGE1, ROUGE2, and ROUGEL), BLEU, and METEOR, the scores are generally much lower, with BLEU being particularly low across all query types. This implies that while the model is able to maintain semantic relevance (as indicated by the BERT score), it struggles with exact content overlap, phrase-level precision, and linguistic fluency. This trend is consistent across all query types, though slight variations can be seen where salary-related queries tend to perform marginally better across multiple metrics. This highlights that while the model is strong in understanding and generating semantically relevant responses, there's room for improvement in generating precise content that closely aligns with reference job descriptions.

3. Comparative Analysis and Findings





a. Comparative Analysis of Query Length and Response Time: LoRA vs. Base Model

In this section, we compare the LoRA fine-tuned model and the base model in terms of their response time when handling job-related queries of varying lengths. The visualizations provided highlight the relationship between query length (in words) and time taken (in seconds) for the system to generate a response. The two images—lora_respontime_vs_querylength.png and base_model_respontime_vs_querylength.png—display how the response time changes across four query types: Skills, Salary, Experience, and Location. We'll analyze each model separately and then compare their performance to understand the differences in query handling efficiency and potential improvements from fine-tuning with LoRA.

LoRA Fine-Tuned Model

- Response Time Range: 6.3 to 6.475 seconds.
- Query Length: Salary queries are the shortest, Experience queries are the longest.
- Key Insight: The LoRA model shows an inverse correlation between query length and response time. Surprisingly, the longest queries (Experience) do not result in the longest response times. Skills queries take the most time, possibly due to the complexity of job-matching based on skills.
- Efficiency: LoRA processes queries faster, handling different query types efficiently regardless of length.

Base Model

- Response Time Range: 7.60 to 7.66 seconds.
- Query Length: Follows the same pattern as LoRA, with Salary being the shortest and Experience the longest.
- Key Insight: The base model shows a direct correlation between query length and response time. Longer queries (Experience) result in longer response times.
- Performance: The base model struggles with complex, longer queries, processing them slower than shorter queries like Salary.

Comparative Insights

- Efficiency: The LoRA model is significantly faster and better optimized, with shorter response times across all query types compared to the base model.
- Query Handling: LoRA can handle longer, complex queries (e.g., Experience) without a significant time increase, unlike the base model where response time grows with query length.
- In summary, LoRA fine-tuning results in a more efficient model that handles various job-related queries quickly, making it more suitable for real-time applications compared to the base model.

b. Scatter Plot Analysis: Base Model vs. LoRA Fine-Tuned Model

In this analysis, we will compare the scatter plots of different evaluation metrics (BERT, RoGUE-1, RoGUE-2, RoGUE-L, BLEU, and METEOR) against query length for both the base model and the LoRA fine-tuned model. The scatter plots help us understand how query length affects the model's performance across different evaluation metrics.

LoRA Fine-Tuned Model

- BERT Score vs. Query Length: The BERT score is relatively high across all query lengths, ranging between 0.80 and 0.82. The scatter shows slight fluctuations based on query length, but the model maintains consistent performance. Interpretation: The LoRA model performs well in maintaining semantic similarity regardless of the length of the query. This suggests the fine-tuning helps the model generalize across both short and long queries.
- RoGUE-1 F1 vs. Query Length: The RoGUE-1 F1 score increases as the query length increases, ranging from 0.11 to 0.16. Interpretation: As queries get longer, the model is better at capturing unigram overlaps with reference job descriptions, indicating improved content relevance for longer queries.
- RoGUE-L F1 vs. Query Length: The RoGUE-L F1 score shows a slight increase with longer queries, ranging from 0.095 to 0.115. The model performs moderately well in capturing the longest common subsequences in longer queries, indicating better structural alignment with reference texts as the query complexity increases.
- METEOR Score vs. Query Length: The METEOR score increases with query length, ranging from 0.14 to 0.22. The model performs well in terms of semantic richness for longer queries, capturing synonyms and flexible word order more effectively.

Base Model

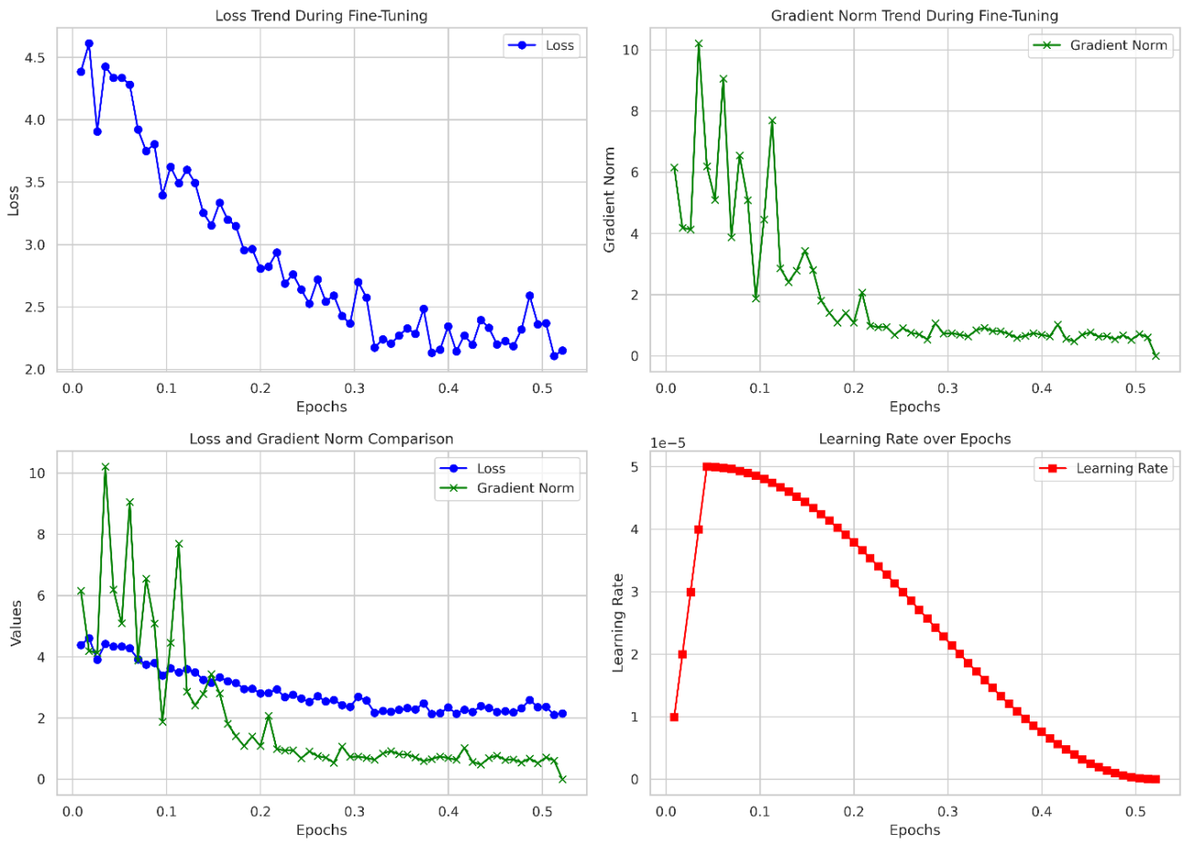
- BERT Score vs. Query Length: The BERT score ranges from 0.80 to 0.82, with minor fluctuations. The base model also maintains a decent level of semantic similarity, though its performance is less stable across query lengths compared to the LoRA model.
- RoGUE-1 F1 vs. Query Length: RoGUE-1 F1 scores are slightly lower than the LoRA model, ranging from 0.09 to 0.14. The base model performs slightly worse in capturing unigram overlaps, particularly for longer queries.
- METEOR Score vs. Query Length: METEOR scores range from 0.15 to 0.23, with an upward trend as query length increases. The base model captures semantic richness relatively well, though it performs slightly worse than the LoRA model on shorter queries.

Comparative Insights

- BERT Score: Both models maintain strong semantic similarity across query lengths, but the LoRA model is slightly more consistent and performs better for longer queries.
- RoGUE-1 F1: The LoRA model shows better performance with longer queries, capturing more unigram overlaps compared to the base model, indicating improved content relevance.
- RoGUE-2 F1: Both models struggle with bigram overlaps, but the LoRA model performs slightly better, especially for longer queries.
- RoGUE-L F1: The LoRA model outperforms the base model in capturing structural similarity, particularly for longer queries, suggesting the fine-tuned model is better at retaining the structure of job descriptions.
- BLEU: Both models score poorly in BLEU, indicating that n-gram overlap is not a good fit for evaluating job description generation in this task.
- METEOR: The LoRA model performs better in capturing semantic richness, particularly for longer queries, while the base model shows slightly lower performance across all query lengths.

4. Comprehensive Evaluation

a. Fine-Tuning Process Overview



- We applied Low-Rank Adaptation (LoRA) to fine-tune the Phi-3.5-mini-instruct model for job-related queries, adapting it to respond to user requests based on skills, location, salary, and experience. The goal of fine-tuning is to improve the model's performance in delivering accurate, relevant, and fluent responses.
- We monitored various metrics throughout the fine-tuning process, such as loss, gradient norm, and learning rate, to track how the model converges and optimizes during training. The following graphs provide insights into the model's learning behavior during fine-tuning.

Loss Trend During Fine-Tuning (Top Left Graph)

- The graph represents the loss reduction over epochs during the fine-tuning process. Loss is a measure of how well the model predictions align with the target job descriptions. The loss starts relatively high (around 4.5) and decreases consistently as the training progresses.
- By the middle of the process, the loss drops to around 2.0, indicating that the model is learning and making fewer mistakes in its predictions. Some fluctuations are visible early in the training process (around 0.1 epochs), which is typical as the model adjusts to the new data.
- A steady decrease in loss indicates that the model is successfully learning from the job-related dataset. The model gradually improves its predictions, which results in more accurate and relevant job descriptions for user queries.

Gradient Norm Trend During Fine-Tuning (Top Right Graph)

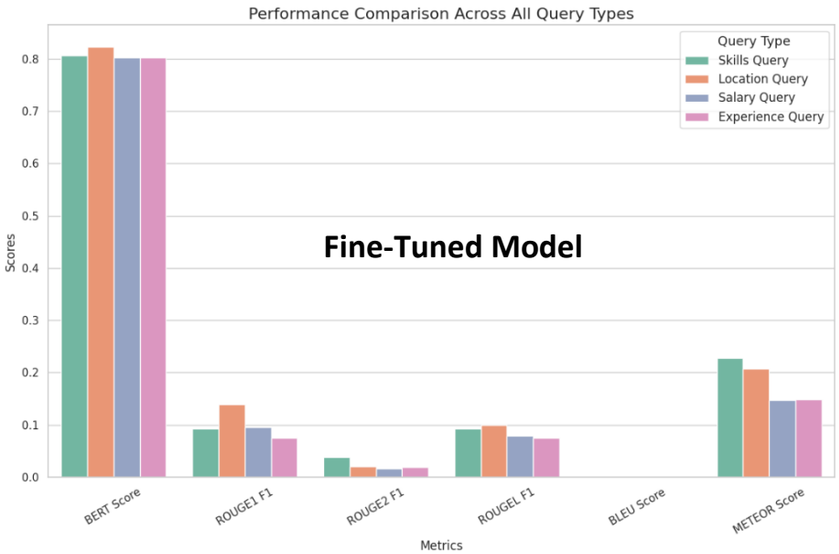
- The gradient norm measures the magnitude of updates made to the model's weights during backpropagation. It gives an indication of how much the model is learning at each step.
- The gradient norm starts high (around 10) but decreases rapidly as training progresses, stabilizing near 1 after 0.1 epochs.
- This stabilization is crucial because it reflects that the model has found a good balance in the update steps, avoiding excessively large or small adjustments that could harm the learning process.
- Early in the training process, the model makes larger updates to its parameters as it starts to learn from the data. Once the gradient norm stabilizes, the model makes more fine-grained adjustments to improve its performance, suggesting that it is converging towards an optimal solution.

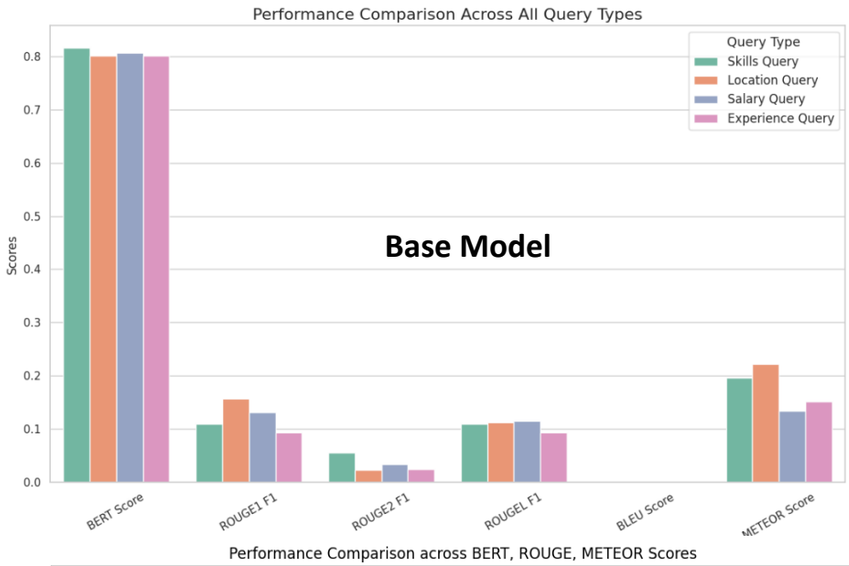
Loss and Gradient Norm Comparison (Bottom Left Graph)

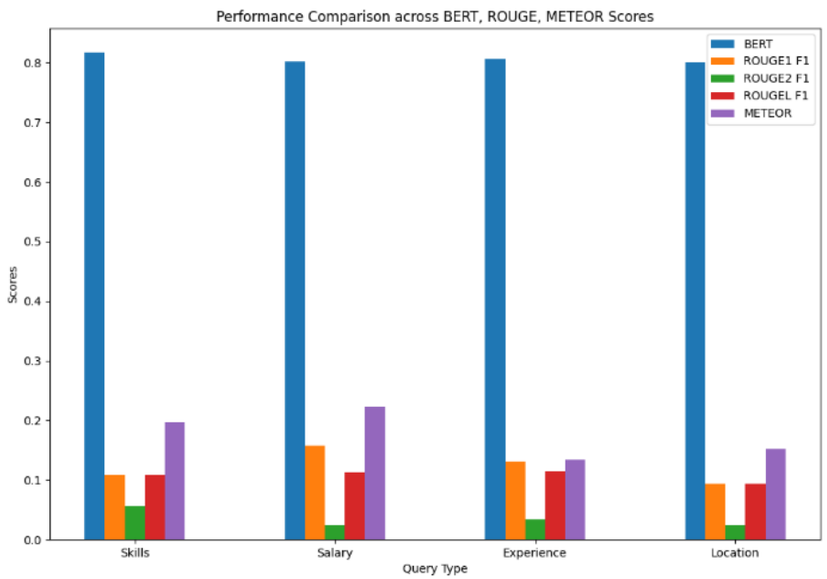
- This graph shows the relationship between loss and gradient norm throughout fine-tuning. By comparing both metrics, we can better understand how the model adjusts its learning process over time.
- Initially, both loss and gradient norm are high, indicating that the model is making significant updates and learning quickly from its mistakes.
- As the gradient norm stabilizes, the loss continues to decrease, showing that the model's learning becomes more stable and consistent over time.
- The convergence of the gradient norm, along with the steady decrease in loss, suggests that the model is finding an optimal path in the training process. The gradual reduction in loss after the gradient norm stabilizes is a sign of efficient learning, as the model fine-tunes its parameters in smaller steps to achieve better performance.

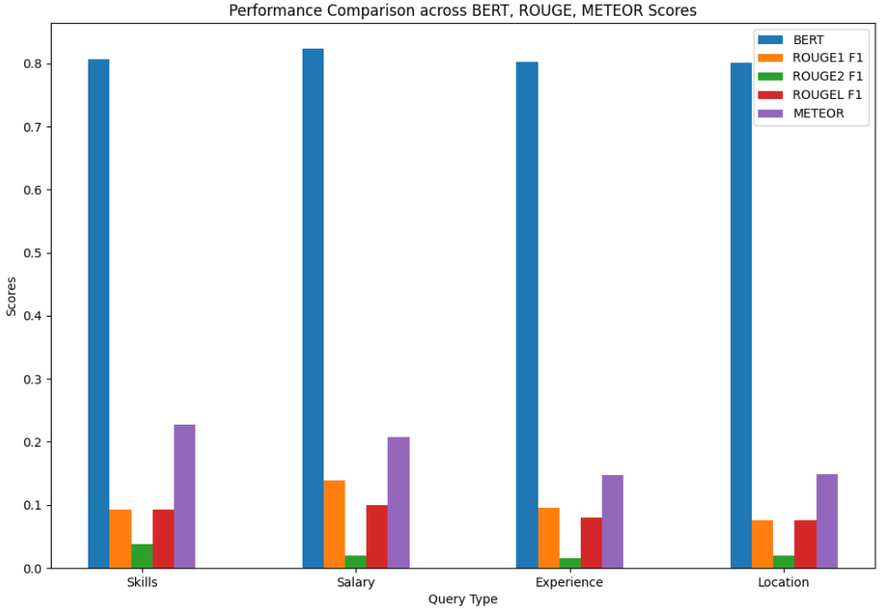
Learning Rate over Epochs (Bottom Right Graph)

- The learning rate controls the size of the updates to the model's parameters during training. A higher learning rate leads to faster updates, while a lower learning rate results in more cautious adjustments.
- The learning rate starts high (around 5e-5) and gradually decays as training progresses.
- This is a common learning rate schedule, where the model starts with aggressive learning and slows down over time to make finer adjustments.
- The decaying learning rate ensures that the model makes rapid progress at the beginning, learning the most significant patterns in the data, and then refines its knowledge as the training continues. This helps prevent overshooting the optimal solution and ensures that the model converges effectively.


Performance Comparison Across All Query Types
Query Type
Skills Query
Location Query
Salary Query
Experience Query
Fine-Tuned Model
Scores
Metrics
BERT Score
ROUGE1 F1
ROUGE2 F1
ROUGEL F1
BLEU Score
METEOR Score


Performance Comparison Across All Query Types
Query Type
Skills Query
Location Query
Salary Query
Experience Query
Base Model
Scores
BERT Score
ROUGE1 F1
ROUGE2 F1
ROUGEL F1
BLEU Score
METEOR Score


Performance Comparison across BERT, ROUGE, METEOR Scores
BERT
ROUGE1 F1
ROUGE2 F1
ROUGEL F1
METEOR
Scores
Skills
Salary
Experience
Location
Query Type


Performance Comparison across BERT, ROUGE, METEOR Scores
BERT
ROUGE1 F1
ROUGE2 F1
ROUGEL F1
METEOR
Scores
Skills
Salary
Experience
Location
Query Type

b. Performance Comparison across metricess and Query Types

BERT Score:

- Both models maintain high BERT scores (around 0.8), which indicates strong semantic similarity between the generated responses and the reference job descriptions.
- Across query types (skills, location, salary, experience), there is very little variation, showing that both models handle these types consistently well when evaluated for meaning similarity.

RoGUE Scores (F1, Precision): The LoRA fine-tuned model consistently performs better than the base model in capturing content overlap:

- RoGUE-1 F1 (unigram overlap) shows the LoRA model slightly outperforms the base model, particularly in location and salary queries. This suggests that the fine-tuned model is better at picking up key job-related words from the dataset.
- RoGUE-2 F1 (bigram overlap) and RoGUE-L F1 (longest common subsequence) scores are low across both models but are higher in LoRA, reflecting its better structural and lexical matching with the training data.

Experience queries have the lowest scores for both models, indicating that handling complex and longer queries might still be challenging.

BLEU Score:

- Both models have consistently low BLEU scores across all query types, indicating that n-gram overlap is not a strong point for either model. This is likely because job descriptions involve diverse wording, making n-gram matching less effective in this task.
- This suggests that BLEU might not be the best metric to evaluate the performance of job search conversational models where semantic meaning and contextual relevance are more critical.

METEOR Score:

- The LoRA fine-tuned model shows higher METEOR scores compared to the base model, especially in skills and location queries. METEOR captures synonymy and word order flexibility, meaning that LoRA is better at understanding different word choices that describe the same concept.
- The base model performs consistently lower, showing that it struggles with more nuanced or semantically rich content generation.

Key Insights:

- BERT scores are high for both models, showing that semantic similarity is strong across the board.
- LoRA fine-tuning brings significant improvements in RoGUE metrics, meaning that it captures more relevant content and structure from the dataset.
- BLEU scores are consistently low for both models, suggesting that exact n-gram overlap is not a useful metric in this context.
- METEOR scores demonstrate that the LoRA model handles semantic richness and word choice flexibility better, especially for skills and location-based queries.
- Experience queries remain a challenge for both models, indicating that more work is needed to handle longer and more complex queries effectively.

5. Alternative Solution - Natural Language Analytics

In our previous work, we concluded that fine-tuning large language models (LLMs) is not the most effective approach for addressing our problem. While Retrieval-Augmented Generation (RAG) offers some advantages over fine-tuning, we believe that an alternative approach may provide the best return on investment (ROI).

From a user perspective, our goal is to provide relevant job postings based on natural language queries, such as *"What jobs are available for Python developers in New York?"* This requirement is essentially a search and retrieval problem that may not necessitate generative capabilities for displaying search results. However, generative models could still be valuable for maintaining sustained conversations with users and remembering attributes of their profiles across multiple prompts.

This realization led us to develop a Natural Language Analytics tool for job-related queries. Below is a step-by-step documentation of our approach.

Step 1: Preprocessing Unstructured Data into Structured Data

We began by utilizing the cleaned dataset from our previous fine-tuning efforts. We performed Named Entity Recognition (NER) on the "Resume" and "Description" columns to extract and classify information into new features such as **"Skills," "Experience_Level," "Job_Role," "Education_Level," "Remote_Work," "Min_Salary,"** and **"Max_Salary."** To accomplish this, we employed two different methods: custom Natural Language Processing (NLP) code and leveraging ChatGPT for NER tasks.

Step 2: Converting Natural Language Prompts to SQL Queries

The next step involved translating user-generated natural language prompts into corresponding SQL queries. Traditionally, search engines have used rule-based systems for this purpose. While rule-based approaches are easier to implement and test, they lack generalizability. On the other hand, LLMs offer greater flexibility but can be unpredictable.

To bridge this gap, we experimented with both methods. We provided the database schema and specific instructions to the model to convert plain English queries into SQL statements. For this task, we used the **Llama 3.1 8B model**, which ensured high-quality SQL queries while maintaining near real-time response times. If necessary, we can fine-tune open-source LLMs to enhance their performance in converting natural language to SQL queries and implement robust error handling.

Step 3: Executing SQL Queries and Displaying Results

We used SQLite to manage the database and execute the SQL queries generated by the LLM. The results were displayed directly in a tabular format, providing users with clear and immediate access to relevant job postings.

Step 4: (Optional) Conversational Response

To enhance user experience, we fed the SQL query results back into the LLM. This allowed the system to assist users with subsequent questions related to the results, facilitating a more interactive and conversational interface.

Limitations

While this approach yielded significantly better results with minimal investment in human effort and computational resources, it was largely effective because we operated within a simplified environment where all relevant data existed in a single table. In real-world applications, data is often siloed across multiple databases owned by different departments within an organization, posing a significant challenge.

Additionally, we worked with brief "Resume" and "Description" entries rather than extensive PDF documents. Performing NER across a large number of PDF files with varied formats introduces additional complexities, such as the need for advanced document parsing and increased computational overhead.

Finally, to effectively compare our alternative approach with fine-tuning, RAG, and baseline models, we will require human evaluation and search-engine-related metrics such as **precision, recall,** and **normalized discounted cumulative gain (NDCG)**. Metrics like **BERTScore** and **ROUGE**, which focus on semantic similarity and overlap with reference text, may not be appropriate for evaluating search and retrieval tasks where the output is a set of relevant results rather than generated text.

Conclusion

Our alternative approach demonstrates that leveraging structured data and LLMs for natural language query translation can be an efficient solution for search and retrieval problems. While there are limitations, particularly when scaling to more complex and varied datasets, this method offers a promising direction for providing users with relevant job postings based on natural language queries.

6. References

Chiusano, F. (2022, November 1). *Learn the BLEU metric by examples | by Fabio Chiusano | NLPlanet | Medium*. https://medium.com/nlplanet/two-minutes-nlp-learn-the-bleu-metric-by-examples-df015ca73a86

Encord. (2023, August 22). *Fine-tuning Models: Hyperparameter Optimization | Encord*. https://encord.com/blog/fine-tuning-models-hyperparameter-optimization/

Paoli, C., & 08/21/2024. (n.d.). *Microsoft Launches Phi 3.5-Mini Models—Redmondmag.com*. Redmondmag. Retrieved October 13, 2024, from https://redmondmag.com/Articles/2024/08/21/Microsoft-Launches-Phi-Mini-Models.aspx

Sooriyarachchi, A. (2023, August 30). *Efficient Fine-Tuning with LoRA: A Guide to Optimal Parameter Selection for Large Language Models*. Databricks. https://www.databricks.com/blog/efficient-fine-tuning-lora-guide-llms

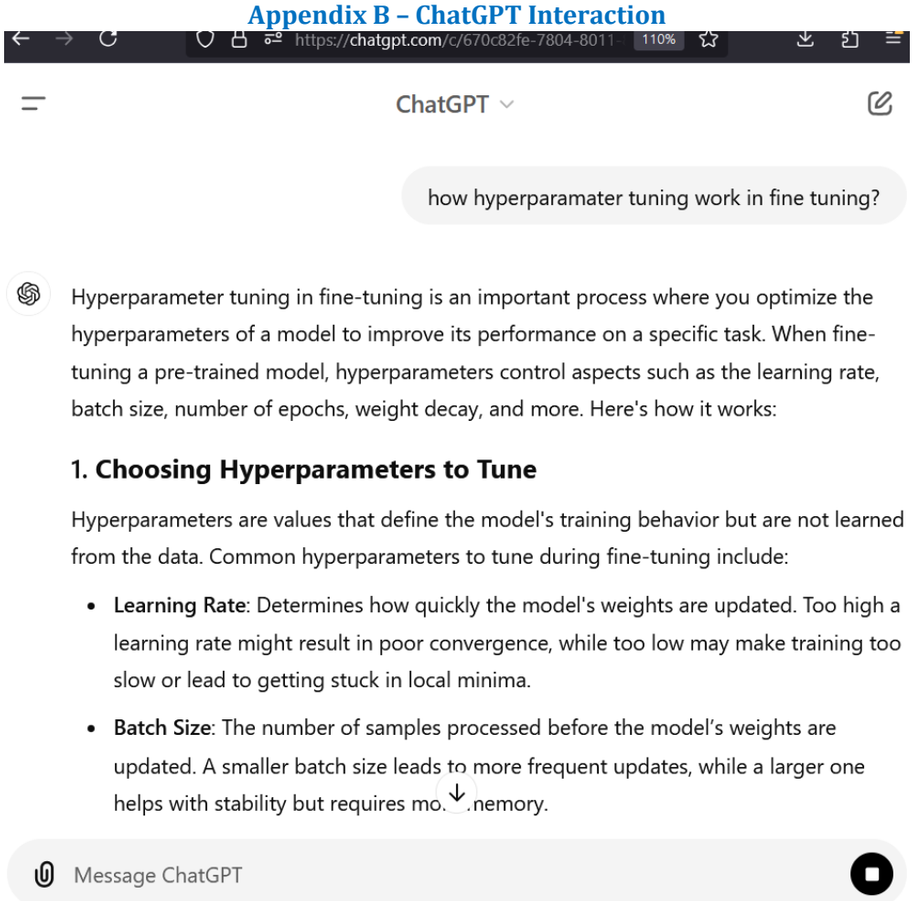
Tianyi. (2023, February 20). *Evaluate-metric (Evaluate Metric)* [Community]. Hugging Face. https://huggingface.co/evaluate-metric

Appendix

Appendix A - Predefined Query Templates

```
# List of instruction variations
instruction_templates = [
    "Answer the following job-related query accurately.",
    "Provide job information based on the user's question.",
    "Respond to the job search query with precise details."]

# List of query variations for different categories
skills_query_templates = [
    "What jobs are available for {}?",
    "Are there jobs for {}?",
    "What roles are available for {}?"]

location_query_templates = [
    "What jobs are open in {}?",
    "What jobs are available in {}?",
    "Give me the information of available jobs in {}."]

salary_query_templates = [
    "What jobs offer a salary of {}?",
    "Which jobs pay over {}?",
    "Are there jobs with a salary of {}?"]

experience_query_templates = [
    "What jobs are available for someone with {} years of experience?",
    "What roles can I find for someone with {} years of experience?",
    "Are there jobs for candidates with {} years of experience?"]
```

Appendix B – ChatGPT Interaction



Full Chat → https://chatgpt.com/share/670c83c7-00d0-8011-ba7f-5adefa7e0324

Appendix C – Command-line User Interface

```
== Welcome To Conversational AI for Job Search ==
Select Which Step You Want to Proceed:
1. Data Preprocessing
2. Data Preprocessing with Summarization Process for Resume & Description Columns
3. Fine-tune the Phi-3 model on <Job Query, Job Description> pairs
4. Handle user queries for job searches based on different criteria (skills, location, salary)
5. Evaluate the model using ARES, RoGUE, BLEU, and METEOR metrics.
6. Exit
== Welcome To Conversational AI for Job Search ==

Enter your choice (1-6):
```