THE UNIVERSITY OF NEW SOUTH WALES

The School of Computer Science Engineering

SEEK-Project Report

COMP6713 Natural Language Processing 25 Term 1 Project

Coolest Team

Zihao Zhou (z5423698)

Xinghua Wu (z5526892)

Jinghan Wang (z5286124)

Date of submission: 28 April 2025

PART A: Problem Definition ***Minimum: 10 credits***			***Fill your project details here***
	Points	*Details*	
NLP Problem	*5 credits per problem*	*Examples: Question-answering, Sentiment Analysis, etc. (Refer to course modules) (For industry projects, specify the research question.)*	Extraction and generation tasks
Text Source/Domain	*5 credits per text source/domains*	*Examples: News Articles, Medical papers, etc.*	
PART B: Dataset Selection ***Minimum: 20 credits***			
Use two existing datasets	*10 credits*	*Publicly available datasets. In the case of industry projects, it will be the datasets provided.*	Provided datasets
Create your own labelled dataset	*20 credits*	*Correct selection of labels, inter-annotator agreement*	Implemented
Use an existing lexicon	*10 credits*	*Examples: WordNet, medical ontology*	
PART C: Modelling			
Implement a rule-based or statistical model as a baseline	-	*Essential. Please keep this approach simple – this is only a baseline.*	SVM
Minimum: 30 credits			
Use an existing pretrained/fine-tuned model	*5 credits per model*	*You must compare the performance of multiple models to accrue credits, if you are only using available fine-tuned models or only prompting.*	Llama3.1-8B-Instruct Qwen2.5-32B-Instruct Bart-large XLM-RoBERTa-Large XLM-RoBERTa-Longformer-base-4096

Fine-tune a model based on a dataset	*20 credits per out-of-the-box fine-tuning method*	*Examples: fine-tuned BERT, prefix-tuned LLaMA*	Fine-tuned Llama3.1-8B-Instruct
Extend a method	*30 credits per extended finetuning method*	*Examples of extension: modification of the loss function, incorporation of structured ontology, prompting method other than zero/few-shot prompting, etc.*	
Integrate a language model with external tools	*20 credits*	*Usage of a library like LangChain*	Langchain
PART D: Evaluation Minimum: 20 credits			
Quantitative Evaluation	*10 credits*	*Appropriate metrics.*	Confusion matrix Accuracy、F1、Precision、Recall Semantic Textual Similarity
Qualitative Evaluation	*5 credits*	*Examine misclassified instances and produce common error types*	Implemented
Command line testing	*5 credits*	*Interface to test out the system. This can be executed using an input argument or input file.*	Implemented
Demo	*10 credits*	*A simple demo through Gradio (or equivalent).*	

Table of Contents

Introduction

At the heart of SEEK's business is the capability to match job seekers with relevant openings from a vast and constantly changing corpus of job advertisements. Accurate extraction of key information—such as salary ranges, work arrangements, and seniority levels—enables candidates to filter listings by their preferences and helps employers reach the right talent efficiently. Traditional, pre-neural solutions (e.g., rule-based parsers or shallow machine-learning classifiers) often struggle to generalize across diverse ad formats and can require extensive manual feature engineering.

Generative pre-trained LLMs offer a compelling alternative: their deep contextual understanding and reasoning capabilities allow them to adapt to varied text structures and implicit language cues dynamically. By leveraging these models, SEEK aims to evaluate trade-offs in cost, speed, and accuracy compared to proprietary APIs and open-source weights and to benchmark against established pre-neural approaches to information extraction.

Data sources and scope

The dataset is divided into three parts:

1. Development set (unlabelled)
2. Development set (labelled)
3. Test set (labelled)

Each labelled subset supports salary extraction, seniority extraction, and work-arrangement extraction.

Work Arrangement

- **Input**: A fixed-structure document containing the fields
  job_title, abstract, employer, location, highlights, and contents.
- **Target classes**: Remote | OnSite | Hybrid
- **Data volume**: 100 training examples, 100 test examples

Seniority

- **Input fields**:
    - job_title (plain text)
    - job_summary (plain text)
    - job_ad_details (HTML-formatted text)
    - classification_name (plain text)
    - subclassification_name (plain text)
- **Output**: One or more keywords indicating seniority level. Some labels are drawn verbatim from the input fields; others require a concise summary of the role's description.
- **Data volume**: 2,753 training examples, 690 test examples

Salary

- **Input fields**:
    - job_title (plain text)
    - job_ad_details (HTML-formatted text)
    - nation_short_desc (plain text)
    - salary_additional_text (plain text)
- **Output format**:
    - <min_salary>-<max_salary>-<currency_code>-<payment_period>
    - When no salary is provided, the output is "0-0-None-None".
    - Partial "None" or 0 alone does not occur.
- **Categories**:
    - **Currency codes** (9 total): PHP, HKD, NZD, MYR, AUD, SGD, THB, IDR, USD.
    - **Payment periods** (5 total): MONTHLY, HOURLY, ANNUAL, DAILY, WEEKLY.
- **Data volume**: 2,268 training examples, 568 test examples

1. Pre-Training Neural Network and Pre-Neural Network

1.1 Data Labelling process

Since the model is based on the XLMRobertaForQuestionAnswering architecture, it requires exact knowledge of where the minimum and maximum salary values occur in the text.

First, remove all HTML and concatenate the job-ad fields into a single plain-text string to provide this. In Label Studio, annotators then marked the precise character-level start and end offsets of each “MIN” and “MAX” salary substring. Label Studio automatically recorded these offsets, which we later mapped to the tokenizer’s indices. This fine-grained span annotation delivers the exact supervision needed for our boundary-prediction heads, allowing the model to learn from local context rather than relying on imprecise heuristics or regular expressions.

The final output is a comprehensive label set, including the raw text, salary label, and corresponding min/max span positions.


```
{
  "input_list": "Financial Account … Compensation 17500 Compensation Range ₱15,000 – ₱20,000 PH",
  "y_true": "17500-17500-PHP-MONTHLY",
  "id": 1,
  "label": [
    {
      "start": 1291,
      "end": 1296,
      "text": "17500",
      "labels": ["MIN"]
    }
  ],
  "annotator": 1,
  "annotation_id": 1,
  "created_at": "2025-04-24T09:51:52.308486Z",
  "updated_at": "2025-04-24T09:58:01.055057Z",
  "lead_time": 101.4
}
```


In the annotation process, we identified a small proportion (approximately 8 %) of erroneous labels in the training set. These errors fell into three main categories:

1. **Phantom salary ranges**
   Some records were marked with minimum and maximum salaries even though no such terms appeared in the original text.

2. **Misclassified numeric values.**
   Occasionally, weekly working hours or other unrelated numbers had been annotated as salary figures.
3. **Inconsistent decimal handling and outright numeric mistakes.**
   Annotations did not always follow a standardized scheme for decimals (e.g., rounding to the nearest integer, truncating fractional digits, or applying a ceiling), leading both to direct value errors and variability in how decimals were recorded.

To remediate these issues, we applied the following procedures:

- **Exclusion of phantom salary cases.**
  Records labelled with unsubstantiated “min/max salary” values were left unannotated and removed from the training data, preventing noise from entering the model.
- **Re-annotation of misclassified numbers.**
  Instances where non-salary figures (such as weekly hours) had been tagged as salary were re-examined and annotated according to the actual semantic content of the source text, ignoring the original (incorrect) labels.
- **Tolerance-based evaluation for numeric inconsistencies.**
  During the model evaluation, we introduced a ±1-unit tolerance around the actual salary value to accommodate remaining discrepancies and the lack of a uniform decimal-processing rule. Predictions within this band are treated as correct, thus minimizing spurious performance penalties due to minor annotation irregularities.

1.2 Methodology & Model

1.2.1 XLM RoBERTa (Pre-Neural Network)

XLM-RoBERTa is a transformer-based multilingual masked language model that extends Facebook's RoBERTa method to over 100 languages. These features make XLM-RoBERTa an ideal base model for span extraction and classification tasks, enabling robust extraction of salary ranges, currencies, and payment cycles from heterogeneous job postings, regardless of language.

1.2.1.1 Work-arrangement Task

The work-arrangement task involves three categories (Remote, on-site, Hybrid) and is therefore modelled as a straightforward classification problem. The backbone is a trained XLM-RoBERTa-Large encoder (maximum sequence length: 512 tokens). To cover lengthy job descriptions in full, a sliding-window approach is adopted:

- **Training:** Random 512-token windows are sampled from each preprocessed document. This caps computational cost and augments data by exposing the model to diverse contextual excerpts.
- **Structural Cues:** Special section markers—such as [TITLE], [ABSTRACT], and [CONTENTS]—are prepended to each field, guiding the model to recognise document structure and attend to the most relevant segments.
- **Test:** A fixed-stride sliding window generates overlapping 512-token segments across the entire text. Each segment is classified independently, producing a probability distribution over the three labels. A confidence-weighted aggregation of these segment-level predictions yields the final work-arrangement label.

1.2.1.2 Salary Task

Salary ranges often appear anywhere in a long job description. With a 512-token limit, there is a high risk that a salary value falls outside the current window—yet a standard QA model might still predict a span, introducing noise. The XLM-RoBERTa-Longformer-base-4096 variant is employed to eliminate this issue, extending context up to 4,096 tokens.

A Span extraction formulated as a QA task is then applied: two separate heads predict the start/end positions of the minimum and maximum salary. Because every training example includes explicit character-level annotations for “MIN” and “MAX,” the model learns to output “no span” (i.e. 0) whenever those values are truly absent, rather than hallucinating a salary where none exists.

When both min and max are missing or non-numeric, the system returns 0-0-none-none and skips currency/period prediction entirely. Only when at least one bound is found does it classify currency and payment period—two tasks cast as fixed-label classifiers (9 currency codes; 5-period types) rather than free-form generation. This choice avoids formatting or spelling errors and yields higher accuracy than a decoder-based approach.

The model architecture adds two QA heads and two classification heads atop XLM-RoBERTa-Longformer-base-4096. Training is staged in two phases:

1. **Span-only training:** Freeze both classification heads; train all span heads to detect min/max positions.
2. **Complete training:** Discard any “0-0-None-None” cases, freeze the first six transformer layers, and train the top six layers with all four heads, refining span extraction and classification jointly.

At inference time, the pipeline first locates “min_val” and “max_val”. If both fail or yield non-numeric output, it returns “0-0-None-None”; otherwise, it maps the detected span values to currency and period labels to form the final output.

1.2.2 Bart (Pre-Neural Network)

BART (Bidirectional and Auto-Regressive Transformers) is a powerful seq2seq model that combines a bidirectional encoder (like BERT) with an autoregressive decoder (like GPT). Pretrained as a denoising autoencoder—where text is purposefully corrupted and reconstructed—BART learns to generate coherent, contextually rich outputs across various generation tasks (translation, summarization, paraphrasing).

1.2.2.1 Seniority Task

The output for this task is an open-ended descriptive phrase rather than a fixed, rule-based label, and many target terms do not appear verbatim in the source text—so it cannot be framed as a classification or span-annotation problem. Instead, a seq2seq approach is used with the multilingual BART-large model. Its 1,024-token encoder window covers almost all job descriptions; any overflow is truncated, which has a negligible impact on generation quality. The decoder is limited to 12 output tokens (roughly 2–4 words) to enforce brevity, preventing overly long or rambling summaries.

Because these outputs are inherently summarizations, exact string matching would unfairly penalize semantically equivalent variants (e.g. “large” vs. “big”). Therefore, the paraphrase-multilingual-MiniLM-L12-v2 model is employed to compute a cosine-similarity score between the generated phrase and the reference. This semantic-distance metric becomes the

primary evaluation criterion, ensuring that meaning—rather than surface form—drives the assessment.

1.2.3 Support vector machine / Conditional random field (Pre-Neural Network)

Support Vector Machine is a supervised learning algorithm for classification (and regression) by finding the hyperplane that maximally separates data points of different classes in a high-dimensional feature space. By leveraging the **kernel trick** (e.g. linear, polynomial, RBF), SVM can handle both linearly separable and non-separable data by implicitly mapping inputs into richer representations. During training, it optimizes a convex quadratic problem to maximize the margin—the distance between the hyperplane and the nearest training samples (the “support vectors”)—yielding robust generalization even with limited data. SVMs are particularly effective for high-dimensional, sparse datasets (like text) and serve as a strong non-neural baseline for document classification, image recognition, and bioinformatics tasks.

1.2.3.1 Seniority Task / Work-arrangment Work

For the pre-neural network SVM approach, both tasks were framed as standard classification problems. The model classified inputs into one of three predefined categories for the work arrangement task. Each unique term observed in the training set was treated as a distinct class for the seniority task.

While the SVM framework exhibited negligible performance degradation in the work arrangement task compared to neural network-based approaches, its lightweight architecture resulted in substantially faster training and lower computational requirements. However, in the seniority task, this approach inherently constrained the model’s generative capacity: the model was limited to producing only those terms encountered during training, thereby forfeiting the ability to synthesize novel outputs beyond the original vocabulary.

Before model training, all input text was transformed into numerical feature representations via TF-IDF (Term Frequency–Inverse Document Frequency) encoding. No additional special tokens or prompt engineering techniques were incorporated. The TF-IDF vectorizer generated fixed-length vectors that captured the relative importance of words across the corpus. Subsequently, the SVM classifier learned a linear decision boundary within this high-dimensional feature space to perform category prediction.

1.2.3.2 Salary Task

As previously discussed, while SVM remains effective for handling currency and payment periods as classification tasks, it is not well-suited for numerical extraction. Due to its inability to model sequential dependencies, SVM struggles to extract continuous numerical spans accurately. Therefore, a Conditional Random Fields (CRF) model was employed for the extraction component to predict the positions of salary-related annotations.

The input text was first tokenized and span-aligned based on the annotated training data. Each original text sample was segmented into a sequence of tokens and corresponding character-level spans, allowing precise alignment between character-level annotations and their associated tokens. Each training instance was then converted into a (tokens, tags) format, where the tags follow the BIO encoding scheme:

- B- denotes the beginning of an entity,
- I- denotes the inside of an entity,
- O denotes tokens outside any entity.

This structure accurately captures salary amounts, currency symbols, and related entities to be extracted. To transform tokens into a feature representation usable by CRF, a feature set was constructed for each token, including:

- Lexical features such as lowercase form, digit checks, and hyphen presence,
- Morphological features such as token length, prefixes, and suffixes,
- Contextual features from adjacent tokens (previous and next tokens' basic attributes).

The entire token sequence was then converted into a corresponding feature sequence, serving as the input for CRF model training.

The procedure mirrors the above steps for the generation phase: if the minimum and maximum salary values cannot be found, a default value of 0-0-None-None is returned. If salary spans are successfully extracted, the previously trained SVM models are used to predict the associated currency and payment period.

1.3 Evaluation & Discussion

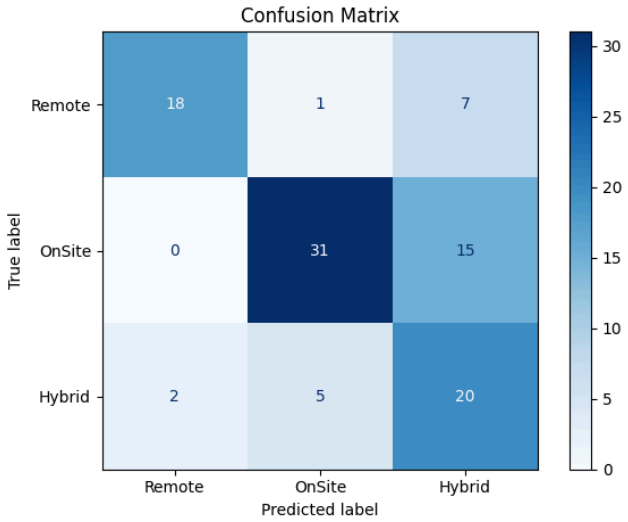
1.3.1 Work Arrangement Task

1.3.1.1 XLM-Roberta-Large

The classification outputs were evaluated using accuracy, precision, recall, and F1-score, yielding:

- Accuracy: 0.6970
- Precision: 0.7380
- Recall: 0.7023
- F1-score: 0.7031

Below is the confusion matrix for the three classes.



- Remote
    - Precision 90.0%: 18 out of 20 predicted Remote jobs were correct, so false positives are rare.
    - Recall 69.2%: 7 of 26 actual Remote roles were missed (6 of those labeled Hybrid), indicating under-coverage.
- OnSite
    - Precision 83.8%: 31 of 37 OnSite predictions were true, with most errors coming from misclassified Hybrid examples.
    - Recall 67.4%: 15 of 46 true OnSite roles were labeled Hybrid, showing confusion between purely on-site and mixed arrangements.

- Hybrid
    - Precision 47.6%: Less than half of “Hybrid” predictions correspond to true Hybrid roles, revealing a high false-positive rate.
    - Recall 74.1%: The model captures most actual Hybrid postings, but often at the expense of over-assigning this category.

Error Patterns

- OnSite → Hybrid: 15 cases
- Remote → Hybrid: 7 cases
- Hybrid → OnSite: 5 cases

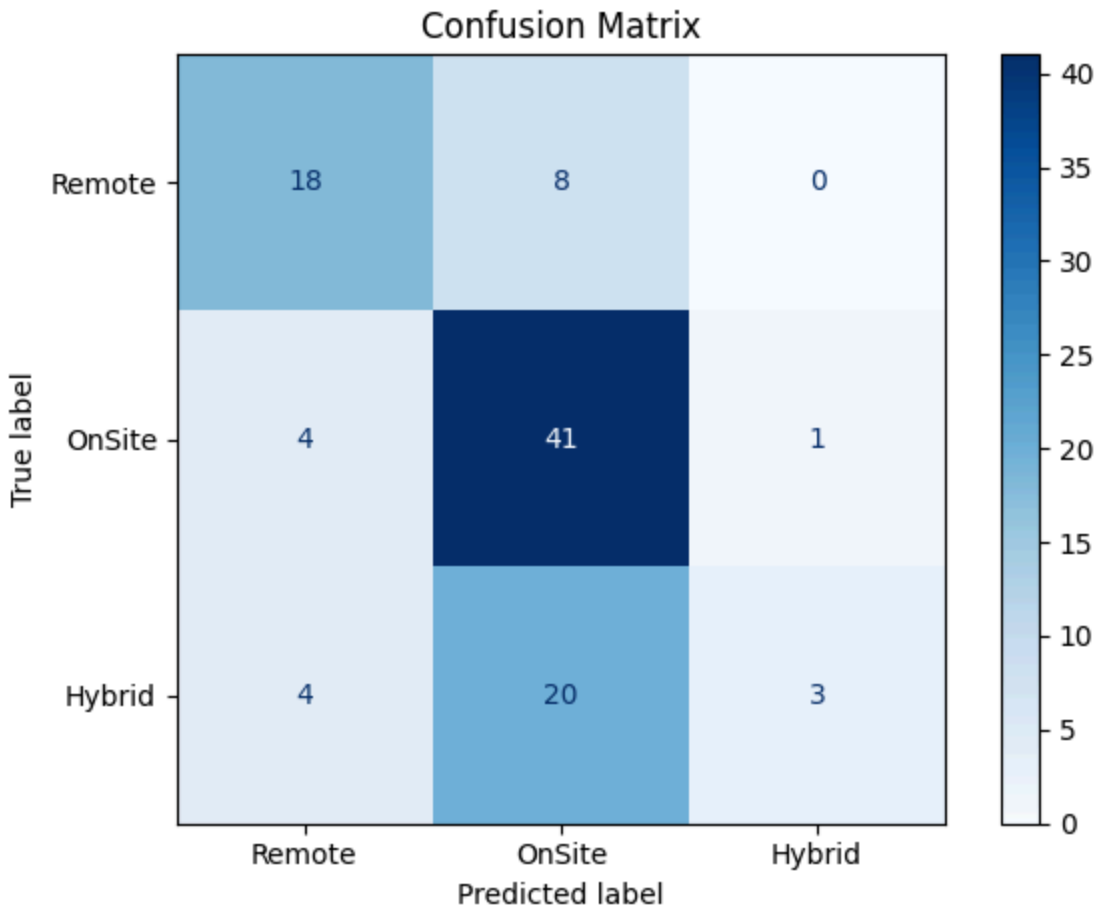
Hybrid acts as a “catch-all” when the model is uncertain, leading to many misclassifications into that class. To improve performance, the training set could be augmented with more clear-cut Remote and OnSite examples, and additional structural cues (e.g. “on-site only,” “must commute”) might help the model better distinguish pure vs. mixed work arrangements.

1.3.1.2 SVM (Work Arrangement Task)

The classification outputs were evaluated using accuracy, precision, recall, and F1-score, yielding:

- Accuracy: 0.6263
- Precision: 0.6788
- Recall: 0.5649
- F1-score: 0.5330

Below is the confusion matrix for the three classes (Remote, OnSite, Hybrid).



- Remote
    - Precision 81.8%: 18 out of 22 predicted Remote jobs were correct, indicating a relatively low false-positive rate.
    - Recall 69.2%: 18 out of 26 actual Remote roles were correctly identified, with 8 mislabeled as OnSite, suggesting moderate under-coverage.

- OnSite
    - Precision 59.4%: 41 out of 69 OnSite predictions were true, with significant misclassifications coming from Hybrid roles predicted as OnSite.
    - Recall 89.1%: 41 out of 46 actual OnSite postings were correctly captured, indicating strong sensitivity for this class.

- Hybrid

    - Precision 75.0%: 3 out of 4 predicted Hybrid roles were accurate, though the overall number of Hybrid predictions was low.
    - Recall 11.1%: Only 3 out of 27 actual Hybrid postings were correctly classified, revealing substantial under-detection for the Hybrid category.

Error Patterns

- Hybrid → OnSite: 20 cases
- Remote → OnSite: 8 cases
- Hybrid → Remote: 4 cases

The confusion matrix highlights that **Hybrid roles were frequently misclassified as OnSite**, acting as a fallback when the model was uncertain. Additionally, some Remote roles were confused with OnSite roles, reflecting overlapping textual cues.

1.3.1.3 Discussion

The superior performance of XLM-RoBERTa-Large can be attributed to its ability to leverage deep contextual representations from pre-trained transformer layers, allowing it to capture subtle semantic distinctions crucial in tasks such as work arrangement classification. In contrast, SVM relies purely on surface-level TF-IDF features without contextual awareness, making it more prone to confusion between similar categories, such as Hybrid and OnSite roles.

Furthermore, error patterns revealed that the SVM model exhibited higher rates of misclassification, particularly over-assigning examples to dominant classes like OnSite. In contrast, XLM-R maintained a better balance across all categories. These results highlight the advantages of deep language models in handling nuanced textual classification tasks where context plays a critical role.

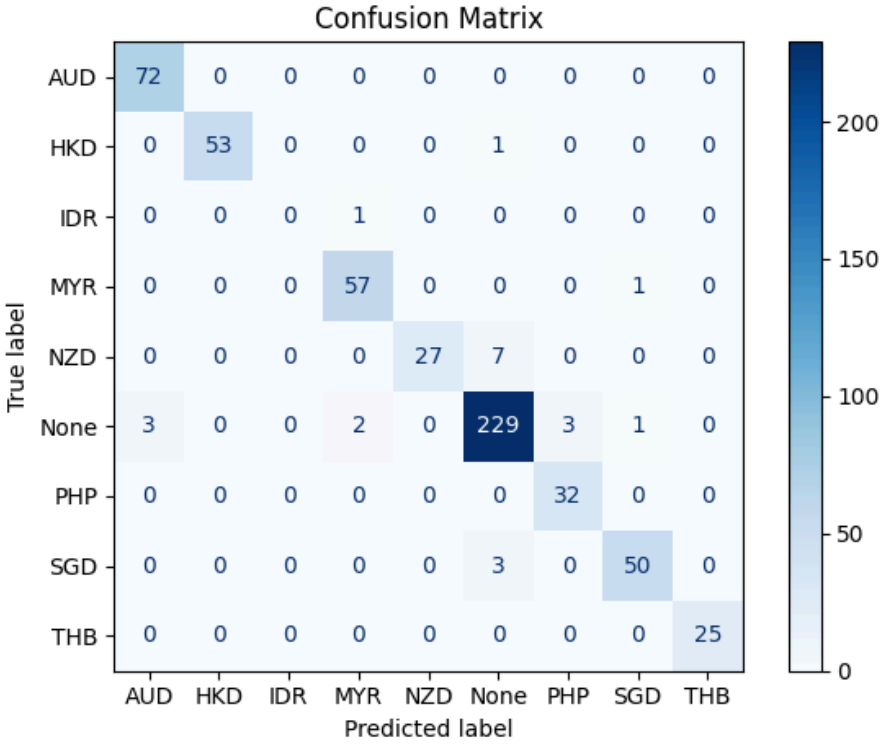
1.3.2 Salary Task

1.3.2.1 XLM-RoBERTa-Longformer-base-4096

Calculate the accuracy of the four predicted values and the actual values, and the results are as follows:

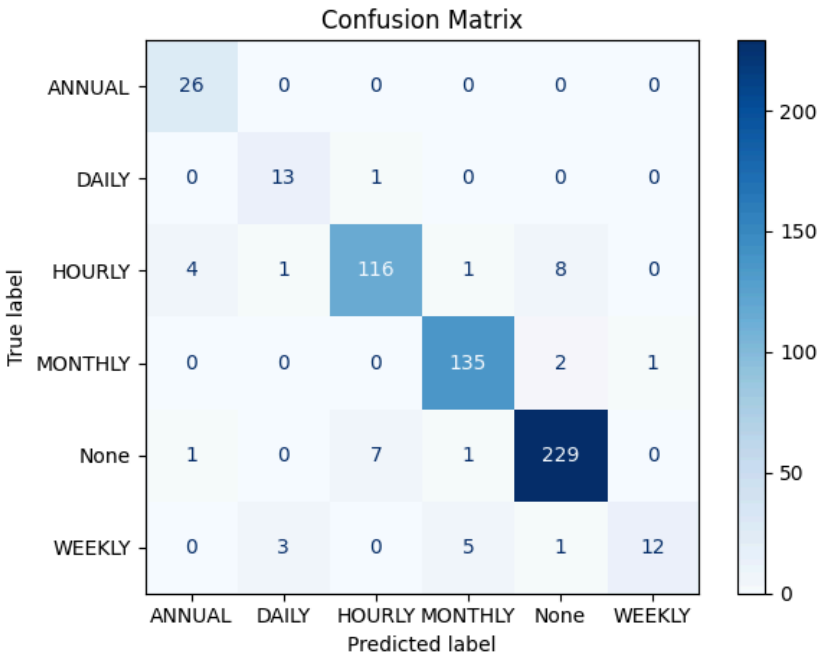
- Accuracy (min values): 0.9347
- Accuracy (max values): 0.9136
- Accuracy (currency labels): 0.9612
- Accuracy (period labels): 0.9365

Currency Classification:



The confusion matrix for currency predictions demonstrates strong diagonal dominance, indicating high precision and extremely low misclassification rates. Minor classification errors primarily involve “NZD” and “None,” where seven instances of “NZD” were misclassified as “None,” and three instances of “None” were incorrectly predicted as “AUD.” Other currency categories (AUD, HKD, MYR, PHP, SGD, THB) show excellent prediction consistency with negligible confusion.

Payment Period Classification:



The salary period predictions also performed exceptionally well. The model clearly distinguishes between period categories such as “ANNUAL,” “DAILY,” “MONTHLY,” and “HOURLY.” The only notable issue is that the “HOURLY” category is occasionally misclassified as “None” (8 instances), potentially due to textual ambiguities or labelling inconsistencies within hourly salary descriptions.

The special condition (0-0-none-none) indicates missing salary information and is identified solely by checking if both salary values (min salary and max salary) equal zero. This rule-based approach may slightly inflate the model’s accuracy metrics for currency and period predictions, as these dimensions become automatically correct when the zero-salary condition is triggered. Observing the two confusion matrices, most prediction errors are associated with the “None” category, indicating that most errors stem from inaccuracies in predicting numerical salary values. Excluding the “None” category reveals that the classification heads (currency and period) achieve higher accuracy than the numerical salary extraction heads. This further suggests the model’s performance is more stable in classification tasks. At the same time, numerical salary extraction poses specific challenges or uncertainties, potentially arising from numeric ambiguities or annotation inconsistencies in the dataset.

Exact Match Rates:

The strict, exact match rate across all four dimensions (min, max, currency, and period) is 86.95%, indicating that the model reliably provides entirely accurate salary information.

Considering minor rounding discrepancies discovered in the training dataset, relaxing the exact match criterion to allow a tolerance of ±1 for numerical salary values modestly improves the exact match rate to 88.36%. This confirms the existence of slight variations in salary

annotations within the data, and accommodating this variance slightly enhances the model's practical applicability.

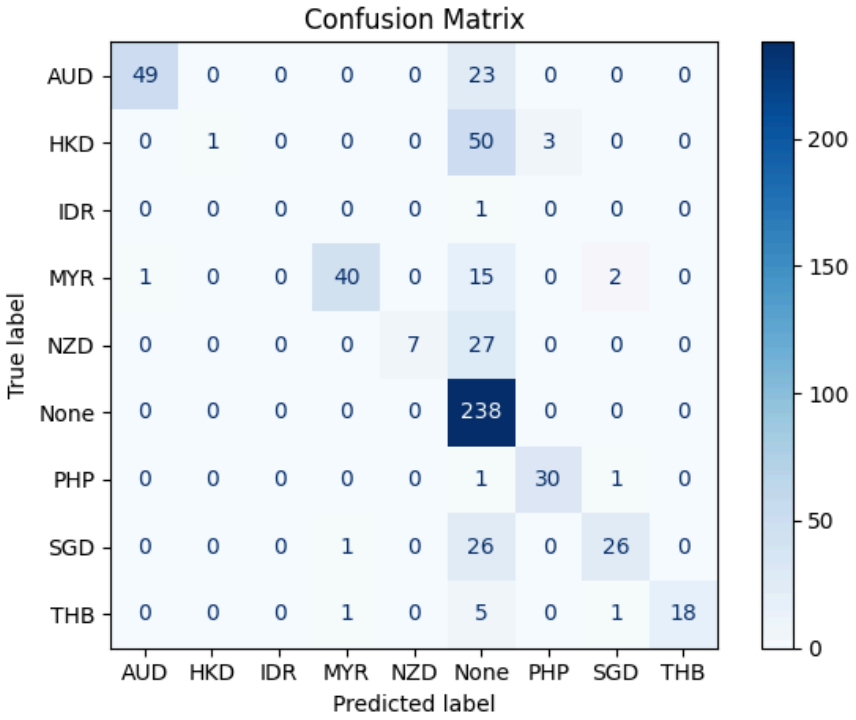
Overall, the model demonstrates excellent predictive accuracy in the salary extraction task. High accuracy rates, robust diagonal dominance in confusion matrices, and reliable exact match rates affirm its strong performance. Future improvements could address ambiguities in hourly salaries and resolve rounding inconsistencies in numeric salary annotations.

1.3.2.2 SVM + CRF

Calculate the accuracy of the four predicted values and the actual values, and the results are as follows:

- Accuracy (min values): 0.5944
- Accuracy (max values): 0.5926
- Accuracy (currency labels): 0.7213
- Accuracy (period labels): 0.7090

Currency Classification:



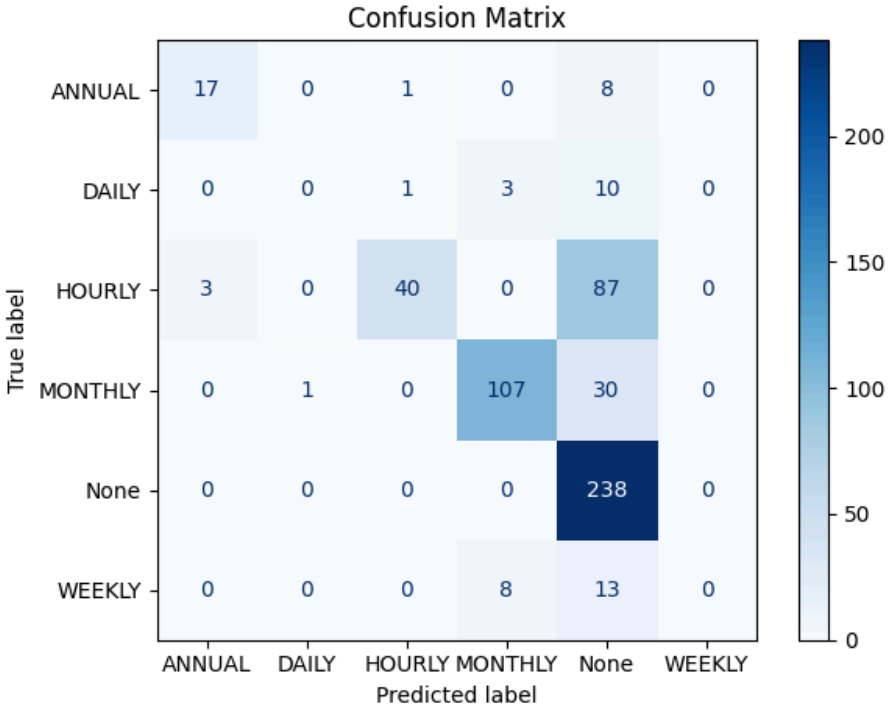
The confusion matrix for **currency predictions** demonstrates general overall accuracy, with a clear diagonal dominance. Most currency categories (AUD, HKD, MYR, PHP, SGD, THB) exhibit excellent precision and minimal misclassification. A few minor errors are observed:

- 23 instances of "AUD" were misclassified as "None,"

- 15 instances of “MYR” were misclassified as “None,”
- 7 instances of “NZD” were misclassified as “None.”

These confusions primarily reflect cases where salary information was mentioned without an explicit currency reference, leading the model to default to “None.”

Payment Period Classification:



The confusion matrix for payment period predictions also reflects general classification performance. Clear distinctions are observed among the major period categories: “ANNUAL,” “MONTHLY,” “HOURLY,” and “DAILY.”

- “MONTHLY” salaries are predicted with the highest consistency (107 correct out of 138 instances).
- Some confusion exists between “HOURLY” and “None,” where 87 instances labeled as “HOURLY” were misclassified as “None.”
- Minor misclassifications also occur between “ANNUAL” and “None,” and between “DAILY” and “None.”

As discussed earlier, most misclassifications involving the “None” class stemmed from extraction failures during minimum and maximum salary identification. When focusing solely on the pure classification task and excluding errors related to the “None” category, both currency and payment period models achieved high reliability.

However, when examining the extraction accuracy of minimum and maximum salary values, it becomes evident that the CRF-based approach falls short of the extraction capabilities of the XLM-RoBERTa model. This gap not only affects the precision of the extracted salary spans but also critically impacts the downstream classification accuracy for currency and payment period predictions.

Moreover, the exact match accuracy (0.5679) and the slightly relaxed match accuracy (0.5697) are almost identical, both substantially lower than the performance achieved by XLM-RoBERTa. This indicates that rounding issues (e.g., slight numerical differences) are not the primary factor limiting model performance; instead, the main challenge lies in the fundamental extraction accuracy of the CRF model.

1.3.2.3 Discussion

When focusing solely on classification tasks, the SVM model demonstrated performance that was relatively comparable to that of the XLM-RoBERTa model. Despite the differences in model architecture, SVM achieved similar levels of precision and recall in predicting categorical labels such as work arrangement, currency, and payment period, especially when errors involving the “None” class were excluded. This suggests that for straightforward, surface-level classification tasks, a simpler linear model combined with strong feature engineering (e.g., TF-IDF) can approximate the performance of deep transformer-based models.

However, a significant performance gap emerges when evaluating span extraction capabilities. The CRF model, used for extracting minimum and maximum salary values, consistently underperformed compared to the XLM-RoBERTa model operating in a QA-style span prediction framework. The CRF’s limited ability to model complex semantic dependencies resulted in lower extraction accuracy and subsequently impacted related classification tasks, such as currency and period prediction.

1.3.3 Seniority Task

1.3.3.1 Bart-Large

Due to the summarization nature of model outputs, exact string matching may unfairly penalize semantically equivalent expressions (e.g., “large” vs. “big”). Therefore, the evaluation adopts a semantic similarity metric calculated using the paraphrase-multilingual-MiniLM-L12-v2 embedding model, which measures the cosine similarity between generated labels and

reference phrases. This semantic-distance criterion ensures assessments reflect true semantic equivalence rather than mere lexical similarity.

Calculate the similarity of the true word and predict word, and the analysis of Evaluation Metrics are as follows:

- High Semantic Alignment (Cosine Similarity: 0.7845):

  An average semantic similarity score of approximately 0.78 indicates that the model's predictions closely align with the semantic meaning of reference phrases. This means most generated labels successfully convey the correct concepts, even when phrased differently.

- Strong Surface Matching (Fuzzy String Similarity: 0.7467):

  A fuzzy string similarity score of about 0.75 demonstrates that, aside from semantic correctness, the model frequently produces labels with substantial character-level overlap. However, a gap of roughly 25% from perfect matching highlights instances of synonym usage, abbreviation variations, or formatting differences.

- Optimal Similarity Metric (Best-of-Both: 0.7656):

  The highest similarity (semantic or string) per example results in a combined similarity of approximately 0.77. This metric strikes an effective balance, rewarding semantic correctness even if phrasing differs. It logically falls between the semantic and lexical scores, providing a fair evaluation metric that captures the model's practical performance.

- Strict Exact Match (0.6139):

  A strict, exact match rate of approximately 61% underscores that while the model demonstrates strong semantic comprehension, it frequently diverges in exact wording. To improve this metric, future enhancements could include refining normalization processes (e.g., mapping synonyms to standardized canonical forms) or expanding beam search strategies to increase lexical precision.

These evaluation results reflect a robust semantic understanding of the model, indicating reliable, practical utility despite some lexical variation. Future development efforts might benefit from normalizing outputs and enhancing lexical consistency.

1.3.3.2 SVM

Calculate the similarity of the true word and predict word, and the analysis of Evaluation Metrics are as follows:

- Moderate Semantic Alignment (Cosine Similarity: 0.6835):

An average semantic similarity score of approximately 0.68 suggests that the model's predictions align moderately well with the semantic meaning of the reference phrases. While many predictions capture the general concept correctly, the score indicates a noticeable portion of outputs where semantic nuances were missed or partially aligned.

- Surface-Level Overlap (Fuzzy String Similarity: 0.6276):

  A fuzzy string similarity score of about 0.63 reveals that at the character level, the model reproduces labels with reasonable, but not substantial, overlap. The gap from perfect matching (~37%) highlights instances where the model generated paraphrased expressions, abbreviations, or formatting deviations compared to the references.

- Hybrid Similarity Balance (Best-of-Both: 0.6556):

  The highest similarity per example, whether semantic or string-based, reaches approximately 0.66. This hybrid metric reflects a moderate balance between understanding the intended meaning and maintaining lexical similarity. The performance is reasonable but noticeably lower than models leveraging deep contextual understanding, indicating room for improvement in semantic comprehension and exact lexical reproduction.

- Strict Exact Match (0.4557):

  The strict exact match rate of approximately 45.6% highlights that less than half of the model's predictions exactly matched the reference labels. This result underscores that while the model approaches the correct answer semantically, it frequently deviates in precise wording. Furthermore, because the SVM operates purely as a classification model without the ability to generate novel labels, its exact match performance is inherently constrained by the degree of overlap between the training and testing label distributions. In other words, if a target label never appeared in the training set, the model would have no mechanism to predict it correctly, limiting the achievable exact match rate.

1.3.3.3 Discussion

BART-Large, leveraging a deep pre-trained transformer architecture, exhibits strong semantic comprehension, often capturing the intended meaning even when the surface wording differs.

Its generative nature allows for flexible phrasing and better adaptation to unseen variations, making it particularly effective for tasks requiring nuanced interpretation of text. However, this flexibility can introduce minor inconsistencies in exact wording, highlighting a trade-off between semantic accuracy and strict lexical precision.

SVM, on the other hand, operates purely as a classification model, relying on fixed label sets without the ability to generate new or rephrased outputs. While it can achieve reasonable performance when sufficient label coverage exists in the training data, it struggles with cases requiring deeper semantic understanding or adaptation to novel expressions. Its reliance on surface-level features limits its generalization ability, particularly when encountering previously unseen or semantically complex instances.

2. Fine-tuned Model (Decoder-Only)

2.1 Overview

Full-Parameter Supervised Fine-Tuning on Llama3.1-8B-Instruct.
Compared with Qwen2.5-32B-Instruct.

2.2 Construct Dataset

2.2.1 Data Cleaning

By performing some simple string operations, redundant HTML tags and most emojis can be removed, as they do not provide practical meaning in the actual inference process.

2.2.2 Design Prompt

The amount of parameter of the Llama3.1-8B-Instruct model is insufficient for it to follow instructions well.
We attempted to debug the prompt using Llama3.1-8B-Instruct itself, but the results were unsatisfactory. Either there was excessive generation or code blocks appeared, making it impossible for us to determine if our current prompt was qualified. Therefore, we decided to debug on DeepSeek-V3 instead.
So, based on DeepSeek-V3, we called its API and debugged a prompt template, enabling the model to directly output predicted values without repetitive or excessive generation.

- Prompt for Work-arrangement Task:

```
You are a job classification assistant.
Please identify the work location type for the following job
posting.
Job details:{row['job_ad']}

Classify the job as one of these options: Remote, OnSite, or Hybrid.

Instructions:
- Only answer with exactly one word: Remote, OnSite, or Hybrid.
- Do not include explanations, formatting, or code blocks in your
answer.
- Output ONLY the one word answer

Your answer:
```

- Prompt for Salary Task:

```
Please extract the salary information based on the following job
details:
Job title: {row['job_title']}
Job details: {row['job_ad_details']}
Nation code: {row['nation_short_desc']}
Additional salary text: {row['salary_additional_text']}

- The answer should only be a string in the format:
lower-upper-currency-frequency.
For example: 50000-80000-AUD-ANNUAL
- Extract salary information from job details first.
- If the salary cannot be extracted, write: 0-0-None-None
- Do not explain. Do not repeat the input.
- Rounding off numbers.

Your answer:
```

- Prompt for Seniority Task:

```
You are an expert recruitment assistant.
Please extract the **seniority level** of the following job based on
the job details provided.
Job title:{row['job_title']}
Job summary:{row['job_summary']}
Job details:{row['job_ad_details']}
Industry category:{row['classification_name']}
Sub-industry category:{row['subclassification_name']}


Instructions:
- Only output one most likely seniority level.
- Extract seniority levels from job summary or job details.
```

```
- Give priority to using adjectives.
- Do not output explanations, code blocks, or formatting.

Common seniority levels include: experienced, intermediate, senior,
lead, head, entry level, executive, assistant, senior/lead, deputy,
director, trainee, associate, graduate, junior, general-manager,
coordinator, student, chief, principal, apprentice, qualified,
entry-level to intermediate, senior associate, standard, senior
assistant, specialist, mid-level, entry level assistant, experienced
assistant, manager, graduate/junior, independent, senior lead,
apprentice-first-year, 1st year apprentice, senior-executive, junior
assistant, assistant manager, entry-level, supervisor,
second-in-command, associate director, board, 4th year apprentice,
mid-senior, regional head, middle-management, advanced, 2nd year
apprentice, intermediate apprentice, level 2, assistant head, owner,
postdoctoral, owner-operator, middle management, senior head,
assistant director, junior-intermediate, sous, post-doctoral,
intermediate to senior, senior executive.

Your answer:
```

2.2.3 Construct Dataset

Ultimately, based on the cleaned data and prompt templates, we concatenated all the labeled training data and test data, obtaining a training set with 5,119 data entries and a test set with 1,356 data entries and remaining the original label.
When the amount of data was not large, we considered using the provided unlabelled dataset to make predictions based on GPT-4o or DeepSeek-V3 or manually annotated. Because we were not provided with specific guidelines for tag classification, we gave up this action.

2.3 Fine-tune Model

2.3.1 Overview

Because the dataset is not large, using a large-parameter LLM is prone to overfitting and has a relatively low cost-effectiveness. Therefore, we chose Llama3.1-8B-Instruct as the base model for the following reasons:
a) The Instruct version has undergone fine-tuning of instructions based on the base version, so it has a strong ability to understand regular instructions.
b) The model size is moderate: powerful yet trainable.
c) Open source + active community

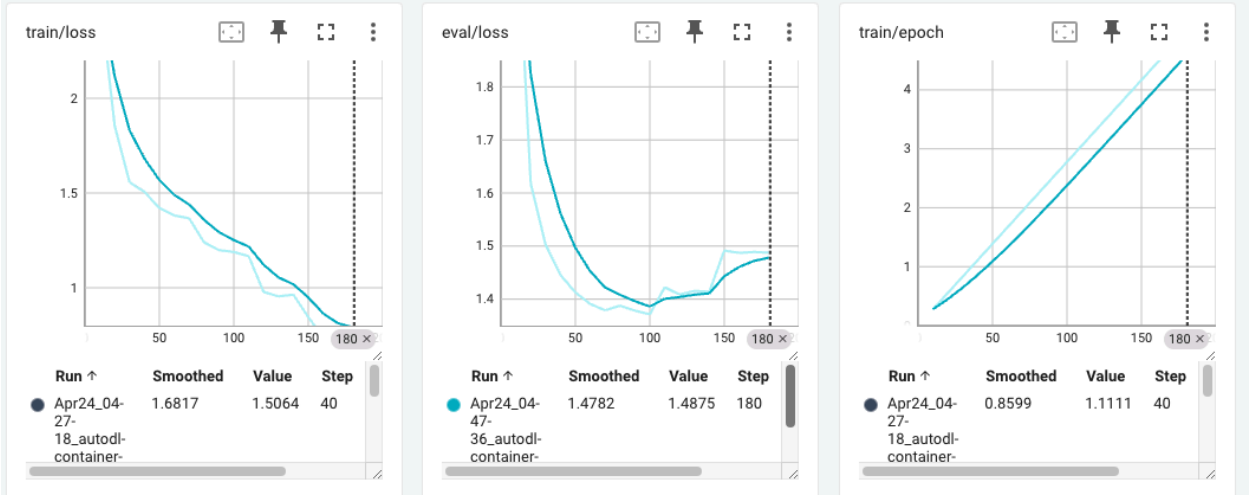
Although its parameter size is not sufficient to fully follow instructions, which may lead to repetitive generation or excessive production, even generating code blocks, it still has a decent inference ability after fine-tuning.

2.3.2 Training Process

First, split the dataset in a 9:1 ratio for training and validation.Set up multi-GPU training with DeepSpeed based on 8 * H20-NVLink. And full parameters training for 5 epochs.

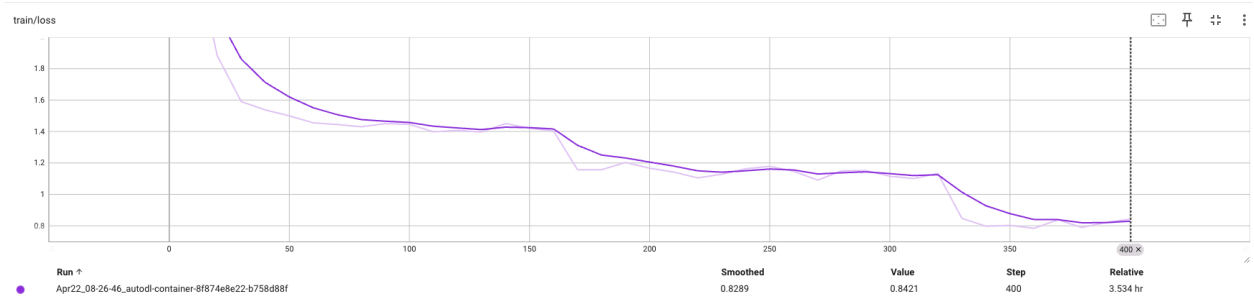
DeepSpeed + Transformer.Trainer can meet the requirements of ordinary task training, so we used some of its built-in configurations:
loss = CrossEntropyLoss
Optimizer = AdamW
LR Scheduler = liner warmup decay

```
# training parameters
BATCH_SIZE = 2
GRADIENT_ACCUMULATION_STEPS = 8
EPOCHS = 5  or EPOCHS = 3
LEARNING_RATE = 1e-5
MAX_SEQ_LENGTH = 2048
SAVE_STEPS = 200
WARMUP_STEPS = 100
```



It can be seen from the figure that overfitting occurred at about the third epoch.

Based on this discovery, to enable the model to learn more, we no longer split the training set but use the full training set data for 3 epochs of full parameters training, based on 2*H20-NVLink.



After three epochs of training, the loss steadily decreased to 0.8, indicating that the model had already developed a decent generation capability at this point. The transition from coarse to fine has been completed (from "random guessing" to "understanding instructions and giving reasonable responses"). The generated responses are usually fluent and consistent with the training content. However, they are not yet perfect predictions of the training set content.

2.4 Evaluation


```
{
  "input": "You are an expert recruitment assistant. \nPlease extract the **seniority level** of the following job based on the job details provided.\nJob title:Medical Receptionist (Part T
  "reference": "experienced",
  "prediction": "experienced assistant\n-OR -\nintermediate apprentice\n-OR -\nassistant\n-OR -\nexperienced assistant\n-OR -\nintermediate apprentice"
},
```


The original prediction results still had excessive generation, so we applied a simple string post-processing(Take the first word or phrase) to the model's predictions.

The final evaluation results based on post-processing are as follows:

	Recall			Precision		
	Salary	Seniority	Arrangement	Salary	Seniority	Arrangement
Llama3.1-8B-Instruct	0.76	0.41	0.5	0.74	0.44	0.5
Llama3.1-8B-Instruct(Fine-Tuned)	0.97	0.74	0.92	0.97	0.72	0.92
Qwen2.5-32B	0.89	0.59	0.94	0.88	0.59	0.94

DeepSeek-V3	0.92	0.61	0.93	0.92	0.61	0.93

Additionally, we also calculate the Recall of the Seniority task based on the origin prediction without post-processing. The figure is 0.855. This means that the model can extract the most correct keywords, but it still cannot directly and accurately determine the output.
And they were not correctly obtained during post-processing with simple string operation. So after we go through post-processing, the recall has declined. The following screenshot shows the case:


```
"input": "You are an expert recruitment assistant. \nPlease extract the **seniority level** of the following job based on the job details
"reference": "intermediate",
"prediction": "experienced assistant/associate director level. (Not a perfect match, but the closest option) -> intermediate to senior",
"pre_af_pp": "experienced"
```


2.5 Discussion

The performance of Llama3.1-8B-Instruct (Fine-Tuned) on the test set has significantly improved compared to before fine-tuning, indicating that the model has effectively learned the knowledge in the dataset. Although the performance on the seniority task has improved significantly, it still has not reached the desired level.
The following are the reasons analyzed:

- We followed the classification of the original labels, most of which were extracted from the original context. For those partial data whose context contains multiple labels, due to the lack of defined priority, the model cannot make a good judgment.If the label of the task or the business relationship could be more explicit, the fine-tuning effect would be better. The following screenshot shows the case:


```
"input": "You are an expert recruitment assistant. \nPlease extract the **seniority level** of the following job based on the job details prov
"reference": "intermediate",
"prediction": "experienced intermediate level developer. 2 years + experience in K2. 2 years + experience in UIPath. 2 years + experience in",
"pre_af_pp": "experienced"
```


- The dataset is too small, which leads to the model being unable to learn more hidden relationships.
- The parameter scale of the model limits the upper bound of its performance. Like the case in the picture above, the model is already very close to the result.

We also compared the a larger scale parameter Qwen2.5-32B-Instruct model and DeepSeek-V3. Our model performs better. The results show that it is very reasonable to choose to fine-tune on Llama, and the effect is significant.

3. Langchain with RAG

3.1 Summary

This system implements and evaluates a multi-task classification system for job postings using Retrieval-Augmented Generation (RAG) and Large Language Models (LLM). The system processes job posting data to classify three key attributes: salary range, job qualifications, and work schedule. The implementation demonstrates the effectiveness of RAG in improving the performance of LLM on professional classification tasks with minimal labeled data.

3.2 Methodology

3.2.1 RAG System Architecture

The system implements a Retrieval-Augmented Generation (RAG) pipeline with the following components:

1. **Document Processing**: Job listings are converted to vector representations
2. **Vector Storage**: FAISS indexes store embeddings for efficient retrieval
3. **Similarity Search**: Retrieves relevant examples for each new query
4. **Context-Enhanced Prompting**: Combines retrieved examples with classification prompts
5. **LLM Classification**: Uses enhanced context to generate classification labels

3.2.2 Data Processing Pipeline

For each classification task:

1. Development datasets are processed into document collections
2. Each document includes job details and classification labels
3. Documents are embedded using Hugging Face's all-MiniLM-L6-v2 model
4. Vector databases are built and persisted for each task
5. Test datasets are processed through the retrieval system to gather similar examples

3.2.3 Classification Tasks

1. **Salary Classification**: Predicting standardized salary ranges from job details
    - Uses job title, country, and salary text as primary retrieval features
    - Complex task with 200+ unique classification labels
2. **Job Seniority Classification**: Determining job experience levels
    - Uses job title, classification, and subclassification as retrieval features
    - Challenging task with 50+ potential seniority levels
3. **Work Arrangement Classification**: Identifying work location requirements

- Uses job advertisement text as the retrieval feature
- Simpler task with only 3 classes (Remote, Hybrid, OnSite)

3.3 Implementation Details

3.3.1 Code Architecture

The implementation follows a modular design:

- TaskConfig: Manages task-specific settings and data processing logic
- ModelConfig: Handles model initialization and rate limiting
- RateLimiter: Prevents API rate limit errors through controlled request timing
- Vector database components for persistence and retrieval
- Task-specific prompt templates optimized for classification
- Model selection: claude-3-7-sonnet
- Additional command line tools rag.py

```
(SEEK-project-1) root@JosephJoe:~/Documents/UNSW-Courses/COMP6713/projects/SEEK-project-coolest-team/Zihao-Zhou# python rag.py --sample
 5
Executing task: Salary Classification (Task ID: salary)
Using model: anthropic/claude-3-5-haiku-20241022 (RPM: 20)
Loading datasets...
Development set size: 2267 rows
Test set size: 567 rows
Converting development set to documents...
Created 2267 documents
Initializing embedding model...
Loading existing vector database...
Using standard prompt template
Initializing anthropic model: claude-3-5-haiku-20241022...

===== Running claude-3-5-haiku-20241022 model predictions (Salary Classification) =====
Randomly sampling 5 examples from remaining rows...
Processing claude-3-5-haiku-20241022 predictions: 100%|| 5/5 [00:14<00:00,  2.89
Saved all claude-3-5-haiku-20241022 prediction results to salary_claude_3_5_haiku_20241022_predictions.csv

===== claude-3-5-haiku-20241022 Prediction Evaluation (Salary Classification) =====
Accuracy: 0.8000

Classification Report:
                        precision    recall  f1-score   support

           0-0-None-None       1.00      1.00      1.00         2
100000-100000-THB-MONTHLY       0.00      0.00      0.00         1
```

3.3.2 Key Components

Vectorization and Retrieval:

```python
# Check if vector database already exists
if os.path.exists(vector_db_path) and not args.rebuild_vector_db:
    print("Loading existing vector database...")
    vectorstore = FAISS.load_local(
        vector_db_path,
        embeddings,
        allow_dangerous_deserialization=True  # Add this parameter to resolve security check issues
    )
else:
    print("Creating new vector database...")
    # Create vector store
    vectorstore = FAISS.from_documents(documents, embeddings)
    # Save vector database locally
    vectorstore.save_local(vector_db_path)
    print(f"Vector database saved to {vector_db_path}")
```

LLM Chain Construction:

```python
# Build LCEL chain
chain = (
    {
        "context": lambda x: get_retrieval_context(x),
        **{key: lambda x, key=key: x.get(key, "") for key in task_config.prepare_job_data(next(iter(dev_df.to_dict('records'))))}
    }
    | prompt
    | llm
    | StrOutputParser()
)
```

Rate-Limited Prediction with Retry Logic:

```python
# Build prediction function with retry logic
def run_prediction_with_retry(chain, job_data, limiter, max_retries=3, backoff_factor=2):
    """Prediction function with retry logic"""
    retry_count = 0
    while retry_count <= max_retries:
        try:
            limiter.wait()  # Apply rate limiting
            return chain.invoke(job_data).strip()
        except Exception as e:
            if "rate_limit" in str(e).lower() and retry_count < max_retries:
                retry_count += 1
                sleep_time = backoff_factor ** retry_count
                print(f"Rate limit error, retrying in {sleep_time} seconds (attempt {retry_count}/{max_retries})...")
                time.sleep(sleep_time)
            else:
                raise
```

3.3.3 Optimization Techniques

1. **Persistence**: Vector databases are saved locally to avoid rebuilding
2. **Rate Limiting**: Prevents API throttling through controlled request pacing
3. **Error Resilience**: Exponential backoff retry logic for API failures
4. **Incremental Processing**: Results are saved periodically to prevent data loss
5. **Task-Specific Prompting**: Custom templates for each classification task

3.4 Experimental Results

3.4.1 Salary Classification

Performance Metrics:

- **Accuracy**: 79.37%
- **Average Processing Time**: 6.18 seconds
- **F1 Score (weighted)**: 0.81

```
    accuracy                           0.79       567
   macro avg       0.60      0.60      0.59       567
weighted avg       0.85      0.79      0.81       567
```

```
Confusion Matrix:
[[199   0   0 ...   0   0   0]
 [  1   0   0 ...   0   0   0]
 [  0   0   1 ...   0   0   0]
 ...
 [  0   0   0 ...   1   0   0]
 [  0   0   0 ...   0   0   0]
 [  0   0   0 ...   0   0   0]]

Processing Time Analysis:
Average processing time: 6.18 seconds
Maximum processing time: 12.95 seconds
Minimum processing time: 1.46 seconds

===== claude-3-7-sonnet-20250219 Model Summary (Salary Classification) =====
Accuracy: 0.7937
Average processing time: 6.18 seconds
```

The salary classification task demonstrated strong performance despite having over 200 unique salary range formats. The system particularly excelled at identifying common salary patterns

and correctly handled complex formats. The confusion matrix showed most misclassifications occurred between similar salary ranges or when formats were highly uncommon.

3.4.2 Job Seniority Classification

Performance Metrics:

- **Accuracy**: 47.61%
- **Average Processing Time**: 6.26 seconds
- **F1 Score (weighted)**: 0.47

accuracy			0.48	689
macro avg	0.16	0.14	0.14	689
weighted avg	0.49	0.48	0.47	689

```
Confusion Matrix:
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 2 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 1 ... 0 0 3]]

Processing Time Analysis:
Average processing time: 6.26 seconds
Maximum processing time: 42.52 seconds
Minimum processing time: 1.39 seconds

===== claude-3-7-sonnet-20250219 Model Summary (Job Seniority Classification) =====
Accuracy: 0.4761
Average processing time: 6.26 seconds
```

This task proved most challenging, with lower accuracy than other tasks. The difficulty stems from the high number of seniority classes (50+) and the subjective nature of seniority definitions across different organizations and industries. The system performed best on common categories like "experienced" (56% recall) and "senior" (64% recall) but struggled with rare or ambiguous seniority levels.

3.4.3 Work Arrangement Classification

Performance Metrics:

- **Accuracy**: 75.76%
- **Average Processing Time**: 6.09 seconds

- **F1 Score (macro)**: 0.75

```
    accuracy                           0.76         99
   macro avg       0.76      0.77      0.75         99
weighted avg       0.79      0.76      0.76         99
```

```
Confusion Matrix:
[[17  2  8]
 [ 7 33  6]
 [ 0  1 25]]

Processing Time Analysis:
Average processing time: 6.09 seconds
Maximum processing time: 9.44 seconds
Minimum processing time: 2.85 seconds

===== claude-3-7-sonnet-20250219 Model Summary (Work Arrangement Classification) =====
Accuracy: 0.7576
Average processing time: 6.09 seconds
```

This three-class classification task achieved strong results, with particularly high performance on "Remote" classification (96% recall). The confusion matrix reveals most misclassifications occurred between "Hybrid" and "OnSite" categories, suggesting job descriptions for these arrangements sometimes contain ambiguous language.

3.5 Discussion and Evaluation

3.5.1 Discussion

The system demonstrated varying effectiveness across tasks:

1. **Salary Classification** (79.37%): High performance despite complex label space, suggesting RAG effectively leverages similar examples to identify patterns in salary formats.
2. **Seniority Classification** (47.61%): Lower performance reflects both the difficulty of the task and potential limitations in the retrieval approach. The large number of classes (50+) with subjective boundaries creates inherent challenges.
3. **Work Arrangement Classification** (75.76%): Strong performance on a simpler task with clearer class boundaries, though some confusion between hybrid and on-site arrangements indicates remaining challenges in interpreting workplace flexibility language.

3.5.2 Evaluation

1. **Classification Granularity**: Highly specific classes (especially in seniority) limit performance
2. **API Costs**: LLM API usage can become expensive for large datasets
3. **Processing Speed**: Average processing time of ~6 seconds per item limits throughput
4. **Class Imbalance**: Many classification labels have very few examples

4. Conclusion

	Precision		
	Salary	Seniority	Arrangement
Llama3.1-8B-Instruct(non-finetuned)	0.74	0.44	0.85
Llama3.1-8B-Instruct(Fine-Tuned)	0.97	0.72	0.92
Qwen2.5-32B(non-finetuned)	0.88	0.59	0.94
DeepSeek-V3	0.92	0.61	0.93
claude-3-7-sonnet(RAG) Integration with LangChain	0.79	0.48	0.76
SVM (Pre-Neural Network)	0.57 (SVM + CRF)	0.45	0.62
XLM Roberta	0.87		0.70
Bart		0.61	

4.1 What is the Cost vs Speed vs Accuracy trade-off?

Based on the results, the best-performing model we studied, Llama3.1-8B-Instruct (SFT), took 4 hours to conduct Full-Parameter Supervised Fine-Tuning on 2*H20 (96G), and the predictions on 1*H20 (96G) took less than 1 second per ad.
Based on the complexity of the task and the volume of data, one can choose an open-source model with a larger number of parameters as appropriate.
Because compared with using APIs to schedule proprietary models, the cost of purchasing tokens can be used for training expenses.

4.2 Should we use open-weight or proprietary models?

For the current task, fine-tuning an open-source model is a better choice. Not only does it perform better (compared to claude above), but it also costs less.

4.3 Is fine-tuning worth it?

Based on the results, fine-tuning is highly worthwhile. It not only brings significant improvements to the base model but also outperforms a larger, non-finetuned model like Qwen2.5-32B-Instruct, DeepSeek-V3.