The University of New South Wales

COMP6713
SEEK Project

By

MA4

Abhishek Moramganti (z5421958)

Aleksandra Kalinic (z5312702)

Ankur Bhalotia (z5232739)

Arvind Chandrasekaran (z5503831)

Madina Khalmatova (z5515915)

Contents

Executive summary

This project addresses SEEK's need to efficiently extract structured information from job advertisements—specifically salary details, work arrangements, and seniority levels. We designed solutions combining rule-based approaches and neural models, evaluating them on SEEK's provided datasets. Initial baselines using NER and regex-based methods underperformed due to the complexity and implicit nature of the tasks, motivating a shift to language models.

We explored a range of models—small (T5, Qwen), medium (Gemma3, Qwen2.5), and large (DeepSeek-V3, Gemini 1.5 Flash, GPT-3.5)—using techniques like standard prompting, Role-Based Incremental Coaching (RBIC), few-shot prompting, and fine-tuning. Our results showed that large models excelled even without fine-tuning, while fine-tuning significantly boosted the performance of smaller and medium models, especially for extraction tasks like salary prediction. Open-weight models like DeepSeek and Qwen performed competitively against proprietary models like Gemini, especially when fine-tuned.

Key findings include: (1) Fine-tuning greatly enhances extraction and classification on small datasets; (2) Proprietary models offer ease of use but limit customisation and transparency; and (3) Open-source models provide better long-term flexibility and control. We recommend SEEK invest in fine-tuning open-weight models for maximum adaptability, privacy, and performance.

Chapter 1

Problem definition

Given the influx of job seekers and employers in the market, this has propelled an increase in job advertisements on various employment platforms. Thus, the process of aligning relevant information for prospective job seekers in an efficient manner has become a pertinent task.

The following project proposal involves utilising labelled and unlabelled datasets from SEEK to extract: salary, work arrangements and seniority, which are crucial for filtering relevant job applications to seekers. The goal is to extract three key attributes from job advertisements:

Extraction

Task 1: Performing information extraction from job ads to obtain salary related information in the format *min-max-currency-frequency*.

Classification with Inference

Task 2: Infer seniority information, from a set of given labels.

Task 3: Determine work arrangements of a job from its description, falling into categories {"Remote, "Hybrid", "OnSite"}

The implementation of the above tasks has been driven by understanding the current accuracy, cost and speed of both open source and proprietary models, to tailor a compatible solution for SEEK. An exploration of these models was conducted into three classes:

1. Small Sized Models
    a. T5: A encoder-decoder transformer model capable of handling a range of NLP tasks through text-to-text transformation [1]

2. Medium Sized Models

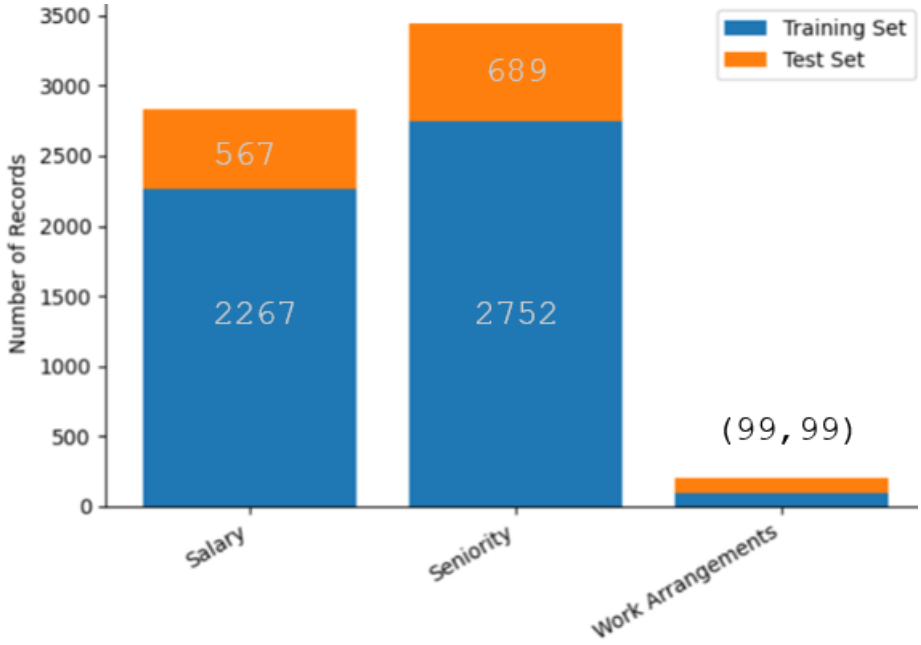
    a. Qwen 2.5 an open source "transformer based decoder", fit for a diverse range of tasks [2]
    b. Gemma3 an open source "decoder only" model [3]
3. Large Sized Models
    a. DeepSeek
    b. Gemini 1.5 Flash

Chapter 2

Exploratory Data Analysis

Based on the provided datasets - salary, seniority, and work arrangements - we identified observations within the development and test sets that led to several crucial assumptions, discussed below.

The total number of entries in each test and training dataset for salary, seniority and work arrangements can be shown with this graph.



Salary, Seniority and Work-Arrangements had multiple fields with some like job_id (id in the case of work arrangements) and job_ad_details (job_ad in the case of work arrangements) being common. There were some fields unique to each of the categories as seen below.

salary.csv

- **job_id**: Unique identifier for each job listing.
- **job_title**: Advertised position title.
- **job_ad_details**: Full HTML/text of the job description.
- **nation_short_desc**: Two- or three-letter country code for the job location.
- **salary_additional_text**: Any extra salary notes or placeholders not in the structured range.
- **y_true**: Ground-truth salary in min-max-currency-frequency format.

seniority.csv

- **job_id**: Unique identifier for each job listing.
- **job_title**: Advertised position title.
- **job_summary**: Brief one- or two-sentence overview of the role.
- **job_ad_details**: Full HTML/text of the job description.
- **classification_name**: Broad occupational or industry category.
- **subclassification_name**: More specific sub-category within that classification.
- **y_true**: Annotated seniority level (e.g., junior, intermediate, senior).

work_arrangement.csv

- **id**: Unique identifier for each job listing.
- **job_ad**: Job advertisement text, including title and abstract.
- **y_true**: Annotated work arrangement label (OnSite, Remote, or Hybrid).

2.1 Salary

Non-English Entries
Salary has 433 non-english entries in the training set and 109 non-english entries in the test set. Since we are using multilingual LLMs, we leave these entries as it is in the pre-processing.

HTML
Salary dataset also had HTML tags, emojis, and duplicate rows which were removed in the pre-processing to prevent any noise while performing the analysis.

salary_addtional_text
Within the "salary_additional_text " column,, there were 3 types of NULL values:

1. '-'
2. 'NaN'
3. <empty> (Nothing is present)

These NULL values had an interesting correlation with the y_true:

- It was observed that for every '-' (234 entries) in salary_addtional_text, there is no salary mentioned in the job ad and the label is '0-0-None-None'.
- In contrast, for every <empty> (692 entries) in salary_addtional_text, there is always salary mentioned in the job ad and thus present in the label.
- As for the case 'NaN' (602 entries), 8.82% (61) entries had a salary mention as for the remaining 89.86% (541) entries there was no salary mention.

Based on the above observations, to ensure that our model does not consider the presence of ‘NaN’ or <empty> in salary_additional_text to be a hint for the presence or absence of salary mentioned in the job_ad_details, we must treat both these values as None.

However, ‘-’ holds a valuable pattern of salary label being ‘0-0-None-None’, hence why this value is preserved.

Ultimately, we treated all of these cases uniformly by replacing every null in salary_additional_context with the placeholder “empty.” This prevents our model from learning that a missing salary_additional_context necessarily implies a salary mention in the job ad. Although the same pattern appeared in our test set, this strategy ensures the model remains robust in real-world situations where that correlation may not hold.

Moreover, there were around 33.2% (753) entries in the training set with a non NULL value in salary_addtional_text and but with no salary mention.

2.2 Seniority

In seniority, we observed that the seniority information may not be mentioned directly in the job advertisement and needs to be inferred by understanding the entire text. Additionally, we understand that seniority does not have a fixed label set like work arrangements, since there are some labels present in the test data set which were not observed in the training dataset.

Seniority training set had around 64 unique labels but with some similar labels like “middle management” and “middle-management”, the testing set also had 39 unique labels, with 8 of them not having exact matches in the training dataset -

Test Label	**Equivalent Training Label**
junior-to-intermediate	junior-intermediate
associate-director	associate director
third-year apprentice	None (1st, 2nd and 4th year apprentice is present)
early-career	None (entry-level is a close label)
cadet	None
senior/principal	senior & principal separately present
intermediate management	None (middle-management , middle management are close labels)
retired	None

We unified all the similar labels present in the training set to ensure some consistency while fine tuning.

Seniority also had HTML tags and non ASCII + punctuation characters that needed to be processed.

2.3 Work Arrangements

Work arrangements was a smaller data set containing only 99 rows, which has downstream effects in performance of model finetuning, discussed further. The dataset contained three labels : {“OnSite”, “Remote” and “Hybrid”} that were extracted based on the “job_ad” column. The training set has a distribution of 44.4% OnSite, 34.3% Remote and 21.2% Hybrid and the test has the distribution of 46.46% OnSite, 26.26% Remote and 27.27% Hybrid.

2.4 Data Cleaning

Preprocessing the data to a standardised format, is crucial for accuracy of both extraction and inference tasks. By removing convoluted characters, emojis and irrelevant details ambiguity is reduced allowing for more accurate model performance. The exact workflow can be found within “MA4_Preprocessing.ipynb” notebook.

2.4.1 Salary

- Duplicate rows were removed
- HTML tags were removed, in order to focalise on text content
- Emojis were removed
- Empty (null) and NaN rows for salary_additional_text column were given a single label

2.4.2 Seniority

- Remove HTML tags and emojis

2.4.3 Work Arrangements

- No steps of pre-processing were added to work arrangements.

Chapter 3

Baseline

A baseline model was conducted as a comparison metric between non-neural and neural approaches, as well as to provide a simple initial pass over the data. Our results conclude a poor performance across both inference and classification tasks, with only moderate success in extracting work arrangements, as shown in Figure 3.1. Our implementation for the baseline involved a combination of rule-based patterns based on regex extracted in job descriptions and simple statistical models.

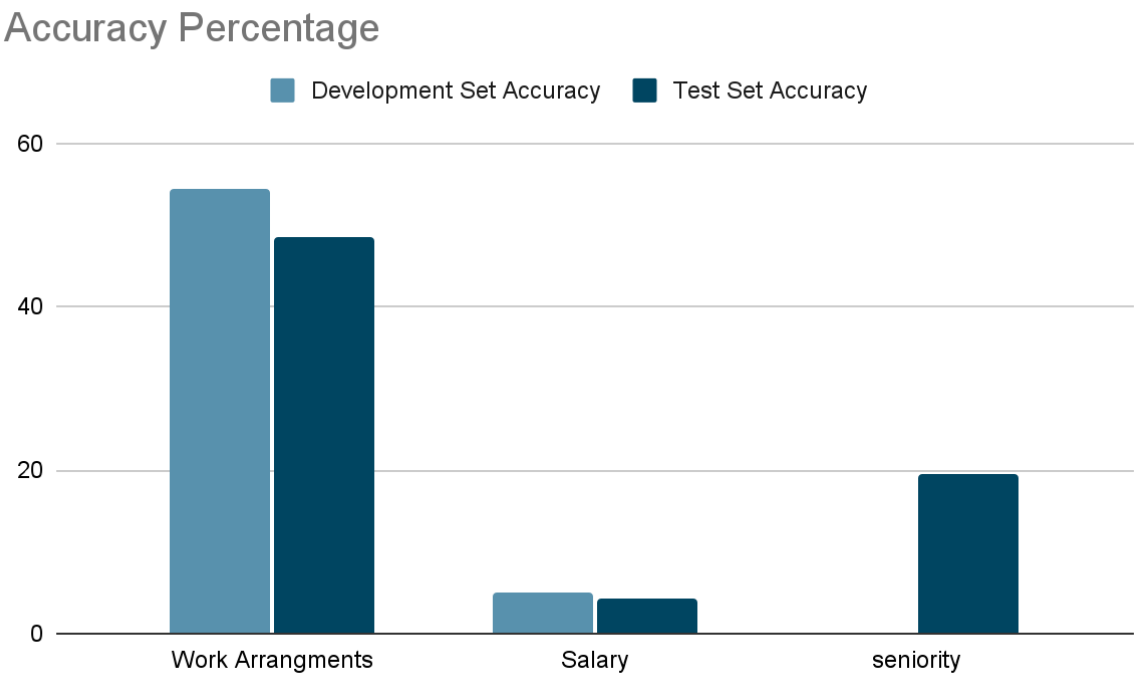


Figure: 3.1 Baseline Results in bar plot

	Work arrangements	Salary	Seniority
Overall accuracy	51.52	4.94	19.44
Development set accuracy	54.54	5.07	-
Test set accuracy	48.48	4.41	19.44

Figure: 3.2 Baseline Results in table format

3.1 Work arrangements Implementation

A rule-based approach was used to classify job postings into work arrangement categories: Remote, Hybrid, and On-site. This was done with mostly rule-based patterns.

1. **Preprocessing:**
    - Convert job ad text to lowercase.
    - Remove unnecessary punctuation while retaining words, spaces, and hyphens.
2. **Keyword Lists:**
    - Defined specific keywords for each category:
        - **Remote:** e.g., "remote", "work from home".
        - **Hybrid:** e.g., "hybrid", "partially remote".
        - **On-site:** e.g., "on-site", "in person".
3. **Keyword Matching:**
    - Count the number of times keywords from each category appear using exact word matching.
4. **Classification Logic:**
    - Assign the category with the highest keyword count.
    - In case of a tie, randomly select among the top labels to ensure fair classification.
    - In case there is no label found, give a random guess

      Note: the random chance of getting an option correct is 0.33.

3.2 Salary extraction Implementation

The following rule-based approach was utilised for salary extraction. To accurately extract salary information from job advertisement text the baseline utilised a combination of spaCy's Named Entity Recognition (NER) capabilities and regular expression-based post-processing.

1. **Text Processing:**
    - The job ad text is processed using the en_core_web_sm model from spaCy.
    - Entities such as MONEY and CARDINAL are identified to detect potential salary information.
2. **Keyword and Pattern Matching:**
    - Regular expressions are used to enhance salary extraction, especially for identifying currency symbols (e.g., $, €, £) and numerical patterns (including ranges like "30k–40k" or "between 50,000 and 70,000").

- Both explicit salaries and salary mentions embedded in numeric modifiers are captured.

3. **Salary Confidence Levels:**
    - Salaries are categorized into:
        - **Higher Confidence:** Detected through spaCy's entity recognition.
        - **Lower Confidence:** Detected through token-level rules and currency/numerical pattern checks.

4. **Normalization:**
    - Detected salaries are cleaned:
        - "k" and "K" suffixes are expanded to "000".
        - Non-numeric and irrelevant characters are removed, keeping important terms like "to" and "-".

5. **Salary Selection Strategy:**
    - Priority is given to higher-confidence extractions containing clear salary ranges.
    - If unavailable, numeric salaries are extracted and a range is computed from the minimum and maximum detected values.
    - If no valid salary is found, a default value of "0-0-None-None" is returned.

6. **Final Output Formatting:**
    - The extracted salary is formatted as: minsalary-maxsalary-currency-frequency
    - Currency is identified based on nearby symbols or text (e.g., USD, GBP, EUR).
    - Payment frequency is inferred (default is YEARLY, with adjustments for mentions like "monthly" or "hourly").

3.3 Seniority extraction Implementation

To classify job advertisements into specific seniority levels based on exact keyword matching against a predefined label set. This is created using the development set.

1. **Text Preprocessing:**
    - The input job ad text is lowercase.
    - Non-alphanumeric characters (except spaces, slashes, and hyphens) are removed to normalize the text for consistent matching.

2. **Label Set Preparation:**

- A predefined set of seniority labels (e.g., "junior", "senior-manager", "entry-level") is used as the reference.
- Each label is matched exactly against the preprocessed text.

3. **Keyword Matching:**

    - For each label, the number of exact matches in the text is counted using regular expressions.
    - This ensures that only complete and accurate label mentions are considered.

4. **Classification Logic:**

    - The label(s) with the highest number of matches are selected.
    - If multiple labels have the same highest count, a random choice is made among them.
    - Random selection also occurs if no labels are matched (i.e., max count is 0), ensuring that a label is always returned.

Chapter 4

Neural Approach: RBIC and Fine-tuning

We propose using Language Models after observing the drawbacks of the non-neural baseline and the exploratory data analysis.

4.1 Exploratory Data Analysis (EDA) Observations

Inference Tasks Identified

- Working Arrangement: Often not explicitly stated; must be inferred from context, not just detected via keywords.
- Seniority: Frequently absent; must be deduced from indirect hints (e.g., "lead a team," "entry-level responsibilities").

Label Gaps and Ambiguity

- Some test labels (e.g., junior-to-intermediate, associate-director, third-year apprentice, early-career, cadet, senior/principal, intermediate management, retired) are absent or inconsistent in the training set. There are cases where only close equivalents or no equivalent exist in the training data.

4.2 Baseline (Non-Neural) Model Flaws

1. Keyword/NER Limitations

- False Positives: Classic NER and rule-based methods fail with context-sensitive phrases (e.g., the word “senior” in “working with a senior team member” does not refer to the job’s seniority).
- False Negatives: They miss implicit information (e.g., inferring remote work from “fully flexible work environment” when the keyword "remote" isn’t present).
- Label Ambiguity: They can’t handle mapping “associate-director” (test) to “associate director” (train), or infer that “third-year apprentice” relates to apprenticeship levels not in the training set.

2. No Generalisation or Reasoning

- Can’t deduce unseen or novel labels.
- Can’t understand paraphrases or subtle implications (e.g., “cadet” ≈ “entry-level”).
- Fail on nuanced context: e.g., “we do not work remotely” is negative evidence for remote work.

4.3 Why Use Language Models?

1. Superior Contextual Understanding

- Language models leverage deep, pretrained world knowledge and can infer meaning from context—even if the explicit label is missing.
    - E.g., They can recognize that "leadership responsibilities" imply "senior," or "onsite" can be inferred from absence of remote/flexible indicators.

2. Generalization and Robust Mapping

- Language models can map test-time variants or synonyms to canonical labels (e.g., "junior-to-intermediate" → "junior-intermediate"), even if not seen during training.
- Can make intelligent guesses for previously unseen labels using language understanding.

3. Handling Nuanced Negations and Implicit Information

- Language models understand negation (e.g., "we do not work remotely" → not Remote).
- Can process implicit information (e.g., inferring "onsite" from "need you here onsite" even if "onsite" is not a recognized label).

4. Outperform on Real-World, Messy Data

- Unlike rule-based methods that break when the data isn't perfectly formatted or explicit, language models perform well on messy, realistic job ad language.

4.4 What is the best way to use Language Models?

In the project, we have explored different methods to provide concrete solutions to SEEK's problem.

We consider the following research questions to find the best way:

1. Is training data needed?
2. Is fine tuning worth it?
3. What is the right size of the model (model size has correlation with the speed)?
4. Are proprietary models better than open-source?

We have also used different language models, both open source and property to answer the research questions.

The approaches are as follows

1. Prompt Engineering
2. Improved Prompt Engineering (RBIC)
3. Fine-tuned Language Model

4.5 Prompt Engineering

We created a standard set of prompts (without providing any examples) for each of the tasks and fed these prompts to the following models -

1. Large - DeepSeek-V3 and Gemini 1.5 Flash
2. Medium - Gemma3
3. Small- T5 and Qwen

Open Source - DeepSeek-V3, Gemma3, T5 and Qwen
Proprietary - Gemini 1.5 Flash, GPT 3.5

Research Question being addressed:

1. By not providing any training data, we are able to understand whether we need high-quality training data or if pre-trained language models are good enough to do the job on their own.
2. We are also able to draw comparisons between a pre-trained model and a fine tuned model.
3. By having models of different sizes we can also have a comparison of running speed for open-source models.
4. Also, by having a variety of open-source and proprietary models, we are able to address another research question regarding whether open-source models are better in comparison to proprietary.
5. Lastly, by comparing this normal prompting approach with the structured prompting we can draw comparison to other structured but slower methods to perform prompt engineering.

1. **Salary Prompt Structure**

“You are an expert job ad annotator. Your task is to extract structured salary information from job descriptions in the format: min-max-currency-frequency. If salary is not found, return: 0-0-None-None.”

“
all the job ad details
job_title
job_ad_details
nation_short_desc
salary_additional_text

”

“Extract structured salary information from this job description in the format: min-max-currency-frequency. Respond in JSON:
{MinSalary: , MaxSalary: , Currency:, Frequency: }”
“If not provided explicitly, output 0 for MinSalary and MaxSalary, and "None" for Currency and Frequency.”
“If the salary is mentioned, always output a range, where MinSalary and MaxSalary can be equal.”

“Use nation_short_desc to determine the correct currency as 3 letters, and use an adverb for frequency (annual, monthly, daily or hourly).”

2. **Seniority Prompt Structure**

“You are an expert job ad annotator. Your task is to extract seniority information from job descriptions.”
“
Job ad details
job_title
job_summar:
job_ad_details
classification_name
subclassification_name
”
“ Infer the seniority label from these job descriptions.
The seniority label may be present in the set: [<all the labels given in the training data for reference>] . If not present in the set, then create a label.”
“Return a structured seniority label in the format {seniority_label: seniority label"}.”

3**. Work-Arrangements Structure**

“You are an expert job ad annotator. Your task is to classify a job ad into one of the work arrangements labels - (OnSite, Hybrid, Remote) from job descriptions.”
“
job_ad
”
“Extract work-arrangements information from the job ad.”
“Return a structured work-arrangements label in the format”
"{work_arrangements_label: work arrangements classification}. "
"Return only the JSON object with no extra details."

4.6 Improved Prompt Engineering (RBIC)

Role Based Incremental Coaching (RBIC) is a way to structure and improve prompt-engineering. It involves a chain of prompts where the model is first taught its role and then the model is made to solve the problem in incremental coaching. This incremental coaching enables the model to generate the prerequisite knowledge it needs to make the inference or extraction properly.
This method is proposed in the research paper, X., Carik, B., Gunturi, U. S., Reyna, V., & Rho, E. H. (2024). *Leveraging prompt-based large language models: Predicting pandemic health decisions and outcomes through social media language*. In *CHI ’24: Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems* (pp. …). ACM. https://doi.org/10.1145/3613904.3642117.

We also added high inference few shot examples in the role explanation phase of RBIC, which we found by manually going through training data.

Models tested with RBIC + few shot (Some of these models were tested using RBIC alone, without few shot examples):

1. Large - DeepSeek-V3, Gemini 1.5 Flash and GPT 3.5
2. Medium - Gemma3
3. Small- T5 and Qwen

Open Source - DeepSeek-V3, Gemma3, T5 and Qwen
Proprietary - Gemini 1.5 Flash, GPT 3.5

Research Question being addressed:

1. We are able to draw a solid comparison between a pre-trained model and a fine-tuned model.
2. By having models of different sizes we can also have a comparison of running speed for open-source models.
3. Also, by having a variety of open-source and proprietary models, we are able to address another research question on whether open-source models are better in comparison to proprietary.
4. Lastly, by comparing this normal prompting approach with this structured prompting we can draw comparison to other structured but slower methods to perform prompt engineering.

Prompt Template Framework (RBIC)

1. Role Specification
   – Purpose: Orient the model to its expert identity and overall task.
   – Structure:
      1. System: "You are an expert [domain] annotator. Your job is to …."
      2. User: "Based on your role, briefly explain what … means and what valid labels/output look like."
2. Pattern Elicitation (Sub-Task Instruction)
   – Purpose: Prime the model on key linguistic cues or patterns.
   – Structure:
      1. User: "As a [task] classifier/extractor, what are some common phrases or numeric patterns that indicate [the target information] in a job description?"
      2. Few-Shot Demonstrations
         – Purpose: Ground the model with real examples to reduce hallucination.
         – Structure: Repeated pairs of
      3. User: { "job_ad": "…", … }
      4. Assistant: "<gold-label for that example>"
3. Iterative Coaching
   – Purpose: Break down reasoning into smaller checks and clue extraction.
   – Steps:
      1. Presence Check:
      2. User: "Does this description include any [target] information? Just answer 'Yes' or 'No.'"
      3. Clue Extraction:
      4. User: "Extract the [target] phrase verbatim. Respond in JSON: {'Clue': ''}."

4. Structured Output Generation
   – Purpose: Synthesize the final, fully structured answer.
   – Structure:
5. User: “Based on the extracted clue, return JSON: { … } in this format:
   1. Salary: {“MinSalary”: “”, “MaxSalary”: “”, “Currency”: “”, “Frequency”: “”}
   2. WorkArr.: {“work_arrangements_label”: “”}
   3. Seniority: {“seniority_label”: “”}

4.7 Fine-tuned Language Models

Fine Tuning is a technique used to improve the performance of a pretrained model by updating its weights through further training on a smaller dataset designed for a specific task. Fine Tuning thus allows us to leverage both the broad foundational knowledge gained from pre-traininged models while also tailoring their performance for detailed and specific tasks with comparatively less data and resources. To accurately explore the techniques’s effectiveness, we performed fine tuning for:

- Deepseek R1 - Larger open source model
- Gemini 1.5 Flask - Larger proprietary model
- Gemma 3 - Smaller open source model
- Qwen 1.5B - Smaller open source model

Since Gemini was a proprietary model, it could only be trained through Google’s own API software development kit. While this method has the advantage of allowing us to upload data and tune the model without needing any local resources, it also introduces prominent limitations that will be discussed in later sections.

The open source models were fine-tuned using **SFTTrainer** from **trl** library [4]. The following models from **Hugging Face** were considered:

- google/gemma-3-1b-it [5]: Gemma3 (same used for prompting)
- Qwen/Qwen2.5-1.5B-Instruct [6]: Qwen2.5 (same used for prompting)
- deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B [7]: Qwen2.5, distilled from DeepSeek-R1

  The original DeepSeek is an open-source model, thus it can be fine-tuned. However both DeepSeek-V3 and DeepSeek-R1 contain 671B parameters, which makes their local hosting extremely resourceful. Unfortunately, we could not afford obtaining enough resources to fine-tune DeepSeek-V3, which we use for prompting-based experiments.

  For this reason we decided to fine-tune a distilled model. Unfortunately, there are no official distills for DeepSeek-V3 provided, and third-party distills exceed the capacity of resources available to us. Hence why we chose one of the official distills of DeepSeek-R1, specifically Qwen2.5 (1.5B parameters). We discuss why such comparison of DeepSeek’s performance is still fair in Appendix B.

For local fine-tuning, models are loaded with *AutoModelForCausalLM* from **transformers** [9] library. The following hyperparameters are used and retained throughout experiments with models for consistency of results:

- seed: 123
- epochs: 5 (“salary” and “seniority” datasets), 4 (“work arrangements” dataset)
- batch size: 1
- gradient accumulation: 8 steps

- lr (max): 0.0001
- lr scheduler: cosine, with warm-up (50 steps)
- weight decay: 0.01
- optimizer: AdamW (paged, 8bit)

The resulting checkpoints are hosted on Hugging Face. For fine-tuning loss and accuracy, refer to Appendix A.

Research questions answered through this approach:

1. By comparing prompted and fine-tuned language models, we investigate if fine-tuning would improve models' performance and eliminate widely appearing prediction mistakes.
2. We compare local fine-tuning of open-source models with fine-tuning a proprietary model via API, discussing limitations and strengths of both approaches.

Chapter 5

Chosen Models

	Model Name	Parameters	Context Window	Open Source/proprietary
Large	Gpt 3.5	175 Billion	~ 2048 (unofficial estimate)	Proprietary
Large	DeepSeek-V3	671 Billion	128K	Open Source
	fine-tuning: DeepSeek-R1 distill Qwen	1.5 Billion	128K	Open Source
Large	Gemini 1.5 Flash	~ 120 Billion (unofficial estimate)	1 Million	Proprietary
Medium	Gemma3	1 Billion	128K	Open Source
Medium	Qwen	500 Million	128K	Open Source
Small	T5	Small: 60 Million Base: 220 Million Large: 770 Million XL: 3 Billion	512	Open Source

While the approach for most models was standardised, there were some design decisions and scope limitations that impacted our approach with T5, Gemini 1.5 Flash and GPT 3.5. These will be discussed below

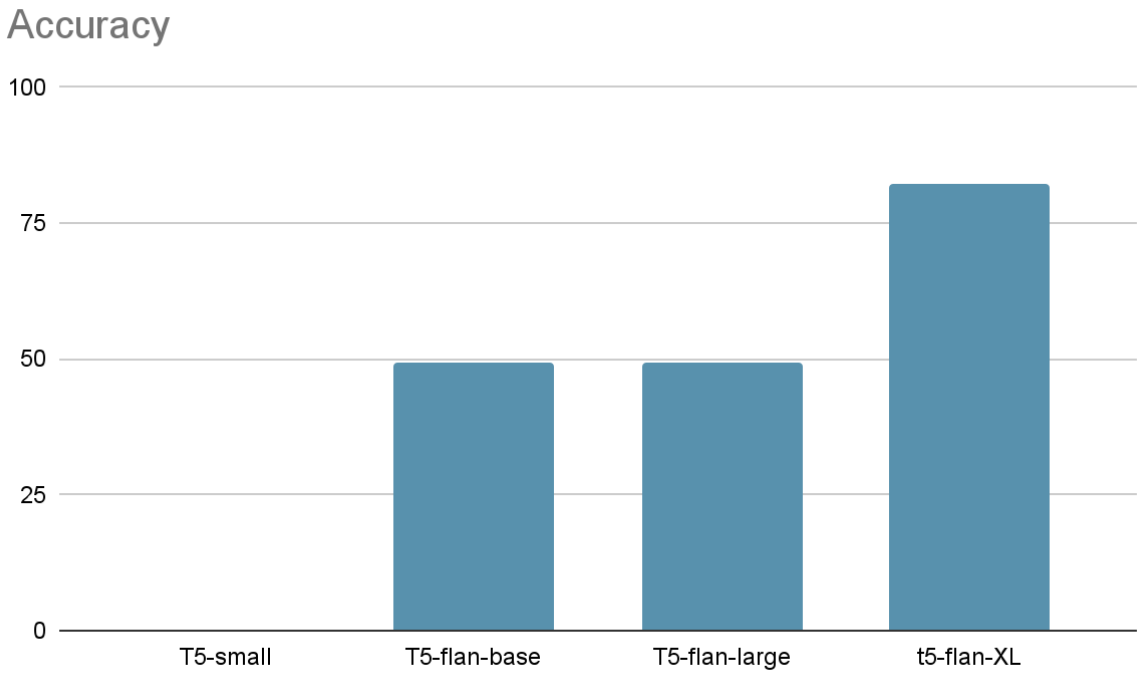
5.1 T5 models from Google

T5 Models tried:

- T5 -Small
- T5- Base
- T5- Large
- T5- XL

There is a T5 XXL variant but it can't run on Colab. So, due to resource limitation this model was not tried.

Results of T5 on work arrangement data set. (Development set = 99 entries, Test set = 99 entries)



Model	Development Set	Test Set	Average
T5-Small	0.00	0.00	0.00
T5-Base (flan)	53.54	45.44	49.48
T5-Large (flan)	53.54	45.44	49.48
T5-XL (flan)	79.80	84.85	82.31

Note: T5-small non-flan version was used because it was used for fine tuning.

5.1.1 Prompting Techniques

The T5 model operates effectively when given clear instruction-based prompts. Throughout the experimentation, both Instruction Prompting and RBIC (Role-Based Incremental Coaching) were evaluated.

Key Observations:

- Instructional prompts alone proved sufficient in guiding the model toward accurate work arrangement extraction.

- RBIC-style prompts (e.g., “You are an expert annotator...”) did not show any measurable improvement in accuracy or quality of the model output.

- In some cases, RBIC language led to increased confusion, likely due to dilution of task focus within the prompt.

5.1.2 Conclusion

Three smaller variants of T5 (base, and large) have similar performance. By reducing the input to essential, concise, and direct instruction-based prompts, the T5-XL model was able to process job advertisements and yield better results than smaller models. The RBIC format was deemed unnecessary and potentially detrimental for this specific task.

5.2 Salary Extraction Using T5: Issues and Solutions

Problem Overview

During the task of extracting salary information from job advertisements using the T5 model, a key challenge encountered was input length limitation. The T5-XL used in this experiment, has a token limit of 512, beyond which inputs are truncated or cause errors. Since many job advertisements contain lengthy descriptions with considerable non-relevant content, this led to excessive token usage and impacted model performance.

Preprocessing Strategy

To mitigate this issue, a targeted preprocessing approach was implemented:

- The job advertisement text was filtered to retain only lines containing numeric values and the line immediately preceding each numeric line.
- This preprocessing effectively preserved contextually relevant salary information (e.g., headers like *"Compensation Range"*) while discarding extraneous content.
- As a result, this approach significantly reduced token count per input without sacrificing critical details required for salary extraction.

T5-small variant was used for testing later on it gave results such as (`>We’re currently looking for a high caliber professional to join our team as b>Vice President, Business Treasury Analyst (Balance Sheet Mgmt Lead Analyst) based in Kuala Lumpur, Malaysia./p>p>/p>p>Citi Finance is responsible for the firm’s financial management and related controls. ---- 0-0-None-None`) which implied that the model was unable to process the request. It gives an output towards the end(`0-0-None-None`) which suggests fine tuning could be useful.

5.3 T5-XL Performance on Salary Extraction Task

Prompt Design and Model Behavior

Multiple prompt variations were tested with the T5-XL model to guide it in extracting salary information from job advertisements in the structured format: min-max-currency-frequency (e.g., 30000-50000-PHP-MONTHLY). Despite the clarity of the instruction prompts, T5-XL performed poorly on this task.

Key Observations

- In many cases, the model failed to produce the required structured format, even when the prompt explicitly instructed it to do so.
- The most frequent output was 0-0-None-None, which technically matched the expected format but did not indicate meaningful understanding or correct extraction. This response often occurred regardless of whether salary information was present in the job ad.
- Although there were occasional correct predictions in the required format, such instances were rare and inconsistent.

Challenges

The underperformance of the model appears to stem from two main factors:

1. Large Input Size: The unprocessed job ads often exceeded the token limit or contained excessive irrelevant information, making it difficult for the model to focus on the salary-related content.
2. Task Complexity: Extracting structured numerical data (especially when expressed in diverse natural language formats) remains a non-trivial task for instruction-tuned models without further fine-tuning.

Conclusion

Out-of-the-box usage of T5-XL, even with well-crafted instruction prompts, was insufficient for reliable salary extraction. The model's tendency to default to 0-0-None-None suggests that it struggled to parse and process the relevant information effectively. Fine-tuning the model on domain-specific examples or incorporating additional preprocessing steps may be necessary to improve performance on this task. T5 small gave 0 results when used on the three tasks.

5.4 Fine tuning:

5.4.1 Fine-Tuning T5-Small on Work Arrangement Classification Task

Fine-tuning experiments were conducted on the T5-small model to investigate the potential of small-scale models (i.e., models with fewer than 1 billion parameters) in domain-specific classification tasks.

Due to resource limitations, T5-XL was not selected for fine-tuning; however, comparable models such as Qwen and Gemma were fine-tuned, and their respective results are discussed in subsequent sections. Initial evaluations suggest that the performance of Qwen and Gemma is comparable to that of T5-XL based on previously observed accuracy metrics.

The T5-small model was fine-tuned on a custom work arrangements dataset and achieved an overall accuracy of 42%.This result is noteworthy, considering that prior to fine-tuning, the model demonstrated considerable difficulty in generating meaningful predictions.The findings of this study

clearly highlight the value of fine-tuning smaller language models, even in environments constrained by computational resources. Fine-tuning not only improved the model's predictive accuracy substantially but also confirmed that small models can be effectively adapted to specialized tasks with relatively modest effort.

5.4.2 Gemini 1.5 Flash fine-tuning limitations

Since Gemini 1.5 flash is a proprietary model, it can only be finetuned using Gemini's own API based python software development kit (SDK). However the approach also raises its own issues. Firstly, Google limits the use of text prompting and generation features such as system prompts and JSON response schema to base non-fine tuned models. Such limitations greatly hinder the functionality of tuned models and highlights a disadvantage of relying on proprietary APIs for fine tuning. Such API limitations have caused us to use a simpler finetuning approach for Gemini 1.5 flash as compared to the locally hosted open source alternatives.

Since Gemini can only be prompted with one text input, it has only one input field (called text_input) and an output field (called output). Thus to accommodate this, every input feature except "job_id" from the cleaned data sets was concatenated together. This text was then augmented with a prompt to form the final text input. This same preprocessing step was done on the test set for evaluation. Lastly, the default and recommended hyperparameters of an epoch count of 5, batch size of 4, learning rate of 0.001 were used for fine-tuning. The tuned model was then used to generate labels from the test set, which were generated in a JSON format that the model was largely able to follow even without a response schema. The test labels were then evaluated accordingly.

5.4.3 GPT 3.5 resource limitations

Since GPT 3.5 was being accessed through a paid API our group did not have the resources to extensively test and fine-tune the model. However we still chose to explore it's performance for prompting with RBIC for the salary dataset (extraction task) and seniority (classification task) to both better model the effectiveness of RBIC and gain deeper insights into the functionality of out of the box proprietary models without.

Chapter 6

Evaluation Metrics

6.1 Salary

Exact match with the correct answers given in the test dataset. Since, we are extracting the following details separately, we perform an exact match for each of these details to analyse individual extraction performance and an overall match of the label.

1. Min. Salary
2. Max. Salary
3. Currency
4. Frequency
5. Overall (overall match will happen only all the previous details)

6.2 Seniority

Seniority does not have a fixed set of labels, and there are 8 labels present in the test set which do not

Test Label	**Equivalent Training Label**
junior-to-intermediate	junior-intermediate
associate-director	associate director
third-year apprentice	None (1st, 2nd and 4th year apprentice are present)
early-career	None (entry-level is a close label)
cadet	None
senior/principal	senior & principal separately present
intermediate management	None (middle-management , middle management are some close labels)
retired	None

have an exact match (although similar values are present for some) in the training set.

Since a fixed set of labels can be provided to perform an F1 test. Also, we can expect the model to give labels that exactly match the correct labels given in the test set. And hence, closely related words can be considered as the correct labels.

Evaluation Strategy

We classify a match into 2 categories,

1. Exact Matches: When the predicted label (given by the model) exactly matches the true label (given in the test set).
2. Similar Labels: We classify the non exact-matches into a similar label category when the predicted label and true label share the same level of seniority.

Level-of-Seniority	Labels in this Level
Entry	“entry-level”, “graduate”, “cadet”, “apprentice” etc.
Early Career	“assistant”, “junior”, “student”, etc.
Mid Career	“intermediate”, “experienced”, “mid-level”, etc.
Senior	“senior”, “lead”, “principal”, “specialist”, “sous”, etc.
Management / Exec	“manager”, “director”, “executive”, “chief”, “head”, etc.

We consider, both these match categories to be a correct match.

Considerations

We went through the obtained labels and test dataset to create these categories (with the specific set of labels) for the specific Language Model we used.

Different language models may give different labels and same language models may give different labels when asked to infer the label multiple times.

Other Evaluation Strategies Explored

1. BERT Test

We created embeddings using BERT and JOBBERT for both predicted labels and true labels and consider a high cosine similarity to be a correct match. However, we observed some mismatches that may occur with this approach, that can be justified with this example-

Based on general understanding of job seniority, the labels of “senior” and “experienced” have a similar meaning in comparison to the meanings of the labels “senior” and “junior”.

But BERT embeddings of “senior” and “experienced” and different versions of the label like “senior level job posting” and “experienced level job posting” etc, had a lower cosine similarity than the embeddings of “senior” and “junior” and different versions like “senior level job posting” and “junior level job posting”.

Around, 0.8581167459487915 and 0.9479404091835022 respectively.

2. F1-Score was also not feasible because there is no fixed set of labels.

6.3 Work-Arrangements

Since this is a classification task with only 3 labels - (OnSite, Hybrid, Remote) we can use an F1 score as an evaluation metric

F_1 score
The F_1 score is the harmonic mean of precision (the fraction of predicted positives that are actually positive) and recall (the fraction of true positives that are correctly identified). It's defined as

F_1 = 2 x (Precision x Recall) / (Precision + Recall)

By combining precision and recall, the F_1 score balances the trade-off between false positives and false negatives, making it especially informative for imbalanced-class or cost-sensitive classification tasks.

- Precision: Measures the accuracy of positive predictions, i.e. the proportion of instances labeled positive by the model that are truly positive.
- Recall: Measures the model's ability to find all positive instances, i.e. the proportion of true positive instances that the model correctly identifies.

Chapter 7

Results

7.1 Salary

Model Name	**Type**	**Accuracy (%)**				
		Overall	Min. Salary	Max. Salary	Currency	Frequency
GPT 3.5	RBIC + fewshot	85.01	90.12	89.77	96.30	95.77
DeepSeek	Regular prompting	87.30	91.36	90.48	97.18	96.65
	RBIC	87.13	91.01	90.65	97.00	96.65
	RBIC + fewshot	88.36	91.36	90.83	97.53	97.71
	Fine-tuning	**94.53**	**95.94**	**96.65**	**97.88**	**97.88**
Gemini 1.5 Flash	Regular prompting	83.42	88.18	87.83	96.83	96.65
	RBIC + fewshot	82.19	88.36	88.54	96.65	92.42
	Fine-tuning	**91.89**	**94.88**	**94.53**	**98.41**	**97.71**
Gemma3	Regular prompting	8.99	51.85	52.91	35.80	47.44
	RBIC	4.76	81.48	72.49	21.69	58.38
	RBIC + fewshot	3.17	65.78	56.79	8.47	32.80
	Fine-tuning	**93.30**	**95.77**	**94.71**	**98.41**	**98.24**
T5 (small)	Regular prompting	0.00	0.00	0.00	0.00	0.00
Qwen	Regular prompting	1.94	27.87	19.40	19.22	19.40
	RBIC	0.00	7.58	6.88	0.71	22.93
	RBIC + fewshot	0.00	3.00	1.23	1.06	17.81

	Fine-tuning	**94.00**	**95.59**	**95.94**	**97.88**	**98.06**

Note: other variants to T5 would have performed at poor levels based on empirical tests done on T5 XL.

7.2 Seniority

Model Name	**Type**	**Accuracy (%)**		
		Exact Matches	**Similar Labels**	**Overall (Exact + Similar)**
GPT 3.5	RBIC + fewshot	47.75	15.23	62.98
DeepSeek	Regular prompting	55.30	11.76	67.05
	RBIC	52.25	**14.37**	66.62
	RBIC + fewshot	49.20	11.03	60.23
	Fine-tuning	**65.75**	11.61	**77.36**
Gemini 1.5 Flash	Regular prompting	**58.60**	18.28	**76.92**
	RBIC + fewshot	52.39	**19.59**	71.98
	Fine-tuning	52.54	14.95	67.49
Gemma3	Regular prompting	22.93	8.71	31.64
	RBIC	17.71	**16.55**	34.25
	RBIC + fewshot	20.61	11.03	31.64
	Fine-tuning	**65.17**	11.61	**76.78**
T5 (small)	Regular prompting	0.00	0.00	0.00
Qwen	Regular prompting	16.00	0.00	16.00
	RBIC	11.00	1.00	12.00

	RBIC + fewshot	10.05	0.00	10.05
	Fine-tuning	**65.60**	**11.03**	**76.63**

7.3 Work Arrangements

Per-class metrics:

Model Name	**Type**	***Remote***			***Hybrid***			***OnSite***		
		Pr.	**Rec.**	**F1**	**Pr.**	**Rec.**	**F1**	**Pr.**	**Rec.**	**F1**
DeepSeek	Regular prompting	92.9	**100**	96.3	96.0	88.9	92.3	**97.8**	**97.8**	**97.8**
	RBIC	**100**	96.2	**98.0**	76.5	**96.3**	85.2	97.5	84.8	90.7
	RBIC + fewshot	92.9	**100**	96.3	82.8	88.9	85.7	97.6	89.1	93.2
	Fine-tuning	92.0	88.5	90.2	**96.2**	92.6	**94.3**	91.7	95.7	93.6
Gemini 1.5 Flash	Regular prompting	92.6	96.2	94.3	85.7	88.9	87.3	95.5	91.3	93.3
	RBIC + fewshot	**92.9**	**100**	**96.3**	85.7	88.9	87.3	**97.6**	87.0	92.0
	Fine-tuning	92.8	100	96.2	**100**	**88.9**	**94.1**	95.7	**97.8**	**96.8**
Gemma3	Regular prompting	65.7	88.5	75.4	39.3	81.5	53.0	80.0	8.7	15.7
	RBIC	30.4	92.3	45.7	35.7	18.5	24.4	**100**	2.2	4.3
	RBIC + fewshot	28.0	**100**	43.7	20.0	3.7	6.2	0.0	0.0	0.0
	Fine-tuning	**77.4**	92.3	**84.2**	**58.6**	**63.0**	**60.7**	84.6	**71.7**	**77.6**
T5 (small)	Regular prompting	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Fine-tuning	**22.0**	**35.0**	**27.0**	**14.0**	**4.0**	**6.0**	**63.0**	**70.0**	**66.0**
Qwen	Regular prompting	85.7	23.1	36.4	39.5	55.6	46.2	57.7	65.2	61.2
	RBIC	25.0	3.8	6.9	30.0	11.1	16.2	42.9	32.6	37.0

	RBIC + fewshot	0.00	0.00	0.00	50.0	7.4	12.9	48.8	87.0	62.5
	Fine-tuning	**88.5**	**88.5**	**88.5**	**85.2**	**85.2**	**85.2**	**93.5**	**93.5**	**93.5**

Note: Google T5-XL results were comparable to Qwen

7.4 Average metrics

Model Name	**Type**	**Acc.**	**Precision**		**Recall**		**F1-score**	
			macro	wtd.	macro	wtd.	macro	wtd.
DeepSeek	Regular prompting	**96.0**	**95.6**	**96.0**	**95.6**	**96.0**	**95.5**	**95.9**
	RBIC	90.9	91.3	92.4	92.4	90.9	91.3	91.1
	RBIC + fewshot	91.9	91.1	92.3	92.7	91.9	91.7	92.0
	Fine-tuning	92.9	93.3	93.0	92.2	92.9	92.7	92.9
Gemini 1.5 Flash	Regular prompting	91.9	91.3	92.0	92.1	91.9	91.6	91.9
	RBIC + fewshot	90.9	92.0	93.1	91.9	90.9	91.8	91.8
	Fine-tuning	**96.0**	**96.2**	**96.2**	**95.6**	**96.0**	**95.7**	**95.9**
Gemma3	Regular prompting	49.5	37.0	65.1	35.7	49.5	28.8	41.6
	RBIC	30.3	20.8	64.2	14.1	30.3	9.3	20.6
	RBIC + fewshot	27.3	12.0	12.8	25.9	27.3	12.5	13.2
	Fine-tuning	**74.7**	**73.6**	**75.6**	**75.7**	**74.7**	**74.2**	**74.8**
T5	Regular prompting	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Qwen	Regular prompting	51.5	61.0	60.1	47.9	51.5	47.9	50.6
	RBIC	19.2	32.6	34.7	15.9	19.2	20.0	23.4

	RBIC + fewshot	42.4	32.9	36.3	31.5	42.4	25.1	32.6
	Fine-tuning	**89.9**	**89.0**	**89.9**	**89.0**	**89.9**	**89.0**	**89.9**

Chapter 8

Discussion

8.1 Salary

As seen from the results for prompting, and RBIC is largely ineffective for smaller language models such as T5 as they are largely optimized for concise and focused prompts that clearly specify the task. Incorporating additional background information through RBIC introduced unnecessary complexity, which diluted the primary instruction signal. As a result, the model often became confused or ignored the intended task. Therefore, simple, direct prompting proved to be significantly more effective for performance.

Similarity, for medium sized models, we observed that prompt-based LLM frameworks—standard prompting, RBIC, and RBIC with few-shot examples—tend to perform poorly on extraction tasks, such as identifying salary information. For example, both Qwen and Gemma frequently misclassify salary labels, with Qwen often incorrectly extracting the label as "--USD-MONTHLY", with no mention of salary. In these cases, fine-tuning was essential to enhance extraction accuracy, pushing overall performance above 90%. Notably, Qwen's performance improved from nearly 0% to 94.5% accuracy after fine-tuning—the most significant gain observed across all models and tasks.

For larger language models however, RBIC shows mixed results in performance. For DeepSeek, traditional RBIC leads to nearly identical performance across the board with the few shot examples improving performance slightly. For Gemini 1.5 Flash RBIC with few shot actually causes a decrease in the model's ability to identify frequency which could be caused by the model hallucinating from the few-shot examples. Just as observed for the smaller models, fine-tuning looks to be the best way to improve model performance, with a roughly 7-8% increase in total accuracy for both DeepSeek and Gemini. While this isn't as extreme as the improvements made with the smaller models, it is still a substantial result.

Observations on the salary dataset:

Initial results for performing prompting and RBIC on large models with the salary dataset revealed several edge cases and problematic data entries within the dataset. While a small number of misses come from rounding preference differences on salary ranges with decimals, where some entries are the ceiling while others are the floor. There were notable cases in the test set such as:

1. The salary range being mentioned without any frequency (example, ID: 65820039)
2. The y_true label includes an answer despite there being no salary range information in the input data (example, ID: 67277027). A majority of these cases came from New Zealand
3. Multiple different salary ranges presented for different job roles that are all included in the same description (example, ID: 70924067)
4. The most prominent edge case however is when both a "compensation" number and a "compensation range" are provided, but the final answer has min and max salary both equal to the number instead of the range. (example, ID: 74289362), a majority of these cases came from the Philippines

With most variations of prompting, the models failed for these cases. With case 1, although the model was capable of extracting the rest of the information, it was unable to determine the frequency based on the salary range. For case 2 the lack of any information made the model rightfully predict 0-0-None-None as the solution. For case 3 the model chose an arbitrary range and for case 4 it always chose the range over the number. These edge cases were the primary cause for the model's errors when prompting and thus do not reflect the model's true performance for the salary dataset, which would be higher given cleaner data. After fine-tuning while the models still failed for cases 2 and 3, they were able to better infer the frequency for case 1 and were also able to adapt to the edge case from case 4. This highlights a potential downside of fine-tuning, which is that it might adapt to biases or edge cases within the dataset, given that their distribution is large enough.

8.2 Seniority

Seniority detection from job advertisements proved too complex for small models like T5-small. The limited capacity of these models made it difficult to capture the nuanced language and contextual cues needed for accurate classification. As a result, their performance on this task was poor.

For medium sized models, classification tasks like determining seniority or work arrangement types, performed substantially better than they did on salary extraction yet performance was still poor. This suggests that medium-sized models are more effective on classification tasks. For standard prompting Gemma accuracy was 31.64% whilst Qwen accuracy was 19.40% suggesting poor performance, fine-tuning drastically increased performance to 76.78% and 98.06% respectively. Fine-tuning plays a pivotal role in significantly boosting the performance of medium sized models like Qwen and Gemma, particularly in scenarios where traditional prompting methods, such as RBIC and few-shot approaches, fall short.

For large models, fine-tuning played a less crucial role. While it increased DeepSeek's overall accuracy from 67.5% (regular prompting) to 77.36%, it didn't benefit Gemini at all (overall accuracy dropped from 76.92% to 67.49%). It is important to note, that among all prompting techniques (regular prompting, RBIC, RBIC with few-shot) both DeepSeek and Gemini got overly confused by RBIC and performed best with regular prompting. We can conclude that for this task LLMs work well enough with simple prompting, and while fine-tuning can be considered as well, one may expect no presence of significant quality improvement.

8.3 Work Arrangements:

We observed with medium sized models, regular prompting performed significantly better than RBIC and RBIC with few-shot, with greater evidence hallucinating the model. For example Qwen outputted irrelevant labels such as {"'Seniority'", "Seniority Classification"} for RBIC and RBIC with few-shot, and Gemma gained a significant bias towards "Remote" class with absence of "OnSite" predictions when evaluated with RBIC with few-shot. However regular prompting still produced moderate results with scores around 50% for both models. To improve results, fine-tuning was paramount in improving accuracy from 51.5% to 89.9% in Qwen and 49.95% to 74.7% in Gemma.
Large models perform extremely well on this dataset with regular prompting, with both accuracy and average F1-score (macro and weighted) of approximately 96% for both DeepSeek and Gemini. Besides, fine-tuning improved Gemini's metrics up to approximately 96%, while decreasing the quality of DeepSeek. Similar to the "seniority" dataset, for both large models we noticed a decrease in performance with RBIC-based prompting techniques in comparison to regular prompting. This

implies that for simple tasks (such as “seniority” and “work arrangements” in comparison to “salary”) RBIC doesn’t benefit LLMs and tends to cause hallucinations.

T5-Base Model: Sensitivity to Instructional Prompting

The experiment demonstrated that even minor punctuation changes can materially impact the performance of large instruction-tuned models like T5-Flan-base.

During experiments with the T5-base model for the work arrangement classification task, an important observation was made regarding the impact of prompt wording on model performance.

Initially, the prompts used were structured as follows:

```
Classify the work arrangement of the following job ad as one
of the following: on-site, Remote, or Hybrid {job_ad}
```

However, when the prompts were modified slightly with a :

```
Classify the work arrangement of the following job ad as
one of the following: on-site, Remote, or Hybrid: {job_ad}
```

there was a notable improvement in model accuracy.

Results

Dataset	Accuracy (Original Prompt)	Accuracy (Modified Prompt)
Development Set	53.53%	57.57%
Test Set	45.45%	54.54%

Accuracy went from 49.48% to 56.06%. This accidental discovery highlights that T5 models are highly sensitive to instructional prompting. Providing clear, structured instructions within the prompt significantly improves the model’s understanding and classification performance, even without any additional fine-tuning.This emphasizes the importance of prompt engineering when working with encoder-decoder models like T5.

Chapter 9

Conclusion

9.1 Open Weight vs Proprietary Models

The decision between usage of openweight or proprietary models ultimately depends on SEEK's priorities regarding control, customisation, initial setup, and data privacy. Open-weight models offer a high degree of customisation, allowing businesses to tailor models to specific tasks and use cases. Fine-tuning further enhances performance by incorporating domain-specific knowledge, which we observed when working with models such as Qwen, DeepSeek, and Gemma (see Chapter 5 for a detailed discussion of results). However, the initial setup of open-weight models can be complex and typically requires a dedicated engineering team to manage ongoing maintenance.

In contrast, proprietary models are easier to set up and can be maintained with a smaller engineering team. They are well-suited for general-purpose applications where minimal customisation is required. However, these models can incur significant long-term costs and may pose concerns around data privacy—especially in use cases involving sensitive user data. In such cases, we do not recommend the use of proprietary models.

For short-term, simple business applications where quick setup is a priority, proprietary models are a practical choice. For larger, long-term applications requiring flexibility and control, we recommend open-weight models, as they offer more independence from specific providers and better alignment with compliance and customisation needs.

9.2 Cost vs Speed vs Accuracy Tradeoff

Model	Cost (per M Tokens)	Speed (Token/Sec)	Accuracy
Deepseek (V3 - R1)	Input $0.27 - $0.55 Output $1.10 - $2.19	60	**SL: 94.53 (F)**
			SE: 77.36 (F)
			WK: 92.7 (F)
Gemini 1.5	Input $0.075 Output $0.30	300	SL: 91.89 (F)
			SE: 76.92 (P)
			WK: 95.73 (F)
Gemma 3	Locally Hosted	2585	SL: 93.3 (F)
			SE: 76.78 (F)
			WK: 74.2 (F)
Qwen	Locally Hosted	183	SL: 94.00 (F)
			SE: 76.63 (F)
			WK: 89.0 (F)

When selecting the right LLM for a task, we face the crucial balancing act of cost, speed, and accuracy. This table visually highlights these trade-offs across the prominent models. Despite the variation in the speeds of these models, for tasks with low output token requirements, like the one

we're considering, the speed difference becomes less critical, regardless the much lower output token speed of the DeepSeek model can be a disadvantage if the model is to be used at scale or for handling multiple complex prompts concurrently. And while Gemma 3 provides the greatest advantage here, both gemini and gwen also provide sufficient speeds for this application.

In terms of cost, Gemini is the clear winner with the lowest per million input and output token price, and is significantly cheaper than DeepSeek. And while at first glance the locally hosted models provide clear cost advantages, the true cost of these models is dependent on whether there is already sufficient infrastructure to deploy them at scale, in which case they are essentially free, but building such an infrastructure from scratch could be quite expensive. However, depending on the scale of the project such an investment could be cheaper than relying on APIs over the long run.

In terms of performance, for extraction tasks like salary the performance of smaller models after finetuning is on par with or sometimes even exceeds the larger models. However even after fine tuning the small models struggle with text classification and inference tasks like those required for the seniority and the work arrangements dataset. In contrast the larger LLMs provide great performance across the board, where even without fine tuning their performance only drops by a few percent.

After considering these observations the optimal model choice comes down to a few factors. If the solution is to be used at scale, but the company does not have the resources to or isn't interested in deploying the models themselves, then Gemini 1.5 flash proves to be a great model that perfectly balances cost, speed and performance. However, If the company does have extensive resources or is able to efficiently outsource them, then the optimal cost, accuracy and speed tradeoffs would be to use a small model like Gemma 3 or Qwen for extraction tasks like salary, and use a larger model like Gemini 1.5 Flash for inference heavy tasks like seniority and work arrangements.

9.3 Is Fine-Tuning Worth It?

Referencing the results from the previous table, we see that the best results from nearly every model is achieved through fine-tuning (indicated by the 'F'), clearly highlighting the impact of fine tuning on performance. However there are some nuances to explore here. For smaller models, the performance jump from prompting to finetuning is large, where without fine tuning they perform very poorly for all tasks and with fine tuning they perform closer to the larger models. Thus if these models are to be used then fine tuning is mandatory. For larger models, while there is still a noticeable performance increase, the difference between prompting and finetuning is much smaller, making it potentially viable for them to be used out of the box with some prompt engineering. Another point to remember is that fine tuning also incurs additional costs, and while this is negligible for smaller datasets such as the ones presented for this project, expanding this to larger data can incur a significant cost. Moreover, hiring personnel to create data and perform the tuning process will also contribute to additional overhead. Thus although finetuning will guarantee the best possible results, if the solution is firmly centred on state of the art larger language models, there is some merit in relying exclusively on prompt engineering.

9.4 Is Our Solution Unbiased, Fair and Transparent?

We believe that our solution is largely unbiased, fair and transparent. Firstly, design choices such as not translating non-english text within the salary dataset ensures that we do not introduce any

unintentional biases that come with translation. This is made possible by our use of newer multilingual LLMs which also boosts the inclusivity and accessibility of the solution. Safety settings within models like gemini can also allow us to prevent the model from generating explicit, harmful or biased information. Moreover, the open source alternatives presented in our approach offer complete control over the process, where the lack of third party processing and data handling improves transparency. This however might be lost to an extent when dealing with proprietary APIs.

Bibliography

[1] T5. Available at: https://huggingface.co/docs/transformers/en/model_doc/t5 (Accessed: 26 April 2025).

[2] Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., ... & Qiu, Z. (2024). Qwen2. 5 technical report. arXiv preprint arXiv:2412.15115.

[3] Gemma explained: An overview of Gemma Model Family Architectures (no date) Google Developers Blog. Available at: https://developers.googleblog.com/en/gemma-explained-overview-gemma-model-family-architectures/ (Accessed: 20 April 2025).

[4] Von Werra, L.,Belkada, Y., Tunstall, L., Beeching, E., Thrush, T, Lambert, N., Huang, S., Rasul, K, & Gallouédec, Q.. (2020). TRL: Transformer Reinforcement Learning.

[5] google/gemma-3-1b-it. Available at: https://huggingface.co/google/gemma-3-1b-it (Accessed: 28 April 2025).

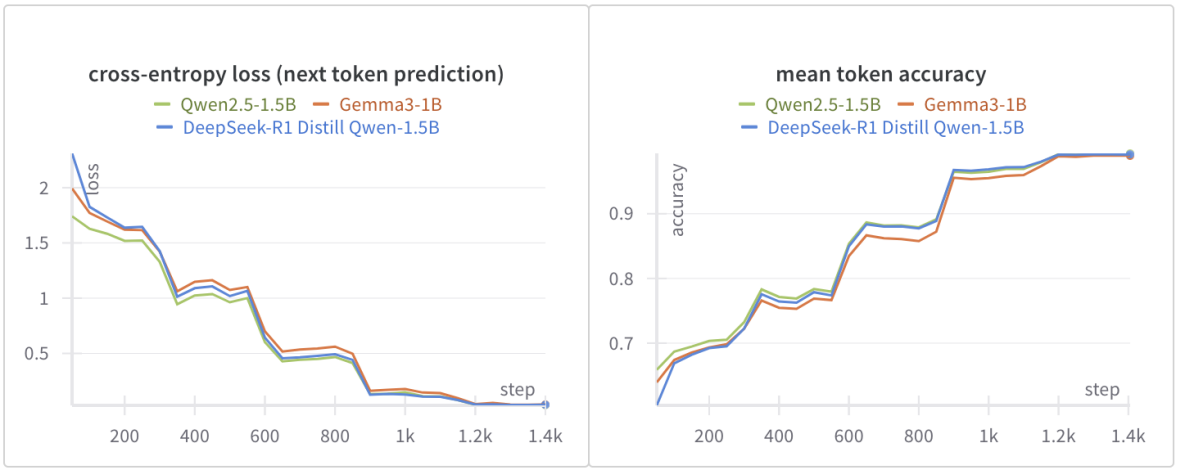
[6] Qwen/Qwen2.5-1.5B-Instruct. Available at: https://huggingface.co/Qwen/Qwen2.5-1.5B-Instruct (Accessed: 28 April 2025).

[7] deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B. Available at: https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B (Accessed: 28 April 2025).

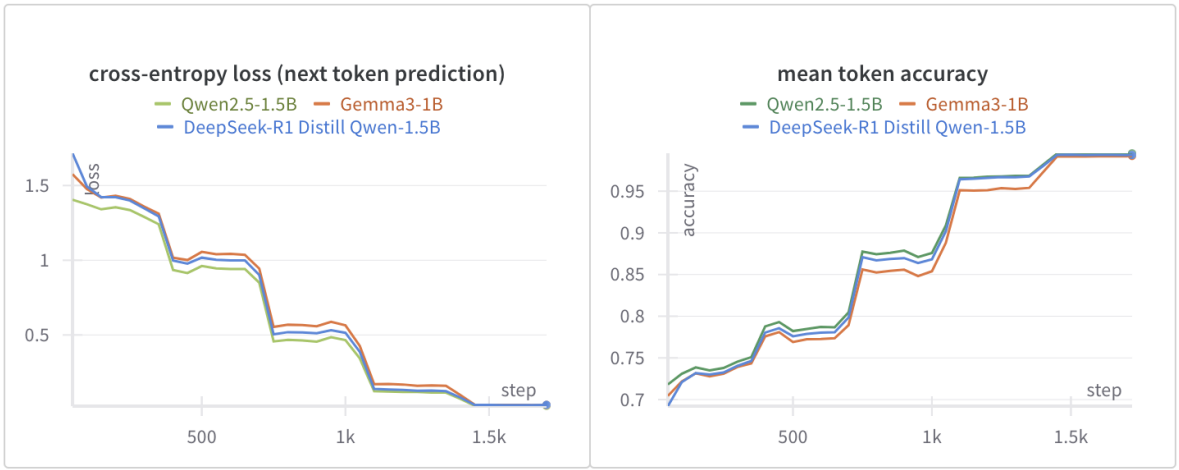
[8] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, A., Von Platen, P., Ma, C., Jernite. Y., Plu, J., Xu, C., Le, T. Scao, Gugger. S, Drame, M., Lhoest, Q., & Rush, A. M. (2020). Transformers: State-of-the-Art Natural Language Processing. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations (pp. 38–45). Association for Computational Linguistics.

Appendix

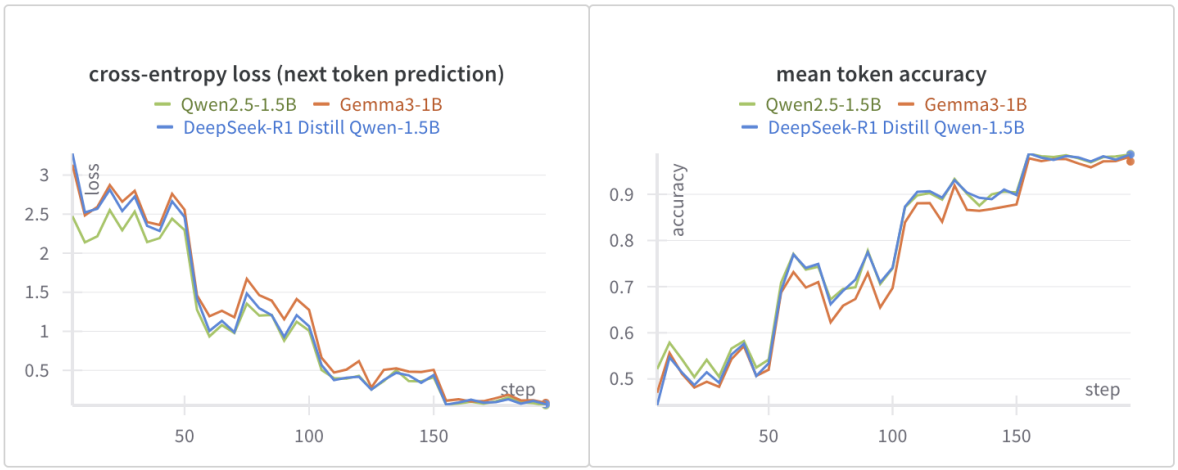
A. Local Fine-tuning Details



(a): “Salary” dataset



(b): “Seniority” dataset



(c): “Work arrangements” dataset

Figure A.1: Fine-tuning loss and token accuracy.

B. DeepSeek-V3 and DeepSeek-R1 Comparison

As discussed in Chapter 4 [*Neural Approach: RBIC and Fine-tuning - Fine-tuned Language Models*], limited resources forced us to consider fine-tuning distilled DeepSeek-R1 and provide its results alongside API-accessed DeepSeek-V3. Here we would like to acknowledge that the final comparison is not entirely fair. However, we claim that it can be trusted.

Table B.1: Quantitative comparison of DeepSeek-V3 and DeepSeek-R1 on the "salary" dataset. Both models were accessed via API. The total time was measured in respect to processing the whole test set.

Model Name	**Accuracy (%)**					**Total Time (seconds)**
	Overall	Min. Salary	Max. Salary	Currency	Frequency	
DeepSeek-V3	87.30	91.36	90.48	97.18	96.65	2917
DeepSeek-R1	86.77	91.01	91.18	98.59	94.36	11425

We compared DeepSeek-V3 and DeepSeek-R1 performance on the "salary" dataset with regular prompting. As shown in Table B.1, both models performed on a similar level, as the task is not complicated enough to require reasoning capabilities from the model. Besides, DeepSeek-V3 was significantly faster (hence why it was chosen for prompting experiments).

Based on this comparison, we assume that DeepSeek-R1 does not considerably outperform DeepSeek-V3, hence why fine-tuning DeepSeek-R1 is a reasonable choice. Besides, we can also assume that the performance of fine-tuned DeepSeek-R1 would not be inferior to the one of its significantly smaller distil.

Marking criteria reference

Part a **Problem Definition**(10 marks)

NLP Problem - Specify the research question

Text Source/Domain

Part b **Dataset Selection** (20 marks)

Use two existing datasets (10 marks)

- Publicly available datasets. In the case of industry project, it will be the datasets provided.

Create your own labelled dataset (20 marks)

- Correct selection of labels, inter annotator agreement

Use an existing lexicon (10 marks)

- 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.

Use an existing pretrained/fine tuned model (5 marks)

- 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.

Fine-tune a model based on a dataset (20 marks)

- Examples: finetuned BERT, prefixtuned LLaMA

Extend a method (30 credits per extended finetuning method)

- 30 credits per extended finetuning method
- Examples of extension: modification of the loss func+on, incorporation of structured ontology, prompting method other than zero/few-shot prompting, etc.

Integrate a language model with external tools

- Usage of a library like LangChain

PART D: Evaluation Minimum: 20 credits

Quantitative Evaluation 10 credits

- Appropriate metrics.

Qualitative Evaluation 5 credits

- Examine misclassified instances and produce common error types

Command line testing 5 credits

- Interface to test out the system. This can be executed using an input argument or input file.

Demo 10 credits

- A simple demo through Gradio (or equivalent).

Report

Part A

Problem Definition

Given the influx of job seekers and employers in the market, this has propelled an increase in job advertisements on various employment websites. Thus, the process of identifying relevant information for prospective job seekers, has become a pertinent task.

Part B

Part c

The chosen models investigated in this research question are as follows:

	Model Name	Parameters	Context Window	Open Source/proprietary
Large	Gpt 3.5			

Observations from meeting:

- Few shot doesn't seem advantageous
    - Since specify output structure

Salary Fail cases

- Singapore --> USD
- Compensation range and answer is compensation value
- Single job listing : multiple salary range and final salary range is mentioned in additional text
    - Salary_additional_text could be passed first

Qwen

Salary

RBIC with Fewshot	RBIC no fewshot	Prompting
The models desired output when prompted was the deliver a JSON element with keys: MinSalary, MaxSalary, Currency and Frequency. For same cases this was - Additional trailing curly braces “}}” - Missing quotes within the value for MinSalary {"MinSalary": □100, "MaxSalary": □250, "Currency": "THB", "Frequency": "monthly"}		