From Text to Talent: Efficiently Extracting Requirements from Job Postings by Transferring Knowledge from a Larger LLM to a Fit for Purpose Encoder-only Model

AdeptID – Team #7

Rafay Basheer
Georgia Institute of Technology

Robert Carlton
Georgia Institute of Technology

Keith Kral
Georgia Institute of Technology

***Abstract* – Extracting nuanced, interdependent job requirements in a structured manner from unstructured job posting text requires methods beyond traditional text parsing systems. Focused on extracting AdeptID's highest priority signal (minimum years of experience required *based* on education), we successfully demonstrated viability of knowledge distillation from a larger, expensive transformer LLM into a smaller, portable encoder-only LLM, fine tuned for application in this multi-label, multi-class classification task. Fine tuning a smaller fit for purpose model optimized for a specific task minimizes inference cost and latency.**

***Keywords* – knowledge distillation, fine tuning, LLM, Gemini, BERT, multi-class multi-label text classification.**

I. Introduction

Artificial intelligence start-up AdeptID develops software and solutions designed to accelerate the alignment of talent and opportunity, matching qualified job seekers to relevant jobs. Its platform is used more than a million times a month to connect people, training and jobs, creating value for individuals seeking employment and hiring businesses.

To successfully pair job seekers with relevant opportunities across all occupation types, it is critical to accurately extract signals from job postings, such as experience and education requirements. Although traditional parsing methods can successfully find keywords, signals can be complex and nuanced with interdependencies or multiple options listed in a single posting (i.e., Bachelor's degree + 3 years' experience or Master's degree + 1 year experience).

With the proliferation of recruiting via mobile or social media applications and the expansion of online recruiting to most types of labor, the unstructured text in job postings is becoming even more challenging to parse via traditional means. To maximize user value, AdeptID is attempting to improve extraction of nuanced job requirements from raw job posting text in a manner that generalizes well to less structured text.

Business Value – Creating a more efficient and accurate job matching service helps businesses hire and individuals seeking jobs, which comprise a large total addressable market (TAM). Businesses need to identify talent to meet recruiting required for growth and remain competitive. Ninety-nine percent of Fortune 500 companies rely on Applicant Tracking Systems (ATS), with large companies spending $125k annually for these systems.[1] Traditional ATS systems simply parse text for keywords.

From a job seeker perspective, more than 30% of Internet users between 18 and 49 years old in the U.S searched online job sites in 2022, and nearly 20% of individuals over 49.[2] Those who use recruitment services to be placed with a company may pay 15%-25% of their first year salary in fees to the recruitment company.[3]

Existing tools, such as ATS, rely on keyword matching to pair job seekers with opportunities. AdeptID has found that traditional keyword parsing systems struggle to process language variations and fail to identify nuanced, interdependent job requirements. This can ultimately lead to missing qualified candidates during the search process. It can also lead to inefficiency arising from misinterpreted postings, wasting time for both the job seeker and the hiring company.

Finding an efficient solution for the shortcomings of these traditional systems could improve performance of job matching, surfacing the right candidate for a job at scale, and provide a compelling alternative, cutting into the large market share of traditional systems. By improving the generalization of signal extraction, there may also be a reduction in the inherent bias in the traditional systems originally designed for white-collar, higher-education jobs.

[1] Myers, S. (2024, May 8). *2023 Applicant Tracking System (ATS) usage report: Key shifts and strategies for job seekers*. Jobscan. https://www.jobscan.co/blog/fortune-500-use-applicant-tracking-systems/

[2] Statista. (May 8, 2023). Share of internet users who searched job sites online in the United States in 2022, by age. In Statista. https://www.statista.com/statistics/479216/internet-users-who-engaged-in-a-job-search-on-computer-within-the-last-month-usa/

[3] Mellado, R. V. (2024, May 2). *Applicant Tracking System Pricing Guide (updated 2024) - SSR*. RSS. https://www.selectsoftwarereviews.com/blog/ats-pricing-short-guide

Business Problem – AdeptID is looking for a low-cost method with quick inference to extract important key signals from job postings, converting raw text into a structured output of signals that can be matched with the resumes of job seekers and improve the basis for their matching service. The method must be able to generalize well to unstructured text with challenging wording and colloquialism. While human readers can naturally "read between the lines", this is a complex task for machines. Several signals of interest were prioritized by AdeptID. Given the timeline constraints of the Summer Practicum, we focused our project on the highest priority signal: minimum years of experience required *based* on education. The complexity and nuance of this signal has been historically difficult for the business to solve; it requires extracting interdependency with multiple options from text and handling cases when there simply isn't a requirement mentioned in the text.

Technical requirements – While AdeptID encouraged creative approaches, proposed solutions must have the ability to scale and adhere to some critical technical constraints:

- *Input/Output* - inference pipeline developed must input raw job posting text and output structured data object with signals
- *Latency* – minimal inference latency (< 2 sec)
- *Cost* – minimal inference cost
- *Performance* – must be measured for each signal and occupation; False Positives are more detrimental to the user than False negatives
- *Privacy* – many laws and regulations govern the hiring of employees; any proposed solution must comply with these laws and regulations and not introduce biases against job candidates

II. Dataset & EDA

AdeptID provided a database of 1,226,569 job postings scraped from the Internet.

Features – The raw job description text for each posting is stored in the *body*. There are also fields indicating the type of occupation, generally derived from the standardized *ONET* code. While the *body* text is the primary feature as defined by the technical requirements, *ONET* is a supplemental feature we also explored. We hypothesized that postings for specific occupations may have similar requirements, so this additional feature could help signal extraction when the text is unclear.

Parsed Labels – AdeptID also provided signal labels of unknown accuracy from their 3rd party text parser. Relevant labels include *min_edulevels*, *min_years_experience*. The education label is an array of codes indicating a degree level or no education listed; the mode is "No education listed" (40%), followed by "Bachelor" (26%) and "High School/GED" (23%). Other smaller education classes include "Associate", "Master", and "PhD/Professional". The experience label is an integer for 0-15 years or 99 if no experience requirement is mentioned. Cursory inspection indicated label accuracy may be limited and warranted further evaluation.

Exploratory data analysis (EDA) of the job postings revealed several challenges that must be addressed:

1. While all postings had some *body* text, some of the scraped text was irrelevant or not a job posting (i.e., 404 error, posting closed, auto-response...). These records should be removed. Generally, within the dataset provided, postings with <50 characters should be removed.
2. Many postings have lengthy *body* text that will arbitrarily be truncated by most pre-trained tokenizers. Figure 1 shows the histogram of word counts. Assuming 0.75 tokens per word, the average posting token length is 573 tokens, which is beyond most tokenizer limits (generally 512). A method of extracting relevant text from the full posting text is required or else we risk truncating the relevant text from the posting during tokenization (Appendix C – Extraction Performance). Anecdotally, requirements seem to appear later in the posting test so truncation from the *left* may be better than truncation from the *right* in this application.

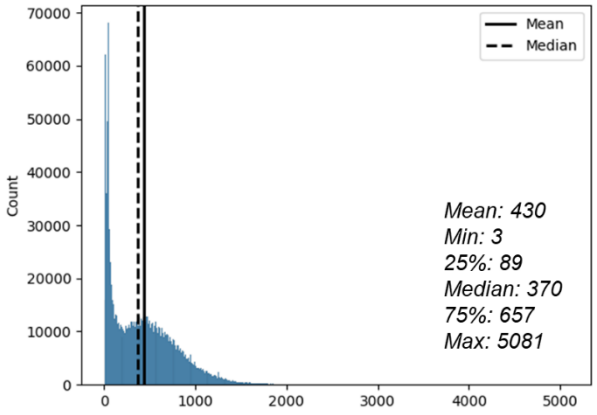

Figure 1- Histogram of job posting word counts.

3. Accuracy of the parsed labels is low (Appendix B – Label Generation Performance) and fails to capture the relationship between education and experience, along with multiple combinations per posting. Anecdotally, we found discrepancies in the parsed labels; 475 postings require H.S. or Ph.D., but nothing between. As a quick proof of concept during EDA, we used the parsed labels to train a classification model and saw poor performance.

Considering supervised learning on the parsed labels will not exceed the performance of the parser, a more accurate labeling method is required for any supervised learning approach.

4. Minimizing cost and latency will likely require a smaller model that can be hosted by AdeptID. Exploration during EDA of zero-shot and semi-supervised learning suggested that desired performance would require a much larger language model, resulting in higher cost and latency. Therefore, a supervised approach focused on the specific signal extraction task may be fit for purpose.
5. The parsed labels indicated highly imbalanced classes, especially when combining education and experience, and revealed many instances of no education or experience requirements mentioned in the posting. This was later confirmed during label generation. Considering all the classes are of equal importance, our approach must handle class imbalance in training and the cases when there is no relevant information in the text.
6. Reading through some of the postings, there were some examples with requirements that contradicted or were very difficult to interpret. Considering how challenging it is for a human reader in these instances, there is some level of subjective noise in that data that makes perfect performance difficult to achieve.
7. Since each posting can have multiple requirements, if any, and the signal classes are discrete, the problem can be approached as multi-label, multi-class text classification. This has implications on how to properly architect the loss function or optimization objective during machine learning.

III. Methodology

In order to solve the business problem and satisfy the technical requirements, our general approach was to train a model using the provided data and create a pipeline that can make inference using the trained model. However, as discovered in EDA, several challenges must be addressed. The pipeline can be summarized in the following steps:

Clean – In order to extract the complex target signal, text embedding must sufficiently gather context that may be spread out of several sentences. Simpler embedding methods (i.e., TF-IDF) do not capture context and word embedding requires arbitrary aggregation at the document level, so we decided on SBERT sentence embeddings using a pre-trained tokenizer from Hugging Face trained on a large corpus of English text. Since BERT models can adequately handle punctuation and other text complexity, minimal text cleaning was required; spaces were normalized and emojis were removed. We also leveraged regex to remove erroneous records from the dataset. To maximize the speed of this step, cleaning was implemented using parallelization via Dask.

Sample for training – Considering the cost of labeling and class imbalance, we used a sample of 50k postings from the 1.2M provided, stratified to balance education and experience classes as best as possible based on the parsed labels. Many of the postings were missing education or experience according to the parser; the parsed labels also suggested that adding more postings to the sample would likely be adding more examples of the majority class. Once the 50k sample was cleaned, extracted, and labeled, we split the sample into 80% train, 10% validation, and 10% test datasets, stratified to help address class imbalance using skmultilearn designed for multi-label data. We dropped any class with <10 instances.

Label for training – With the limitations of the parsed labels revealed during EDA, more accurate labels were required. Researching SOTA Large Language Models (LLMs) that leverage transformer architecture, we found that modern LLMs with extremely large capacity and training could produce quality labels after prompt engineering. However, these large models can have billions of parameters, making them too large to host locally, be costly to use since they are a paid service, and have constrained API limits. Since the business technically requires a low cost, fast solution, these LLMs are not fit for purpose to process every inbound job posting. Instead, we can extract knowledge from a more powerful LLM (teacher model) to train a more fit for purpose, lightweight encoder-only model (student model) designed specifically for our signal extraction task; this is the concept of knowledge distillation. For text classification task at hand, a full transformer model is generally not required.

After testing a few options, we selected Google's Gemini 1.5-flash-001 LLM based on cost, schedule (API limits), output quality and batch processing (Appendix A – LLM Cost & Schedule Analysis). To extract knowledge from this LLM, we engineered a prompt specific for the LLM that could process a posting and return structured multi-labels, listing combined education and experience requirements. By leveraging batch processing via the Vertex Gemini API, we labeled 50k postings for <$30 with 400 ms per inference. As part of the output, we also had the LLM advise the first few words of the relevant text so we could locate the relevant text and its relative positioning inside the posting in hopes this would inform our extraction/truncation strategy.

We tested the quality of the label output by assessing performance against a small set of 58 manually annotated postings with 75 labels between them (Appendix B – Label Generation Performance). Generally, the LLM performed well at identifying the combined education and experience requirements when they existed, but struggled to detect when neither was mentioned. The parsed labels appeared to perform in the opposite manner, failing to detect all

combinations, but sufficiently determining when there was no mention of an education or experience requirement. For this reason, we chose a hybrid approach to label generation that combined the LLM labels and parsed labels, which resulted in the best performance on our manually annotated set.

Extract relevant text – In order to prevent arbitrary truncation of relevant text during tokenization, we implemented a regex-based method to extract sentences that seem relevant to the signal, while removing the rest (Appendix C – Text Extraction Performanc). We attempted a few truncation strategies; our initial attempt was a simple heuristic based on the position of education and experience descriptors within the job description. The heuristic was formed using relative positioning data output from the LLM during label generation. However, modeling showed this approach would only result in extracting the correct descriptive text in about 19% of the job descriptions, as shown in Figure 2, indicating the relevant text is not positioned consistently enough across all postings. The final regex-based method extracted text for 99.98% of the 50k sample postings and reduced the likelihood of truncation at 512 tokens from 87% to 3%.

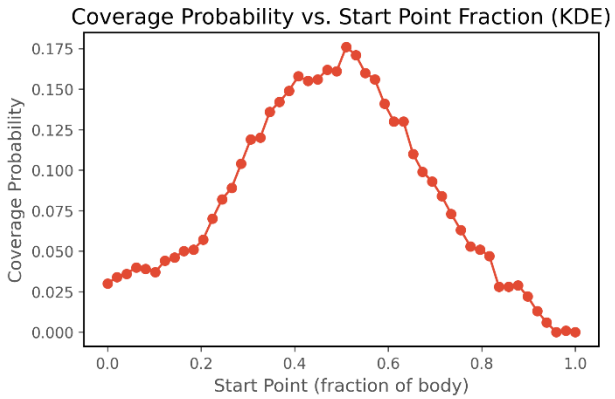

Figure 2 - Kernel density modeling of expected coverage of correct descriptive text using positional heuristics.

Train multi-label classifier – We architected a classifier for this task by leveraging a base pre-trained encoder-only BERT model from Hugging Face and adding a classification head in PyTorch specific to our multi-label task. The classification head includes a linear layer corresponding to the number of classes along with a dropout layer to reduce overfitting. The classifier module we developed includes customizable training loops with model checkpointing and data prep that leverages a pre-trained tokenizer from Hugging Face corresponding to the selected base BERT model. Since this is multi-label problem, BCEWithLogitsLoss was used as a loss function to return logits for all possible classes, that can then be converted to probabilities of each class using a sigmoid function. Considering the class imbalance, relative class weights were input to the loss function to treat each class equally.

The base BERT model was fine-tuned for our task by training the architected classifier on our 80% training set, initialized with the pre-trained weights from Hugging Face. A Google Colab GPU was leveraged for training. Various base models (bert, distilbert, distilroberta, roberta) with different model configurations were compared by evaluating performance on the validation set. Loss curves were produced for the training and validation sets to detect overfitting. Weight decay and dropout helped reduce overfitting. Adam optimizer converged more quickly than SGD.

The model configuration for the chosen model is shown in Figure 3, performing best on the validation set after 11 epochs of training.

Model configuration	
label version	'labels_hybrid_v1'
base	'distilroberta-base'
family	'roberta'
learningRate	1e-5
maxLen	512
batchSize	32
N_EPOCHS	15
truncationSide	'left'
dropout	0.1
lowerCase	False
optimizer	'Adam'
wtDecay	1e-4
momentum	0
n_layers	768
onet_feature	False

Figure 3 - Model configuration for selected model (16)

Loss curves for this model are shown in Figure 4 and validation performance for evaluated models is shown in Appendix D – Validation Performance.

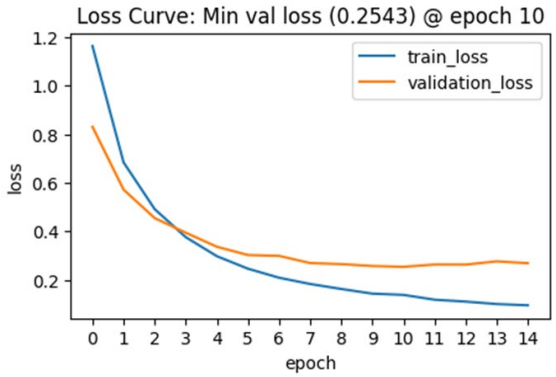

Figure 4 - Loss curves for selected model to help check overfitting and select best epoch.

Inference from classifier – In order to use the model, we developed a pipeline module to pre-process raw input text and make inference from the fine-tuned model. Per the technical requirements, the input to the pipeline is raw posting text and the output is a structured data object, listing all returned class labels as (education, experience) tuples, each with a probability score. The pipeline can handle single or batch inference.

Our self-contained, fit for purpose modeling approach enables AdeptID to customize the model as desired and host it locally, removing dependency on 3rd party APIs/pipelines and their associated costs. Rather than interact with the base model through Hugging Face API or methods, we simply initialize our self-contained deep learning architecture with the pre-trained model weights available open source. Limited dependency helps minimize cost; the only cost would be server costs to host the model, which should be nominal to the enterprise for our lighter weight model. There is an additional cost to generate labels for training, however this should conceptually be a one-time or infrequent activity.

Classifier performance – In order to assess the expected performance of the selected model, we developed a metrics module to generate performance reports that assess key metrics, which are further defined in the Results section. Per the technical requirements, these reports show classification performance by class and ONET occupation. Our performance evaluation also looks at challenging edge cases to determine limitations of the model. Test results for the selected model are shown in the Results section.

The architecture behind our pipeline modules is summarized in Appendix F – Architecture. Code is available in GitHub repo: https://github.com/kakral/adeptID_7/tree/main.

IV. Results

The model with configuration listed in Figure 3 was selected based on validation performance (Appendix D – Validation Performance). It leverages a distilroberta base model, which outperformed the distilbert base model at the cost of +24% model complexity (83M parameters vs. 67M).

Key performance metrics – Accuracy is defined differently in a multi-label setting, when a given prediction may be partially correct. Hamming Loss is the ratio of incorrect labels to correct labels; Hamming Accuracy = 1 – Hamming Loss, so a high Hamming Accuracy indicates many more correct labels than incorrect labels; it is a measure of accuracy at the label level. Zero one (0/1) Accuracy is much stricter since it is the fraction of datapoints with all labels correct; it is a measure of accuracy at the datapoint (or posting) level.

Since False Positives are more detrimental to users than False Negatives, we are also interested in maximizing Precision, so Precision-Recall trade-off is evaluated as shown in Figure 8. Recall and precision for the selected model vs. probability threshold is shown in Figures 6 and 7 below).

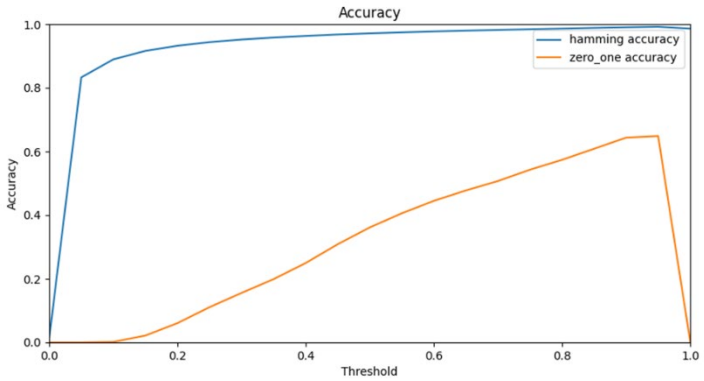

Figure 5 – Test dataset accuracy for selected model vs. probability threshold for classification.

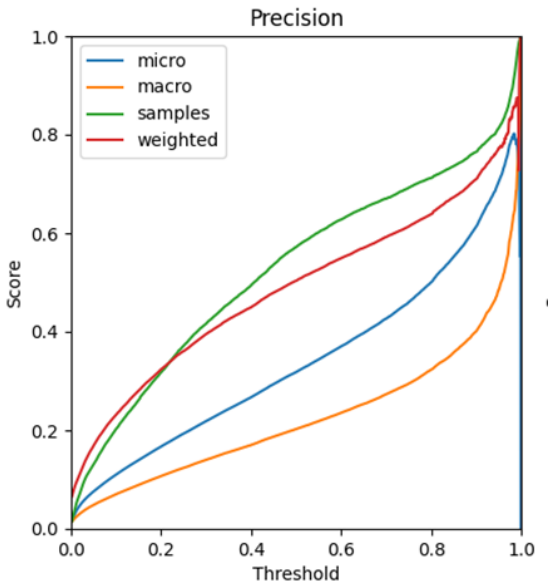

Figure 6 - Test dataset precision for selected model vs. probability threshold for classification.

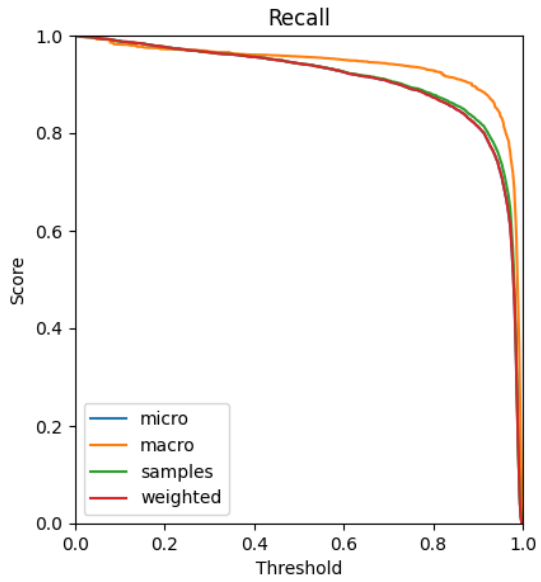

Figure 7 - Test dataset recall for selected model vs. probability threshold for classification

Since total Precision and Recall must be averaged over the dataset, we chose to use samples averaging (average over postings), rather than class averaging.

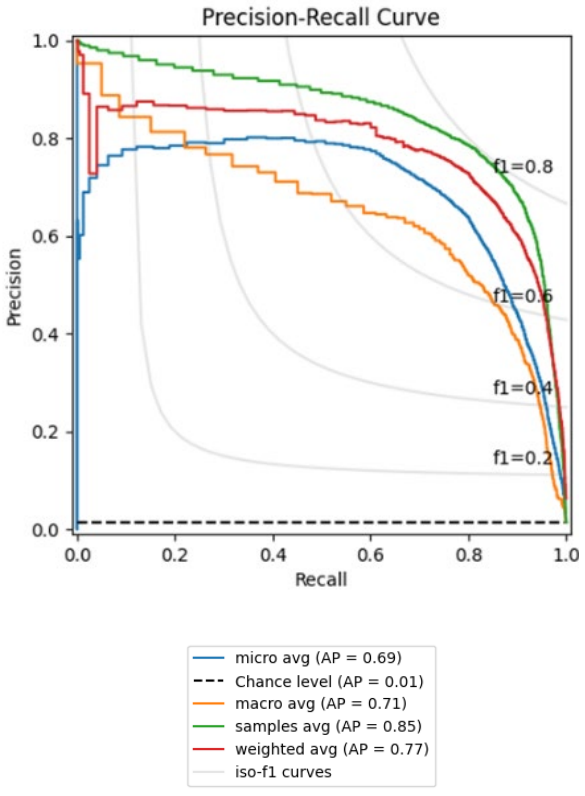

Figure 8 - Test dataset precision-recall curve for selected model

With a 0.92 probability threshold for classification, the selected model showed the following performance (Appendix E – Test Performance):

Selected Model Test Set Performance	
Hamming Accuracy	99%
0/1 Accuracy	66%
Precision	79%
Recall	80%

As the threshold increases, Precision increases at the cost of Recall and 0/1 Accuracy. With a 0.96 threshold, Precision increases to 83%, while 0/1 Accuracy drops to 63% and Recall drops to 71%. 0/1 Accuracy drops off steeply for thresholds above 0.96.

Inference latency – Due to the use of a lightweight model (83M parameters) fine-tuned for a specific task and parallelized pre-processing, inference latency was minimized and demonstrated to be well below the 2 second technical requirement when using a Google Colab T4 GPU:

Selected Model Inference Latency	
Single	288 ms
Batch (per inference)	19 ms

Inference cost – As mentioned in the Methodology section, our self-contained model/pipeline was designed to remove 3rd party dependencies. The only cost required for operational inference is the cost of hosting the model, which can be done locally by AdeptID given the model size (83M parameters).

Model limitations – Below are actual example inputs and outputs of our pipeline demonstrating its abilities and limitations for some challenging cases. The output for each raw text input is a list of tuples, each comprised of an (education, experience) tuple and probability score. Education is encoded (0 = High School/GED, 1 = Associate's, 2 = Bachelor's, 3 = Master's, 4 = Doctorate, 99 = None mentioned). Experience represents the minimum number of years experience required for that education level; it is a float since we were able to successfully label partial years of experience (e.g., 6 months) during LLM label generation.

Example 1
Input text `"In need of a skilled data scientist. Bachelor's degree with 7+ yr experience or Master's with 5 years experience. If you have Ph.D. then no experience required. Competitive pay. DM me if interested."`
Output `[`

```
    ((2.0, 7.0), 0.9400),
    ((3.0, 5.0), 0.9796),
    ((3.0, 0.0), 0.9342),
    ((4.0, 0.0), 0.9499)
]
```

The first example shows how the model can extract multiple combinations, successfully extracting all 3 experience requirements dependent on education level. It also shows how well the model handles nuanced ways of writing the experience requirement (i.e., "7+ yr", "5 years", "no experience"). However, with the probability threshold of 0.92, an additional incorrect label was returned.

Example 2

Input text

```
"I'm looking for an electrician. They
need to have a Journeyman electrician
license. Minimum of 1 year experience. I
would like them to have 2 years of
experience, but I'd be open to working
with somebody with less experience. We
want them on site 3 days a week. Pay is
$115. I don't need somebody with a PhD,
I just want a really great electrician."
```

Output

```
[
    ((99.0, 2.0), 0.9467),
    ((99.0, 1.0), 0.9246)
]
```

The second example shows how the model can handle postings that do not explicitly list an education degree and navigate irrelevant mentions of a degree (i.e., "I don't need somebody with a PhD") that may confuse a traditional parser. However, the model struggled to differentiate between the minimum requirement and preference, returning labels for both; the preference achieved a higher probability score than the minimum requirement.

For each example and others tested, further manipulation of the input text showed that the performance depends on the wording to some extent and the class itself. Simply changing the integer for years of experience in the text to make the label that of a much smaller training class size, reduced performance. Also, not every possible class was included in the dataset (each education class + 0-15 years experience). When the posting includes a label for a class that wasn't in the dataset, the model fails to return any label with a meaningful probability score.

ONET performance – classification reports were produced for ONET Pathways and Clusters in addition to classes; they are available in the results folder in the GitHub repo. Performance appears to differentiate by occupation type, indicating higher performance could be achieved when limiting the model to select occupations. Below are the top 3 and bottom 3 ONET Pathways by 0/1 Accuracy for the selected model using the test set with 0.92 probability threshold.

Top 3 ONET Pathways: 0/1 Accuracy	
Personal Care Services	95%
Emergency & Fire Mgmt Services	80%
Production	76%

Bottom 3 ONET Pathways: 0/1 Accuracy	
Marketing Mgmt	52%
Family & Community Services	50%
Manufacturing Prod Process Dev	48%

We also created a modified version of the selected model that takes ONET (parent, child) features as input in addition to the raw posting text, however this modification did not improve performance (Appendix D – Validation Performance).

V. Conclusion & Next Steps

Although the model does have some limitations, we successfully demonstrated a proof of concept for AdeptID that could improve signal extraction from job postings for the highest priority signal: minimum years of experience required *based* on education. Not only did we demonstrate viability of our approach on the dataset provided, but also, we developed the framework for a LLM label generation batch process, customized fine tuning of LLMs, performance measurement, and inference that could be applied for extraction of other signals beyond education and experience, such as role type (office vs. hybrid vs. remote) or compensation. We encountered several challenges (see EDA) that we were able to successfully work through (see Methodology), with learnings likely applicable to other related work at AdeptID.

Our final approach distilled knowledge from a larger, expensive transformer LLM into a smaller, portable encoder-only LLM, fine-tuned for signal extraction, which we framed as a multi-label, multi-class classification task. Fine tuning a smaller model for a specific task enabled us to minimize inference cost and latency. The self-contained nature of our solution doesn't require any costly 3rd party APIs or LLMs for inference and can be quickly customized internally by AdeptID in the future if needed. Although there is some cost associated with up front label generation, this is a one-time cost for training that is not applicable to inference. The final trained model is small enough to be hosted by AdeptID, so the only inference cost would be the infrastructure cost to serve an 83M parameter model.

Our approach satisfies all technical requirements defined by AdeptID, producing structured signal output from

raw job posting text. While we recommend our general approach, there are some opportunities to boost performance:

More training data – As mentioned in the Results, performance was somewhat dependent on wording and the size of the class in the training dataset. Some possible classes had no training instances. Figure 4 shows performance was materially higher for larger training classes, achieving >90% 0/1 Accuracy and Precision when there were at least ~2500 instances of the class in the training set. Adding more training data balanced across all possible classes or grouping some classes together to increase the instances per class may help improve performance. The 1.2M postings are heavily class imbalanced, so simply using more of the provided postings beyond the 50k sample will likely only increase the size of the majority classes. Data augmentation could also help generate more training instances for minority classes; it may also help improve generalization, making model performance less dependent on wording.

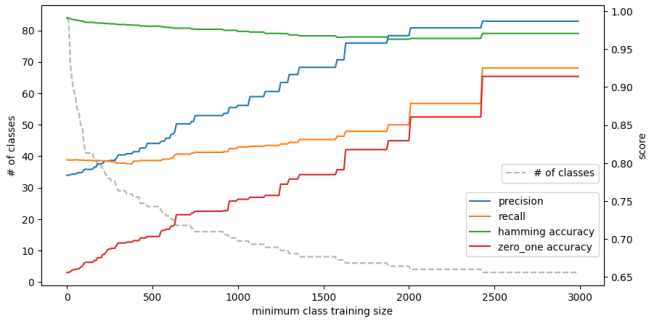

Figure 9 – key performance metrics for selected model on test set vs. minimum class training size.

Improving label generation and text extraction – our selection of an LLM for label generation was constrained by the minimal project budget. There is opportunity to improve accuracy of the labels generated (Appendix B – Label Generation Performance) by using a higher capacity LLM or manual annotation, both of which can be costly. In supervised learning, model accuracy cannot exceed that of the labels themselves, so higher quality labels will likely boost model accuracy. Our regex approach to text extraction was constrained by the short project schedule. The regex approach could be refined with further research and calibration of the search pattern or replaced by a chunk classifier that breaks the text into smaller pieces and classifies them as relevant or not. Improved text extraction could help improve learning by ensuring all relevant text is returned and minimizing irrelevant text that could dilute the embedding.

Output handling – As seen in the example outputs in the Results, incorrect labels can be returned with correct labels, even with high a high probability score threshold. In some cases, the model may return multiple experience requirements for the same level of education (e.g., Bachelor's + 2 years experience and Bachelor's + 1 year experience). This may be a result of model error, failure for the model to differentiate between a minimum requirement vs. a preference or contradictions in the posting text itself. A handling scheme could help reduce these overlapping labels into one and create a buffer that tolerates some level of error. Possible handling may be to filter to labels that achieved some minimum threshold, then prioritize one label for each education label returned based on the probability score. Prioritization could be implemented by independent soft max on each education class set. Some error tolerance could be created by a +/- band on the returned minimum years of experience.

VI. References

Sun, C., Qiu, X., Xu, Y., & Huang, X. (2020, February 5). *How to Fine-Tune BERT for Text Classification?* https://arxiv.org/pdf/1905.05583

APPENDIX A – LLM COST & SCHEDULE ANALYSIS

PRICING							
Model	**Requests per Day**	**Requests per Min**	**Tokens per Min (k)**	**Tokens per Day (k)**	**Token Window (k)**	**Input Price (per 1M tokens)**	**Output Price (per 1M tokens)**
Google Gemini 1.5 Flash		1000	2000		128	$0.35	$1.05
Anthropic Claude 3 Haiku		5	25	300	200	$0.25	$1.25
OpenAI ChatGPT 3.5	200	3	40000		16.385	$0.50	$1.50
OpenAI Batch (24 hr)	50000					$0.25	$0.75
AWS Bedrock							
Titan Text Express		400	300		8	$0.15	$0.20
Titan Text Lite		800	300		4	$0.20	$0.60
Claude 3 Haiku		1000	2000		200	$0.25	$1.25
Mistral 8*7B		400	300		32	$0.45	$0.70
Metla Llama 3 (8B)		800	300		8	$0.40	$0.60

Table 1.1 – LLM pricing breakdown by model

TOKEN ASSUMPTIONS	
Contingency	10%
Prompt token length	521
Avg Posting token length	631
Output token length	110

Table 1.2 – Token assumptions for estimating cost & schedule

ESTIMATED COST ($)				
Labels (k)	**10**	**50**	**100**	**1000**
Google Gemini 1.5 Flash API	5	26	52	519
Anthropic Claude 3 Haiku API	4	21	43	426
OpenAI ChatGPT 3.5 Turbo-0125 API	7	37	74	741
Batch API (24 hr turn-around)	4	19	37	371
AWS Bedrock				
Titan Text Express	2	10	19	195
Titan Text Lite	3	15	30	296
Claude 3 Haiku	4	21	43	426
Mistral 8*7B	6	30	60	595
Metla Llama 3 (8B)	5	26	53	527

Table 1.3 – Estimated cost in $ by model based on number of labels

ESTIMATED TIME (hrs)				
Labels (k)	**10**	**50**	**100**	**1000**
Google Gemini 1.5 Flash API	0.2	0.8	1.7	16.7
Anthropic Claude 3 Haiku API	1009.7	5048.3	10096.5	100965.3
OpenAI ChatGPT 3.5 Turbo-0125 API	1200	6000	12000	120000
Batch API (24 hr turn-around)	24	24	48	480
AWS Bedrock				
Titan Text Express	0.7	3.5	7.0	70.1
Titan Text Lite	0.7	3.5	7.0	70.1
Claude 3 Haiku	0.2	0.8	1.7	16.7
Mistral 8*7B	0.7	3.5	7.0	70.1
Metla Llama 3 (8B)	0.7	3.5	7.0	70.1

Table 1.4 – Estimated time in hrs by model based on number of labels

APPENDIX B – LABEL GENERATION PERFORMANCE

		Accuracy		**Samples Avg**		
LabelGen Method	**Prompt**	**Hamming**	**0/1**	**Precision**	**Recall**	**F1**
Parsed (provided)	---	0.98	0.64	0.74	0.68	0.70
LLM (Gemini 1.5 Flash)	v1	0.98	0.48	0.54	0.54	0.54
LLM (Gemini 1.5 Flash)	v2	0.98	0.57	0.59	0.60	0.59
Hybrid (v1, Parsed + LLM)	**v2**	**0.99**	**0.79**	**0.82**	**0.82**	**0.82**

Table 2 – Label generation performance across methods on manually annotated sample of postings

APPENDIX C – Text Extraction Performance

		% of postings dataset truncated by tokenizer		
Posting text	**Max token length**	**Has parsed min education label**	**Has parsed min experience label**	**Has parsed min education and experience labels**
Raw	512	83.3%	81.0%	86.7%
	256	98.0%	66.8%	98.6%
	128	99.7%	99.5%	99.9%
Extracted via our regex method	512	2.2%	2.5%	2.5%
	256	15.5%	16.4%	16.4%
	128	37.7%	39.3%	39.3%

Table 3 – Extraction performance for 50k sample showing materially less postings requiring truncation after extraction

APPENDIX D – VALIDATION PERFORMANCE

								Accuracy		**Samples Avg**		
Base model/tokenizer	**Features**	**~Params**	**Learning Rate**	**Dropout**	**Wt Decay**	**Best Epoch**	**Optimal Threshold**	**Hamming**	**0/1**	**Precision**	**Recall**	**F1**
distilbert-base-cased	text	66M	1E-04	0.5	0	8	0.55	0.92	0.00	0.01	0.05	0.01
distilbert-base-uncased	text	67M	1E-04	0.5	0	5	0.50	0.52	0.00	0.02	0.60	0.03
distilbert-base-uncased	text	67M	1E-03	0.5	0	5	0.50	0.55	0.00	0.02	0.58	0.03
distilbert-base-uncased	text	67M	1E-05	0.5	0	8	0.90	**0.99**	0.60	0.77	0.78	0.72
distilbert-base-uncased	text	67M	1E-06	0.5	0	10	0.70	0.93	0.21	0.52	0.58	0.33
distilbert-base-uncased	text	67M	1E-05	0.3	0	8	0.90	**0.99**	0.61	0.75	0.81	0.74
distilbert-base-uncased	text	67M	1E-05	0.1	0	8	0.90	**0.99**	0.63	0.77	0.81	0.75
distilbert-base-cased	text	67M	1E-05	0.1	0	8	0.90	**0.99**	0.62	0.77	0.81	0.75
distilbert-base-cased	text	67M	1E-05	0.1	1E-04	9	0.90	**0.99**	0.65	0.77	0.81	0.77
distilbert-base-cased	text	67M	1E-05	0.1	1E-03	10	0.90	**0.99**	0.57	0.75	0.74	0.68
bert-base-cased	text	111M	1E-05	0.1	1E-04	7	0.90	**0.99**	0.57	0.75	0.74	0.68
roberta-base	text	126M	1E-05	0.1	1E-04	10	0.90	**0.99**	**0.67**	**0.78**	**0.84**	**0.78**
distilroberta-base	text	83M	1E-05	0.1	1E-04	11	0.92	**0.99**	0.66	**0.78**	0.81	0.76
distilroberta-base	text + ONET	83M	1E-05	0.1	1E-04	10	0.9	**0.99**	0.60	0.76	0.74	0.70

Table 4 – Validation dataset performance of various models, leading to selection of distilroberta-base text only model

APPENDIX E – TEST PERFORMANCE

								Accuracy		Samples Avg			Inference Time*	
Base model/tokenizer	Features	~Params	Learning Rate	Dropout	Wt Decay	Best Epoch	Optimal Threshold	Hamming	0/1	Precision	Recall	F1	Single	Batch (per inference)
distilroberta-base	text	83M	1E-05	0.1	1E-04	11	0.92	0.99	0.66	0.78	0.80	0.76	288 ms	19 ms

Table 5 – Test dataset performance for selected model; *inference using Google Colab T4 GPU

APPENDIX F – ARCHITECTURE

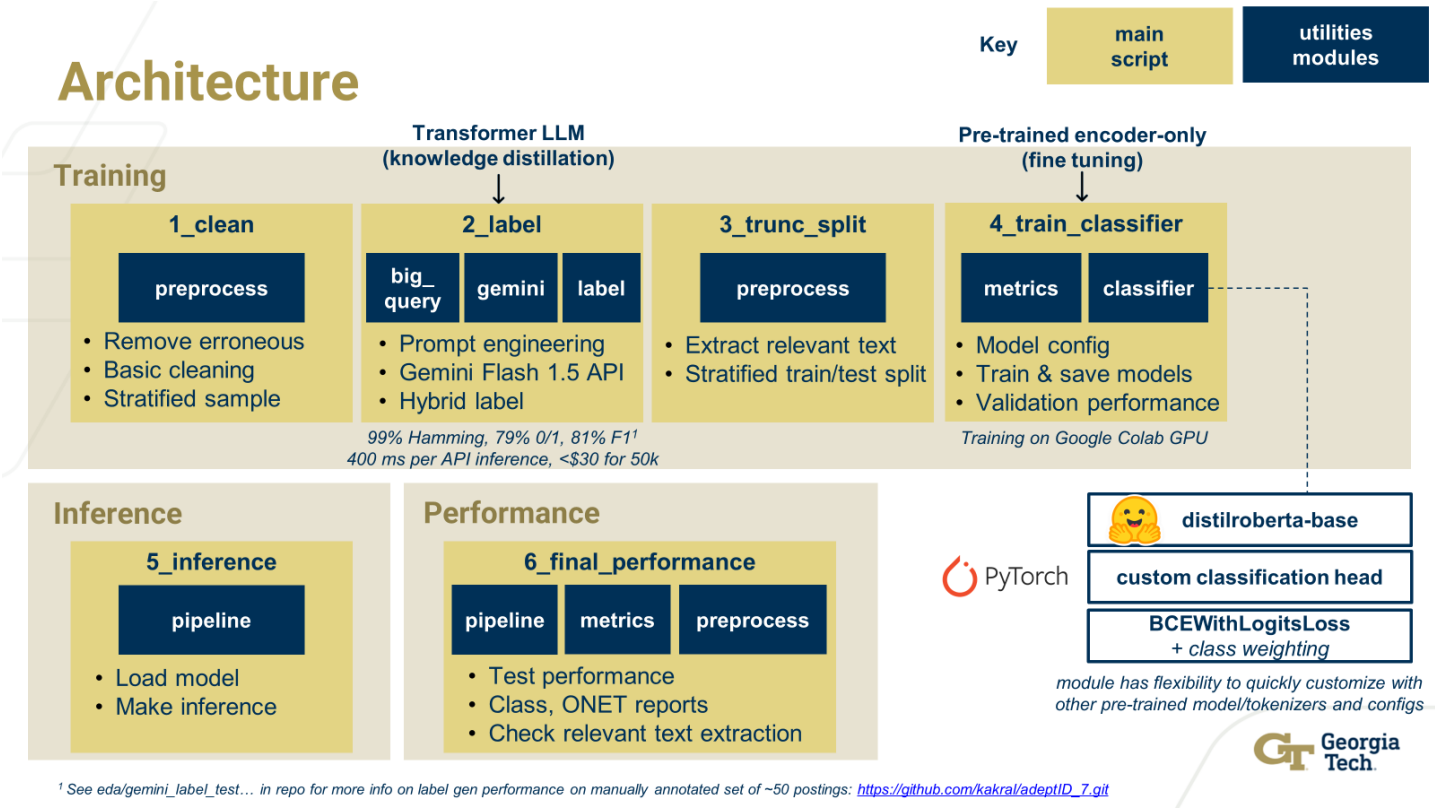

Figure 9 – Structure of provided solution