# Structured Information Extraction from Job Descriptions using Transformers

Team-09: Krystian Szczepankiewicz, Ming Kai Hsu, Xinyu Lan, Pranav Kuchibhotla, Melissa Laiz

# Objectives & Background

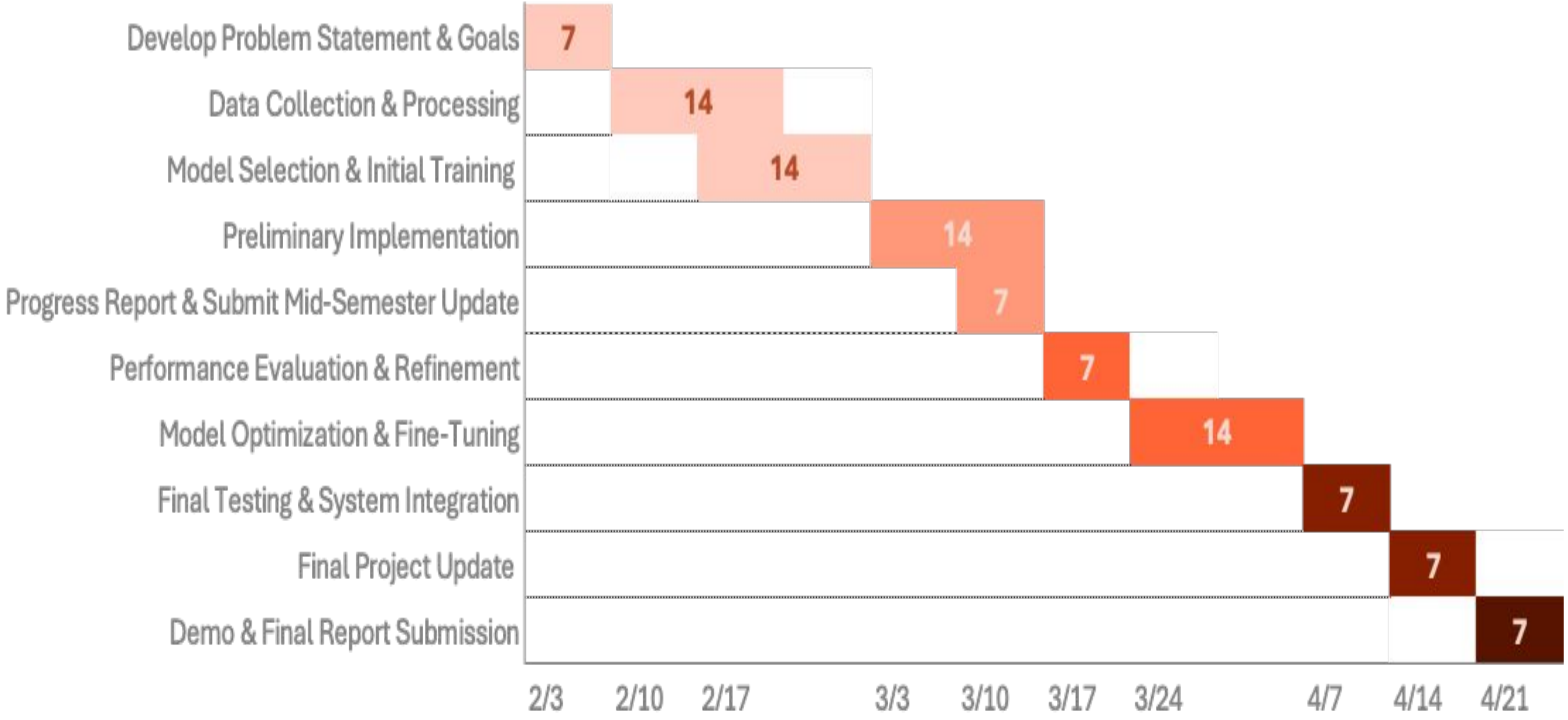
## Objective:

1. Develop an AI model to extract structured job details from unstructured job descriptions.
2. Identify key attributes like Job Title, Skills, Experience, Location, and Responsibilities using NLP and Transformer models.
3. Improve job search efficiency by providing categorized, structured, and ranked job listings.

## **Background**:

1. Traditional job search platforms return broad and often irrelevant results.
2. Unstructured job descriptions make it difficult to filter and match relevant roles efficiently.
3. Manual job filtering is time-consuming and lacks personalization. AI-based parsing can automate extraction, improve job-to-candidate matching and save time.

# Blueprint/Timeline



# Dataset Research

## Data Collection:

1. **Web Scraping:** Extract job postings from platforms like LinkedIn, Indeed.
   - Tools: LinkedIn unofficial api, Linkedin scraper
2. **Public Datasets:** Utilize open job posting datasets from Kaggle
   - LinkedIn Job Posting
3. Focus on gathering diverse job listings to improve model generalization.

## Data Preprocessing:

1. **Text Cleaning:** Remove HTML tags, special characters, and stopwords.
2. **Segmentation & Parsing:** Convert job information into semi-structured text prompts before input to Language Model.
3. **Supervised Labeling :** Use Deepseek api to label job descriptions ➔ JSON format and build high-quality dataset.
   - Price: Deepseek api $0.3/ million tokens; GPT 4o api $5/ million tokens;

# Challenges for Phase I

1. **Job Data Retrieval:** How to get large number of job descriptions efficiently
   - Solution: We used an unofficial LinkedIn API retrieval method from a GitHub repository to pull raw job data.
2. **Data Labeling:** Manual labeling of large datasets is too time-consuming.
   - Solution:  Used DeepSeek API for advanced text cleaning, and structured extraction to job postings before feeding them into AI models for training.
3. **Model Selection Uncertainty:** Selecting the best AI model for job extraction is challenging.
   - Solution: Conducted background research and leveraged insights from ChatGPT and academic papers compare model performance. We compared models like BERT, T5, and open source LLMs for accuracy and efficiency. Selected T5-XL as the optimal model for structured job extraction.

# Next Steps and Potential Challenges to Phase 2:

- **Data Preprocessing & Data Cleaning**: Standardize job descriptions by removing HTML tags, stop words, and redundant text. With the implementation of LLM to extract key job attributes such as required skills and job description.
- **Database Optimization**:  Store extracted and structured job data likewise in databases like MongoDB, SQLite, and PostgresQL for scalable storage and fast retrieval. Also creates an efficient job searching.
- **Fine-Tune T5-XL**: Train and evaluate using benchmarking metrics for structured job data extraction
- **API Cost Management for Scalability**: DeepSeek API requires purchasing tokens; we need an efficient token budgeting strategy to ensure long-term scalability and storage of data.

# PHASE II

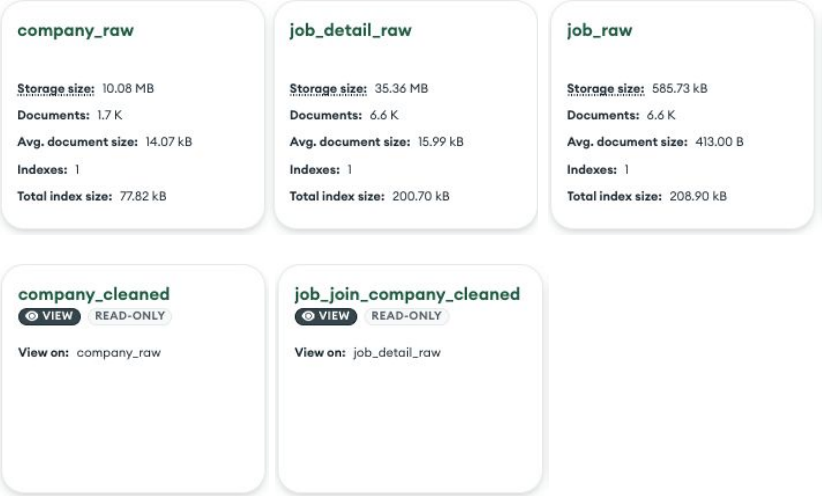
# Outline

- Data Storage Setup
- Data fetching pipeline
- Data cleaning and data Labeling
- GPU Server Deployment
- Model Training Pipeline
- Challenges faced and Solutions
- Issues to be resolved
- Blueprint of next steps

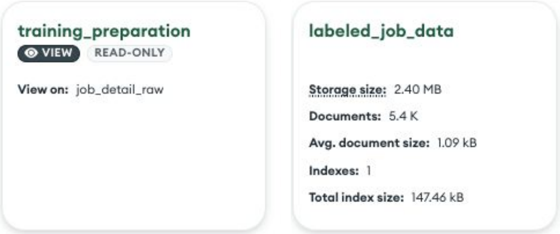
# Data Storage Setup

## Configured MongoDB to store:

- **raw data fetched from LinkedIn:**
  - Job list from searching
  - Job detail searched by job id
  - Related company
- **Cleaned data:**
  - Cleaned company data
  - Cleaned job data combined with company data

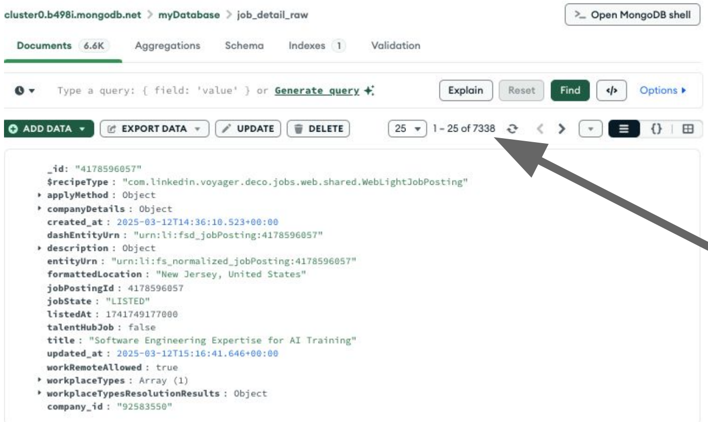



- **Unstructured data and Labeled data for training process:**



# Data fetching pipeline

- We fetched data from LinkedIn, based on the keyword “Software Engineer”


- Until Mar 16, we collected 7338 raw data

# Data cleaning and data Labeling

- Data Cleaning
  - using mongoDB pipeline, we removed some unnecessary fields, to reduce the input tokens, then we get the unstructured job data, example:

    …… Key job responsibilities\n\nDepending on your experience, interests and business needs, you will own the front-end, back-end, or full stack design and development of product features, building scale, efficiency, and differentiated customer experiences. We're looking for software engineers passionate about building software solutions end-to-end, have strong software development experience delivering at scale solutions, and systems design skills. You should have a demonstrated ability delivering within a DevOps delivery model from scoping requirements, requirement analysis, design, development, test, CI/CD…

- Data labeling
  - We used Deepseek V3 api to label the data

deepseek

- 671B MoE parameters
- 37B activated parameters
- Trained on 14.8T high-quality tokens

# Our prompt input for DeepSeek-API

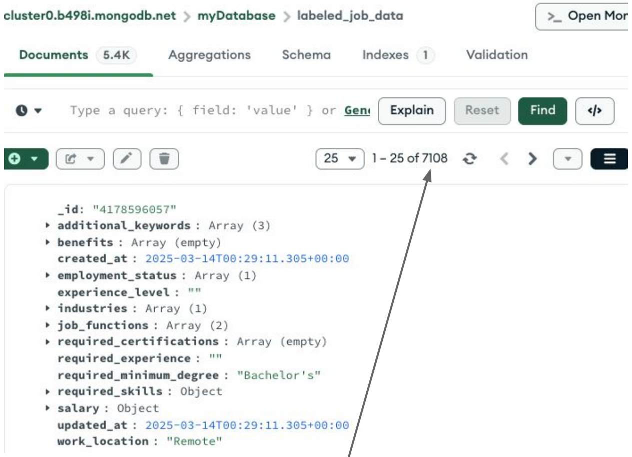
**Example:** If we are given an unstructured job post, our transformer can extract key details like:

**Input given to DeepSeek:**

Prompt: “You are a helpful AI trained to label job posting data. Label the following job posting with only json format output as per the example: *{example_labeled_data}*, no comment or explanation, just output the json format. if no information found, just leave it blank.\n raw job data:*{job_data}*”

# DeepSeek-API Output

```
example_labeled_data = """
{
    "experience_level": "",  // e.g., "Entry-level", "Mid-level", etc.
    "employment_status": [], // e.g., ["Contract", "Permanent", "Freelance", "Part-time", etc.]
    "work_location": "",    // e.g., "Remote", "Hybrid", "On-site", etc.
    "salary": {
        "min": "",          // e.g., "60000"
        "max": "",          // e.g., "80000"
        "period": "",       // e.g., "hour", "month", etc.
        "currency": ""      // e.g., "USD", etc.
    },
    "benefits": [],
    "job_functions": [],        // e.g., ["Backend", "Full Stack", etc.]
    "required_skills": {
        "programming_languages": [],  // e.g., ["Python", "Java", etc.]
        "tools": [],                  // e.g., ["Git", "Docker", etc.]
        "frameworks": [],             // e.g., ["Django", "React", etc.]
        "databases": [],              // e.g., ["MongoDB", "PostgreSQL", etc.]
        "other": []                   // e.g., ["Cloud Services", etc.]
    },
    "required_certifications": [],
    "required_minimum_degree": "",   // e.g., "Bachelor's", "Master's", "PhD"
    "required_experience": "",      // e.g., "1 year", "2 years", "3 years", etc.
    "industries": []           // e.g., ["Software Development", "Healthcare", etc.]
    "additional_keywords": []  // not mentioned above
}
"""
```




Until Mar 16, we labeled 7108 jobs

# Model Selection

We aim to implement a Transformer-based architecture using T5-XL:

- **Efficient with Large-Scale Text Data:** Transformers handle high-dimensional text embeddings efficiently through parallelized training.
- **Optimized for Text-Based Tasks:** Language modeling, text classification, and semantic similarity analysis (job role prediction, skill extraction, and job recommendation systems).
- **Self-Attention Mechanism:** Use multi-head self attention, allowing the model to assign different importance weights to different word, improving job description understanding and structured data generation.

# Model Training Pipeline

- We built a pipeline on RTX 6000 from chameleon cloud.
- Due to the RTX 6000's memory limitation, we implemented:
  - 8-bit quantization for memory efficiency
  - AMP (Automatic Mixed Precision) for faster training
  - Robust NaN/Inf detection in data and gradients
  - Gradient Clipping for stability
  - Batch Size = 1 to prevent memory overflow
  - Optimal Learning Rate: Settled on 3e-5 as the best trade-off for performance and computational
- We have reserved more powerful GPUs such as V100 and A100 in the next phase to improve training speed and handle larger batch sizes.

# GPU-Reservations Timeline

# Challenges and Solutions

- **LinkedIn API rate limits affecting job retrieval (almost 8.5 seconds per job).**
  - Solution: We used multiple computers to speed up fetching.
- **Labeling unstructured job descriptions into structured format.**
  - Solution: We implemented labeling using Deepseek api
- **DeepSeek API response latency(17 seconds per job)**
  - Solution: Implement parallel processing(5 threads reducing response under 4 seconds per job)
- **DeepSeek API Cost Management**
  - Solution: We implemented a scheduled script that runs during off-peak hours to minimize costs

# Issues that need to be resolved

- GPU limitations for training T5-XL(3B parameters).
    - **Candidate Solutions:**
        - **QLoRA (Quantized LoRA)** – Quantization + LoRA fine-tuning reduces memory usage.
        - **Gradient Checkpointing** – Saves memory by recomputing intermediate activations.
        - **DeepSpeed ZeRO Optimization** – Efficient memory distribution for multi-GPU setups like V100/A100.
- T5-XL's default 512-token limit (job descriptions often exceed 1,000 tokens)
    - Candidate Solutions:
        - **Sliding Window Approach** – Split text into overlapping chunks
        - **Text Summarization (Pre-processing Step)** – Use similar lightweight models to summarize lengthy job descriptions before passing to T5-XL.
        - **Increasing Token Limit** (Planned for Phase III) – Increase max_length to 1,024–2,048 tokens when upgrading to V100/A100 GPUs.
        - **Smart Truncation** – Focus on critical sections like "Responsibilities," "Skills," and "Qualifications."
- Data Acquisition Issues : unofficial LinkedIn API verification challenges (ChallengeException)
    - Candidate Solutions: Try other tools or modify the code inside the api.
- Training Model Generalization
    - Candidate Solutions: data augmentation, Expand data collection industries

# Blueprint for next phase

## GPU Upgrade & Memory Optimization

- Transition from **RTX 6000** to **V100 (32GB)** or **A100 (40GB/80GB)** for improved performance and expanded token limits.
- **QLoRA (Quantized LoRA)** – Quantization + LoRA fine-tuning reduces memory usage.
- **Gradient Checkpointing** – Saves memory by recomputing intermediate activations.
- **DeepSpeed ZeRO Optimization** – Efficient memory distribution for multi-GPU setups like V100/A100.

## Data Expansion and Model Improvement:

- Collect additional job postings from linkedin API, and then clean and process the new data to improve model training.
- Fixing the API's ChallengeException problems

## Model evaluation

- Precision, Recall, F1-score – For measuring entity extraction accuracy (e.g., skills, job titles).
- BLEU & ROUGE Scores – For evaluating the quality of structured text generation.
- Exact Match (EM) – To assess if the extracted structured data (e.g., JSON output) exactly matches the ground truth.
- Edit Distance (Levenshtein Distance) – To measure how closely generated outputs resemble the expected format.