LLM-Enhanced Organizational Analysis: Job Function Classification at Scale

Liu, Jeremy* Tome, Jean* Connolly, Natalia Mabry, Joshua McKeown, Ryan

Abstract

Mapping employees to standardized job taxonomy labels is critical for organizational analysis, but manual normalization of diverse job titles is time-consuming and difficult to scale. Job titles like "Front-end Developer," "Payroll," or "Data Analyst" must be classified into standardized function and sub-function categories to enable effective benchmarking and strategic planning. We present an automated approach using large language models (LLMs) with retrieval-augmented generation (RAG) to address this challenge. The system works in three steps. First, enriched taxonomy entries are embedded into a vector database. Second, the most relevant candidates are retrieved for each employee. Third, an LLM assigns labels using hierarchical context from the organization, including manager chains, peers, and direct reports. Each employee receives a ranked shortlist of plausible labels to speed human validation and correction. On a representative technology/media roster the correct label appears near the top (MRR 0.64); across synthetic rosters spanning multiple industries MRR spans 0.26–0.64.

1 Introduction

Roster mapping—assigning each employee to a standardized taxonomy label—is a critical input for organizational transformation. By mapping employees to a curated job taxonomy, analysts can benchmark organizations against industry standards, plan headcount changes, reorganize management structures, and optimize onshoring/offshoring strategies. However, this task is complicated by the diversity and inconsistency of job titles across organizations. Manual normalization of titles is time-consuming and difficult to scale, particularly when dealing with large organizations where thousands of employees must be mapped to standardized function and sub-function categories.

These challenges stem from multiple sources. Job titles are frequently untidy due to non-standard naming conventions—"software developer" in one roster might appear as

*These authors contributed equally to this work.

"SDE" in another. Titles also exhibit semantic closeness that depends on organizational context and required skill sets. For example, "Project Manager" and "Product Manager" are semantically similar and sometimes confused, even though they represent distinct roles whose responsibilities may overlap or diverge depending on the organization.

Beyond title-related issues, a roster's structure, completeness, and accuracy are not guaranteed. Internal classification schemes may suit local operational needs yet misalign with transformation objectives. Available data for each employee may be minimal—sometimes only a title and a manager. Organizational graphs can also be malformed: employees with multiple managers, orphaned executives, or broken reporting chains. These challenges are particularly acute in organizational roster mapping because the goal is to produce consistent, accurate labels that are mutually exclusive and collectively exhaustive across the entire organization for ease of interpretation and meaningful analysis. This requires adapting to diverse industries and evolving taxonomies while handling the idiosyncratic, non-standard job title naming conventions used within specific organizations—often without access to substantial domain-specific training data.

Our solution navigates these challenges to produce high-quality mappings that help produce insights on organizations. We frame roster mapping as a data classification problem and employ a modern approach of using an LLM-based classifier with retrieval-augmented generation (RAG) to efficiently handle large taxonomies [5]. The RAG architecture combines embedding vector databases for efficient candidate retrieval with LLM-based classification that leverages both organizational context (reporting relationships, hierarchical position, peer titles) and the reasoning capabilities of large language models. By retrieving the most relevant taxonomy candidates before classification, the system scales effectively to large taxonomies while maintaining classification accuracy. This approach enables rapid adaptation to new taxonomies and diverse organizational contexts without requiring extensive retraining, addressing key limitations of prior methods that focus primarily on job title text alone.

2 Related Work

Job title classification and normalization have been active areas for application of natural language processing (NLP) techniques as this standardized data is useful for talent management and recruitment activities. Earlier approaches used traditional machine learning classifiers, demonstrating improvements using word and document embedding techniques like word2vec [8]. More recent work has explored deep learning architectures to capture semantic relationships in job titles [4, 1, 3]. Notably, `JobBERT` [2] developed a BERT-based model trained on job postings to learn representations that link titles with required skills. Other systems like JAMES [6] have utilized multi-aspect graph embeddings to normalize job titles by combining semantic, syntactic, and contextual information.

3 Problem Definition

3.1 Roster Mapping Problem

We have an employee roster set X and a taxonomy set

$$T = \{(f, sf)_1, (f, sf)_2, \dots (f, sf)_m\} \tag{1}$$

where T is a collection of function f and sub-function sf tuples. A job function broadly describes what a particular role is responsible for in an organization, and a sub-function more granularly describes duties specific to a given job function. Examples include (Engineering, Front-end Developer), (Human Resources, Payroll), or (Finance, Analyst). In the typical use case, organizations are large and many employees are mapped to the same taxonomy tuple, thus $|X| \gg |T|$, but this is not guaranteed. For instance, a typical organizational taxonomy might contain more than 400 entries, so $|X| < |T|$ for small organizations.

Taxonomies can be provided by clients or constructed by analysts based on industry standards and the organization's context. In addition to the (job function, sub-function) tuples, taxonomies may also contain additional metadata like organizational division, seniority level, or typical responsibilities to aid in classification.

Each employee has a collection of characteristics (like manager, industry, compensation, etc.):

$$\text{Employees} = \{x_i \in X | x_i = (x_{i1}, x_{i2}, \dots x_{ij})\} \tag{2}$$

where j is determined by the information available about each employee[1]. At a minimum, the roster mapping solution needs employees' managers and titles, so $j \geq 2$. Ideally we obtain as much hierarchical employee information[2] as possible to differentiate between ambiguous titles. For example, x_1 and x_2 might both have the generic title of "Analyst," but if we know x_1's manager has the title "Finance, Vice President" and x_2's manager has the title "Marketing, Vice President," we can reasonably infer that x_1's assignment is (Finance, Data Analyst) and x_2's assignment is (Marketing, Business Analyst).

We aim to assign to each employee x_i a taxonomy entry $\hat{y}_i \in T$ with confidence c_i. Essentially we estimate a roster mapping function

$$g(x_i; T) = (\hat{y}_i, c_i) \tag{3}$$

which is a well-understood classification problem. In addition to providing a final taxonomy label $\hat{y}_i$ for each $x_i \in X$, $g(\cdot)$ also outputs a ranked list of other taxonomy candidates $[\hat{y}_{i1}, \hat{y}_{i2}, \dots \hat{y}_{ik}]$. This is done because the roster mapping tool is intended to

[1] Organizations are asked to provide as much information as possible about employees in tabular form. Employee characteristics are thus contained in data columns

[2] Hierarchical information includes information about the employees surrounding a given employee, i.e. manager title, skip-manager title, peer-level titles, direct reports, reporting chains, etc.

be used as a first pass algorithm before human review; the tool is not intended to create final mapping decisions. If a human reviewer disagrees with the inferred $\hat{y}_i$, they can refer to the generated list of other taxonomy candidates, which are likely to contain acceptable answers.

3.2 Evaluation Metrics

Measuring the model's performance is relatively straightforward. Given a complete dataset with ground-truth taxonomy mappings $y_i \in Y$,

$$D = \{(x_1, y_1), (x_2, y_2), \dots (x_n, y_n)\} \tag{4}$$

we can calculate accuracy [3] as

$$\text{Accuracy} = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}(y_i = \hat{y}_i) \tag{5}$$

where $\hat{y}_i = g(x_i; T)$ and mean reciprocal rank (MRR) as

$$\text{MRR} = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{\text{rank}_i} \tag{6}$$

where rank_i is the rank of y_i in the ranked list of taxonomy candidates for x_i. If the list does not contain y_i, the reciprocal rank is considered to be $\frac{1}{\infty} = 0$.

Perfect accuracy in this setting is not possible to achieve because we are asked to pick the one best option out of many highly relevant candidates to ease downstream decision-making. An expanded measure of accuracy, which we refer to as HitRate@k, considers whether y_i is found in the top-k candidates $[\hat{y}_{i1}, \hat{y}_{i2}, \dots \hat{y}_{ik}]$:

$$\text{HitRate@}k = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}(y_i \in [\hat{y}_{i1}, \hat{y}_{i2}, \dots \hat{y}_{ik}]) \tag{7}$$

To evaluate the position of the correct answer in the ranked list of candidates, we use MRR as defined in Equation 6. MRR captures how highly the correct answer is ranked among the candidates; a higher MRR indicates that the correct answer tends to appear closer to the top of the list.[4]

[3] Unless otherwise specified, Accuracy refers to exact-match accuracy on the final (function, sub-function) label; component-wise (function-only or sub-function-only) results are distinguished explicitly where reported.

[4] We use MRR and HitRate@k rather than Precision@k or Recall@k because we enforce exactly one correct taxonomy entry y_i per employee x_i to maintain label mutual exclusivity and avoid ambiguity. With a single relevant item, Precision@k and Recall@k necessarily yield low scores unless the correct answer appears within the top-k positions, providing little insight beyond what HitRate@k and MRR

4 Motivation and Design Considerations

Prior work has advanced job title normalization, but roster mapping at enterprise scale presents distinct challenges that existing approaches do not fully solve:

- **Messy, organization-specific titles:** Internal titles are abbreviated, stylized, or overloaded (e.g., "SDE," "Contractor," "Chief of Staff" used across functions). Methods trained on public job postings struggle to consistently map these to mutually exclusive, collectively exhaustive taxonomy entries required for transformation analysis.
- **Hierarchy-dependent disambiguation:** Human analysts rely heavily on neighborhood context (manager, skip-manager, peer titles, direct reports) to distinguish semantically close roles (e.g., "Project Manager" vs. "Product Manager"). Most prior classifiers ingest only isolated title text.
- **Taxonomy scale and drift:** Real taxonomies may exceed hundreds of (function, sub-function) pairs and evolve across engagements and industries. Retraining large supervised models for each variant is costly and slow.
- **Sparse, privacy-restricted data:** Access to historical rosters is limited; models requiring large labeled corpora are impractical in consulting contexts with strict data use constraints.

These constraints motivate our design choices for how to approach the roster mapping problem:

- **Context Enrichment:** We synthesize hierarchical and relational metadata for each employee, then summarize it with LLM-generated keywords/descriptions prior to embedding, enabling role disambiguation beyond surface title strings.
- **Deliberately Simple Embeddings:** We adopt robust general-purpose embeddings and invest effort in structural/context enrichment rather than domain-adapted embedding models, given limited training data.
- **Hybrid Retrieval-Augmented Architecture:** Vector + BM25 retrieval narrows candidate space before LLM reasoning, improving accuracy and scalability when $|T|$ is large without requiring domain-specific model retraining.

already capture. When multiple taxonomy entries are semantically equivalent (e.g., (Engineering, Software Developer) and (Engineering, Software Engineer)), we consolidate them into a single canonical entry (e.g., Engineering, Software Engineer) during taxonomy preprocessing to prevent metric inflation from trivial synonym matches.

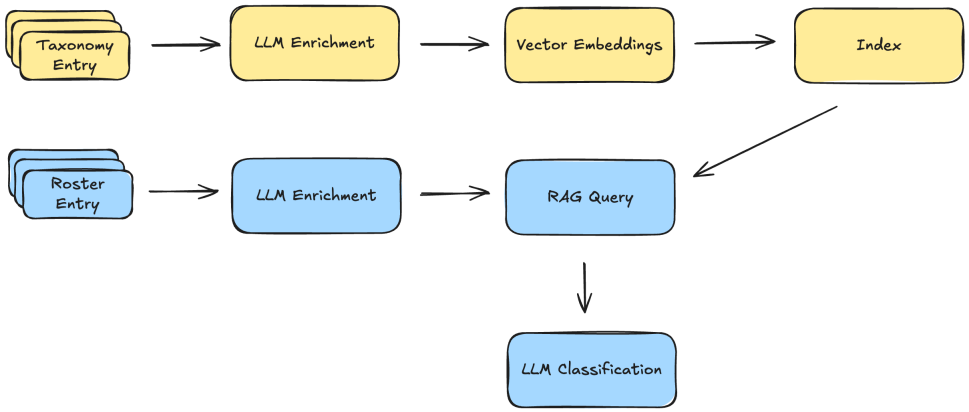


Figure 1: Roster mapping workflow showing the end-to-end process from raw inputs to final assignments. First, taxonomy entries are enriched with contextual descriptions and embedded into a searchable vector database. Next, employee records are enriched with hierarchical context (manager, peers, reports) and embedded for similarity search against the taxonomy database. The top-k most relevant taxonomy candidates are then combined with the enriched employee data in a structured prompt sent to an LLM classifier, which produces the final job function and sub-function assignments along with confidence scores.

- **Zero-Shot Prompt-Driven Classification:** LLM classification with carefully constructed prompts allows rapid adaptation to new industries and taxonomies with no fine-tuning cycle.
- **Synthetic Data Generation:** Custom organization and taxonomy synthesis provides diverse evaluation scenarios while respecting privacy constraints on real organizational data.

In summary, the roster mapping problem demands a system that fuses hierarchical context, scalable retrieval, and taxonomy curation rather than relying solely on title text embeddings or supervised retraining. The following sections formalize the problem and describe the architecture and its measured performance.

5 Algorithm and Workflow

We outline the roster mapping process in Algorithm 1 to provide context for more detailed descriptions of system components found in other sections. Section 6 describes functions related to embeddings, enrichment, hybrid search, and LLM classification.

Both employees and taxonomies go through an enrichment and embedding process before being combined together in one prompt sent to the LLM classifier.

A visual representation of Algorithm 1's workflow is found in Figure 1.

Algorithm 1 Roster Mapping Algorithm

$X \leftarrow$ employees
$T \leftarrow$ taxonomy
if evaluating on synthetic data **then**
    $Y \leftarrow$ ground truth (sub-)function assignments
end if

$\hat{Y} \leftarrow \{\}$ ▷ Predicted (sub-)function assignments
$T_{enriched} \leftarrow$ **EnrichTaxonomy**(T)
$T_{emb} \leftarrow$ **EmbedTaxonomy**$(T_{enriched})$
for all $x \in X$ **do**
    $x^{+} \leftarrow$ **EnrichEmployee**(x)
    $emb \leftarrow$ **EmbedEmployee**(x^{+})
    $candidates \leftarrow$ **HybridSearch**(emb, T_{emb})
    $\hat{\mathbf{y}}_i, \mathbf{c}_i \leftarrow$ **LLMClassify**$(x^{+}, candidates)$ ▷ Prediction and confidence vectors
    $\hat{Y}[x] \leftarrow ([\hat{\mathbf{y}}_{i1}, \hat{\mathbf{y}}_{i2}, \ldots, \hat{\mathbf{y}}_{ik}], [\mathbf{c}_{i1}, \mathbf{c}_{i2}, \ldots, \mathbf{c}_{ik}])$
end for

if Evaluating on synthetic data **then**
    MRR, Accuracy, HitRate@$k \leftarrow$ **CalculateMetrics**$(Y, \hat{Y})$
end if

return $\hat{Y}$

6 Model

The roster mapping model has a searchable vector database and a LLM classifier as its two main components. Recall that the model takes as input employees X and a taxonomy T to compute $\hat{y}_i = g(x_i; T)$, the job function and sub-function assignments for employees. The main engine for inferring $\hat{y}_i$ is the LLM classifier, but it needs to be supplied with a prompt containing adequate context and information. The vector database plays that role by injecting a prompt with employee information and taxonomy candidates before that prompt is passed to the LLM for a final classification decision.

In the course of iteratively evaluating and improving the model, we used synthetic organizations and taxonomies as discussed in Sections 7.1 and 7.2. Implementation details are found in Appendix A.

6.1 Hybrid Retrieval

The goal of the retrieval stage is to pair an "enriched" employee e_i^+ (which we discuss shortly) with several promising taxonomy candidates $[t_{i1}, t_{i2}, \ldots t_{ik}]$ that might be good job function and job sub-function assignments; this pairing $(e_i^+, [t_{i1}, t_{i2}, \ldots t_{ik}])$ is then passed to the LLM classifier for a final decision $\hat{y}_i$.

The searchable vector database is composed of an embedding for each $t_i \in T$. Taxonomy entries are also enriched by asking the LLM to produce a textual description of what duties a particular job function and sub-function tuple might have. After this taxonomy enrichment, we calculate an embedding vector for the job function and sub-function entry. Each embedding vector captures information about a particular combination of job function and sub-function that exists in a given taxonomy, and these vectors are generated by OpenAI's `text-embedding-3-small` model.

This enrichment step parallels evidence from adjacent taxonomy mapping research (e.g., TELEClass [7]), which shows that augmenting terse labels with LLM-generated duty narratives improves downstream retrieval and classification quality. We incorporate enrichment in production, but stop short of performing more aggressive, automated taxonomy restructuring, given the potential risks and complexities involved.

A retrieval query against the database seeks the taxonomy entries most similar to enriched employee e_i^+. While e_i contains basic employee information, e_i^+ includes information external to e_i such as manager job title, skip-manager (manager's manger) job title, direct reports, peers, etc. The model then takes the additional step of using `gpt-4o` to read this context and generate a list of summary keywords along with a general description. From there, `text-embedding-3-small` generates an embedding of the concatenated context, keywords, and description before using that embedding to search against the taxonomy entries in the database to find the top-k[5] closest (function, sub-function) tuples.

[5] We return the top $k = 10$ taxonomy entries in our model

The keyword description generation uses `gpt-4o` and thus requires an accompanying prompt. The prompt is constructed from a basic recipe:

(a) **Personify the system** as a Human Resources expert that specializes in identifying job functions and sub-functions based on textual descriptions.

(b) **Inject employee information:** Add employee e_i's basic information (i.e. title, salary, number of direct reports, compensations, etc.)

(c) **Enrich with context:** Include e_i's secondary inferred information (i.e. manager's title, skip-manager's title, peer information, etc.)

(d) **Request structured output:** Given all prior information, request a k-keyword summary[6] and general description of e_i.

With the prompt in hand, `gpt-4o` can be invoked to produce the desired keyword summary and general description composing e_i^+; we can likewise invoke `text-embedding-3-small` to create an embedding for e_i^+ and search the database for $[t_{i1}, t_{i2}, \ldots t_{ik}]$.

To address the challenges posed by non-standard naming conventions, semantic ambiguity, and varying taxonomy sizes, our retrieval system employs a hybrid search approach that combines vector-based semantic similarity with keyword-based lexical matching. Rather than relying solely on embedding similarity, we augment the retrieval process with BM25 full-text search, which applies statistical term-frequency ranking to the textual descriptions of taxonomy entries. The hybrid search computes a weighted combination of vector similarity scores and BM25 relevance scores, with weights $w_v = 0.8$ for vector search and $w_{bm25} = 0.2$ for BM25. This weighting prioritizes semantic understanding while ensuring that exact keyword matches receive appropriate credit, particularly important when job titles use standardized terminology or industry-specific jargon.

The hybrid approach provides several key benefits for the roster mapping problem. First, it handles title variation gracefully: vector search recognizes that "SDE" and "Software Developer" are semantically equivalent, while BM25 rewards exact matches when standardized titles appear. Second, it disambiguates identical titles in different contexts—for instance, distinguishing between "Contractor" in IT versus Construction by leveraging both the semantic context from enriched employee data and keyword signals from job function fields. Third, the combination proves robust across industries: while vector embeddings may suffer from training data bias toward common industries like *technology*, BM25 provides a statistical safety net that works equally well across specialized domains. The quality of the top-k candidates retrieved by hybrid search is critical to overall system performance, as these candidates form the input to the LLM classifier in the subsequent stage. If the ground-truth taxonomy entry does not appear in the top-k results, the LLM has no opportunity to select the correct answer. Our experimental results demonstrate

[6] We use 5 keywords in our model

that the hybrid approach successfully places relevant candidates within the top-10 results, as evidenced by strong HitRate@5 and MRR metrics across diverse datasets.

6.2 LLM Classifier

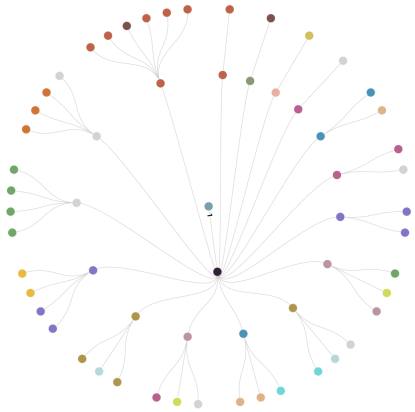
Given the complex knowledge and data required to properly train a traditional ML model to estimate $g(\cdot)$, we instead opt to use OpenAI's `gpt-4o` and careful prompting to accomplish the same task.

As mentioned in Section 6.1, the LLM receives a pair $(e_i^+, [t_{i1}, t_{i2}, \ldots t_{ik}])$ as input. Properly invoking `gpt-4o` requires us to construct a prompt p_i, which follows the same basic recipe steps (a), (b), and (c) shown in Section 6.1. But instead of adding a request for keywords as in (d), we inject $[t_{i1}, t_{i2}, \ldots t_{ik}]$ and ask the LLM which t_{ij} among the list is the most appropriate assignment for e_i^+.

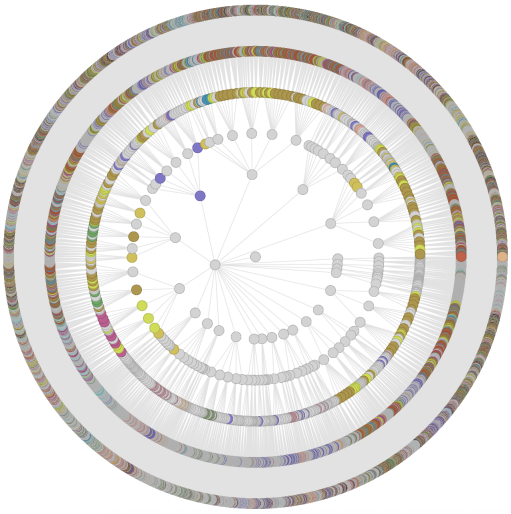
7 Data

As indicated in Equations 4, 5, and 6, ground-truth taxonomy mappings $y_i \in Y$ are required to properly evaluate mapping model performance. Historical data from organizational analyses would ideally provide complete employee rosters with assigned mappings. However, organizational data is generally unavailable for research purposes due to privacy concerns, regulatory restrictions, and proprietary business considerations that limit data sharing. Additionally, available mapping exercises typically involve larger organizations in specific industries such as automotive manufacturing, creating generalization challenges when developing models that must perform across diverse organizational contexts and industry sectors. While these constraints are not insurmountable, addressing them requires long-term data collection efforts. To address these data limitations, we developed a synthetic data generation approach that creates realistic organizational structures and taxonomies. While off-the-shelf libraries such as Synthetic Data Vault (`sdv`) and YData Synthetic (`ydata-synthetic`) exist for synthetic data generation, they proved inadequate for our specific requirements. These libraries either generated unrealistic organizational hierarchies that failed to capture authentic reporting relationships or required substantial historical organizational data as training seeds—data that was unavailable due to the privacy constraints mentioned above.

We therefore built a custom synthetic data generation system specifically designed for the roster mapping problem. This system produces complete datasets containing employee rosters X, taxonomies T, and ground-truth mappings Y that reflect the diversity and complexity of real organizational contexts across different industries and company sizes. Our synthetic data generation proceeds in two coordinated stages: organization synthesis (Section 7.1) and taxonomy synthesis (Section 7.2). The organization synthesis stage creates realistic hierarchical structures with appropriate reporting relationships, while the



(a) A synthetic technology startup with approximately 60 employees. The organization maintains a flat hierarchy with the deepest reporting level only two steps from the CEO (center node). This structure reflects typical early-stage startup characteristics.



(b) A synthetic maritime logistics company. There are ∼17000 employees and the organization has many layers of management.

Figure 2: Synthetic organizations generated using LLMs.

taxonomy synthesis stage generates industry-specific job function and sub-function categories. For each simulated company, an LLM derives a plausible set of job function and sub-function entries conditioned on the organization's industry, size, and structural characteristics. By jointly generating the roster and taxonomy rather than treating them as independent components, we ensure that evaluation datasets span the diverse organizational contexts that the roster mapping system encounters in practice while maintaining internal consistency between organizational structure and job categorization schemes.

7.1 Synthetic Organizations

In our synthetic data generator, an organization is a directed tree $G = (X, E)$ where $e_i = (x_a, x_b) \in E$ indicates employee x_a manages employee x_b. We also need accompanying function/sub-function pairs $y_i \in Y$ for employees $x_i \in X$ for our mapping needs[7].

Our organization generator is a simple recursive function. We start from a root (CEO or a super-root if there are co-CEOs) and create children based on parameters θ, recursing on each child. Each generated node receives attributes $x_i = (x_{i1}, x_{i2}, \ldots x_{ij})$ sampled from distributions governed by θ. In practice we use lightweight parametric choices (e.g.,

[7]Recall that $y_i = (f_i, sf_i)$

truncated normal for non-negative counts or multinomial for categorical variables) rather than committing to any single family.

Termination of the generation process is determined by θ: if properly parameterized, eventually deep nodes will generate no additional children. Each type of employee has some $\theta_i = \{\theta_{i1}, \ldots \theta_{ik}\}$ that parameterizes a distribution over how many children the employee can have; in our case, the model uses a truncated normal distribution (truncated at zero to ensure non-negative employee counts), so $\theta_i = \{\mu_i, \sigma_i\}$. Deep nodes will have $\mu_i \approx 0$ and $\sigma_i \approx 0$, so they will rarely generate new direct reports and the recursion will end.

We determined good θ to create realistic organizations through two methods. The first was by examining existing organizational data to calculate $\theta = \{\theta_1, \theta_2, \ldots \theta_k\}$, and the second was using a large language model to create θ.

Calculating θ from existing organizations involves gathering summary statistics and fitting simple distributions keyed on function, sub-function, and depth. For example, θ might specify that a Software Engineer / Architect three levels from the CEO has a mean of 5 direct reports, a standard deviation of 2, and a multinomial over direct report types with probabilities $[0.7, 0.2, 0.1]$ for [Software Engineer / AI Engineer, Software Engineer / DevOps, Administration / Assistant], conditioned on the manager's title and organizational depth. This frequentist construction just counts occurrences under different contexts; it is efficient but depends on access to sufficiently diverse real rosters.

When comprehensive organizational data are not readily available, an alternate method to obtain θ is through systematic desk research, including archival research and web-based data collection. However, this method has little chance of working in practice because such parameters and statistics are generally unavailable or proprietary. These conditions lead us to consider using LLMs to parameterize θ instead. LLMs are useful for parameterizing θ because they are trained using all available text. The text corpus contains some knowledge on practically all topics including information on company structures. Because our own research is limited by a lack of available data, we can exploit an LLM's general knowledge to fill in our missing parameters. Prompts posed to `gpt-4o` are quite simple: as a base, we provide a company's description, its industry, and any other relevant context. Then given an archetypal employee ae (a function, sub-function, and depth), we ask `gpt-4o` create various $\theta_i \in \theta$ that describes the characteristics of ae's children. As mentioned before, these characteristics include the mean number of potential children, the standard deviation of that number, the functions and sub-functions of the children, and how those (sub-)functions are distributed. This children information results in new archetypal employees, and we recursively generate additional $\theta_i \in \theta$ from these archetypal children. Once fully constructed, θ can be sampled to produce synthetic organizations like those pictured in Figures 2a and 2b.

To simulate real-world noisiness, we include an adjustable probability[8] of reporting chains being incorrect. That is, instead of producing an employee's direct reports according

[8] 5% chance in our generation process.

to the appropriate θ_i, we replace a direct report with a randomly generated employee drawn from any distribution. This results in situations such as a (Legal, General Counsel) employee reporting to a (IT, Cyber Security) employee. Such noisiness can reflect situations resulting from companies merging with one another or poor organizational practices.

Another procedure to add noise is to inject ambiguous job titles into the organization. To illustrate, suppose $e_1 \to$ (IT, Software Developer) and $e_2 \to$ (Construction, Electrician). Both e_1 and e_2 could conceivably have the same job title "Contractor" even though their job functions and sub-functions are different. To inject ambiguity, we select pairs of employees and ask `gpt-4o` to create a job title that the pair can share; we then replace both their job titles with the suggested title. Ambiguity injection is a post-processing step that occurs after $G = (X, E)$ has already been generated.

We created several datasets for evaluation purposes.

1. `Historic`: A Technology and Media organization parameterized by roster data from a past organizational analysis.
2. `Historic, Ambiguous Titles`: The same as above, but with some job titles made ambiguous.
3. `Construction`: A Construction company parameterized by LLM.
4. `Amusement Park, Hierarchy`: An amusement park parameterized by LLM that includes manager, skip-manager, and direct reports data.
5. `Amusement Park, Deduplicated`: An amusement park parameterized by LLM with deduplicated job titles.

7.2 Synthetic Taxonomies

Synthetic organizations generated with an LLM yield an initial taxonomy T of (function, sub-function) tuples assigned to employees that serves as the ground-truth reference, but it often contains near-duplicate paraphrased variants (e.g., (Human Resources, Payroll) vs. (HR, Employee Payroll)). Left unmerged, these variants inflate the search space and can artificially raise apparent accuracy by presenting multiple semantically identical "correct" answers. We therefore apply a systematic two-stage consolidation pipeline: (1) hierarchical clustering to reduce the naive $O(|T|^2)$ pairwise comparison space to groups of likely duplicates, and (2) within-cluster normalization and merging that selects a canonical representative for each semantically equivalent set. The cleaned taxonomy is then passed to the mapping pipeline.

Stage 1: Hierarchical Clustering. Each taxonomy entry is converted to an enriched text string combining industry, function, sub-function (and optional description). We embed all strings with `sentence-transformers/all-MiniLM-L6-v2` and compute cosine distances. Using agglomerative (Ward) clustering on the embedding vectors, we sweep a

cut threshold from 0.3 to 0.9 and select the smallest threshold that (a) keeps median cluster size ¡ 5 and (b) materially reduces the total candidate pairs. The resulting clusters serve as compact candidate duplicate groups for Stage 2.

Stage 2: Within-Cluster Merging. We calculate pairwise cosine similarities within each cluster. Entry pairs exceeding a configurable similarity threshold τ (typically ≥ 0.8) are merged, retaining the longer, more descriptive entry as the canonical representative. We track all merge relationships in a mapping $M : T_{llm} \to T_{cleaned}$ and resolve transitive merges to ensure consistency—if $t_a \to t_b$ and $t_b \to t_c$, then $M(t_a) = t_c$. The roster is then updated by applying M to all employee assignments: for each employee x_i with original assignment $y_i \in T_{llm}$, we replace y_i with $M(y_i) \in T_{cleaned}$. Historical mappings are preserved in auxiliary columns (y_i^{old}, y_i^{new}) to maintain provenance.

The impact of deduplication is substantial. In one synthetic amusement park organization, the initial LLM-generated taxonomy contained $|T_{llm}| = 461$ unique entries. After deduplication with $\tau = 0.8$, this reduced to $|T_{cleaned}| = 115$ distinct entries—a 75% reduction. Beyond computational efficiency, deduplication creates a more realistic and challenging classification task: without consolidation, the model could achieve a "correct" classification by selecting any of several semantically identical options, making the task artificially easier than real-world scenarios. The cleaned taxonomy better reflects real-world organizational structures where job titles are standardized and controlled, improving both candidate retrieval efficiency and the validity of our evaluation metrics.

8 Results

The roster mapping model was evaluated on Accuracy (Equation 5), MRR (Equation 6), and HitRate@5 (Equation 7). Evaluation took place across several datasets of varying complexity to give insight on how performance differed across industry and size. Industry type may significantly impact model performance. Organizations in Technology and Media (the `Historical` datasets) achieved much higher Accuracy and MRR scores, while Entertainment and Hospitality organizations (the `Amusement Park` datasets) proved more challenging to map accurately. This performance gap likely stems from training data exposure bias: LLMs may encounter substantially more text about technology companies—software development careers, organizational structures, job functions—than about specialized industries like hospitality or entertainment. The abundance of news articles, blog posts, and professional discussions about tech companies could provide richer contextual knowledge for understanding their organizational patterns, whereas niche industries like amusement parks may receive limited coverage. Consequently, the model appears to demonstrate stronger performance when mapping familiar organizational contexts but may struggle with specialized domains where training data could be sparse.

We also observed that incorporating hierarchical information substantially improves model performance. This finding aligns with our intuition based on observing human

Dataset	Fn Hit@5	Fn Acc	Sub-fn Hit@5	Sub-fn Acc	MRR
Historical	0.95	0.86	0.78	0.56	0.64
Historical (Ambig Titles)	0.95	0.60	0.86	0.26	0.48
Construction	0.79	0.51	0.77	0.46	0.58
Amusement Park (Hierarchy)	0.52	0.32	0.49	0.30	0.37
Amusement Park (Dedup)	0.50	0.21	0.41	0.21	0.26

Table 1: Summarized roster mapping results on different datasets. Each value represents the average performance across multiple runs. Detailed results for individual runs on each dataset are provided in Appendix C.

reviewers, who frequently rely on an employee's organizational context—manager, skip-manager, peers, and direct reports—when performing manual mapping tasks. These contextual factors often prove decisive when an employee's basic information (title, compensation, etc.) alone is insufficient for confident classification. To validate this hypothesis, we conducted a series of ablation experiments where we systematically varied which hierarchical features were included during the employee enrichment process. As expected, we observed consistent improvements in HitRate@5 performance as more contextual information was incorporated (see Table 2).

Experiment Configuration	Hit@5	Relative Change
No Hierarchy Info	0.633	–
Direct Report Info	0.593	-6.3%
Skip Manager Info	0.687	+8.5%
Manager Info	0.720	+13.7%
Manager + Skip Manager Info	0.713	+12.6%
All Information	0.773	+22.1%

Table 2: Performance improvement with hierarchical information

9 Conclusion

We have presented an LLM-enhanced roster mapping system that addresses the practical challenges of enterprise-scale organizational analysis through retrieval-augmented generation. Unlike traditional approaches that rely on job title embeddings with nearest-neighbor lookup or supervised classifiers requiring extensive labeled data, our architecture combines hierarchical context enrichment, hybrid search, and zero-shot LLM classification to handle messy organization-specific titles, evolving taxonomies across industries, and the sparse, privacy-restricted data typical of consulting engagements. By incorporating manager chains, peer information, and direct reports alongside general-purpose embeddings,

the system disambiguates semantically close roles without domain-specific model retraining or costly fine-tuning cycles.

Our synthetic data generation pipeline enabled rigorous evaluation across diverse industries while respecting privacy constraints, demonstrating that thoughtful integration of organizational context with retrieval and LLM reasoning produces practical tools for accelerating human-driven roster mapping. Future work includes evaluation on real organizational data, refinement of hierarchical context encoding, and active learning from human corrections.

10 Acknowledgements

Ken Florentino, Miro Samawil, Nicolas Aimetti, John Nascimento, and Firas Zebib for building the infrastructure and architecture to host and deploy the roster mapping model.

11 Disclaimer

This work represents the independent research and opinions of the authors and does not necessarily reflect the views, positions, or policies of AlixPartners, LLP. The methods, results, and conclusions presented herein are provided for academic and informational purposes only. AlixPartners, LLP makes no representations or warranties regarding the accuracy, completeness, or suitability of this work for any particular purpose and accepts no liability for any errors, omissions, or consequences arising from the use of this information. Any application of the techniques or findings described in this paper should be undertaken with appropriate professional guidance and due diligence.

References

[1] Maiia Bocharova, Eugene Malakhov, and Vitaliy Mezhuyev. VacancySBERT: the approach for representation of titles and skills for semantic similarity search in the recruitment domain. *Applied Aspects of Information Technology*, 6(1):52–59, April 2023. arXiv:2307.16638 [cs].

[2] Jens-Joris Decorte, Jeroen Van Hautte, Thomas Demeester, and Chris Develder. Jobbert: Understanding job titles through skills, 2021. Develops JobBERT, a BERT-based model for understanding job titles through associated skills. Uses job postings data to learn representations that capture semantic relationships between titles and required skills. Important baseline for skill-aware job title classification.

[3] Hao Liu and Yong Ge. Job and Employee Embeddings: A Joint Deep Learning Approach. *IEEE Transactions on Knowledge and Data Engineering*, 35(7):7056–7067, July 2023. IEEE TKDE 2023. Joint learning approach for job and employee embeddings using three granularity levels: content (transformer), context (shallow NN), and sequence (RNN encoder-decoder). Models career paths and job similarities. Comprehensive approach combining multiple neural architectures for HR analytics.

[4] Junhua Liu, Yung Chuen Ng, Zitong Gui, Trisha Singhal, Lucienne T. M. Blessing, Kristin L. Wood, and Kwan Hui Lim. Title2Vec: a contextual job title embedding for occupational named entity recognition and other applications. *Journal of Big Data*, 9(1):99, September 2022. Journal of Big Data 2022. Introduces Title2Vec embeddings using bidirectional language models for occupational NER. Provides IPOD dataset

with 475,073 job titles from 192,295 profiles. CRF and LSTM-CRF models outperform human baselines. Dataset and embeddings publicly available - valuable resource for job title analysis.

[5] Sowmya Vajjala and Shwetali Shimangaud. Text Classification in the LLM Era – Where do we stand?, February 2025. arXiv:2502.11830 [cs].

[6] Michiharu Yamashita, Jia Tracy Shen, Thanh Tran, Hamoon Ekhtiari, and Dongwon Lee. James: Normalizing job titles with multi-aspect graph embeddings and reasoning. In *2023 IEEE 10th International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–10, 2023. DSAA 2023. Presents JAMES system for job title normalization using multi-aspect graph embeddings and reasoning. Combines semantic, syntactic, and contextual information through graph-based approaches. Key contribution to job title standardization and entity mapping in HR analytics.

[7] Yunyi Zhang, Ruozhen Yang, Xueqiang Xu, Rui Li, Jinfeng Xiao, Jiaming Shen, and Jiawei Han. Teleclass: Taxonomy enrichment and llm-enhanced hierarchical text classification with minimal supervision, 2025. Accepted to WWW 2025. Proposes hierarchical text classification with minimal supervision using only class names. Combines LLM knowledge with task-specific features from unlabeled corpora. Achieves comparable performance to zero-shot LLM prompting with significantly less inference cost. Highly relevant for job title taxonomy enrichment and classification.

[8] Yun Zhu, Faizan Javed, and Ozgur Ozturk. Document Embedding Strategies for Job Title Classification. 2017. Explores document embedding strategies to replace BOW models for job title classification. Proposes customized embedding approach for multi-aspect job ads including descriptions, requirements, and company details. Shows improved accuracy over traditional BOW methods for job classification systems.

A Implementation

We implemented the roster mapping model using Microsoft Azure's Durable Functions framework to orchestrate parallel processing through a fan-in/fan-out paradigm that applies to both taxonomy preparation and employee classification. During taxonomy processing, the orchestrator fanned out individual taxonomy entries across multiple workers for concurrent enrichment and embedding generation. These workers invoked Azure OpenAI's `gpt-4o` to generate semantic descriptions and `text-embedding-3-small` to produce vector embeddings. The orchestrator then fanned in the enriched taxonomy entries and persisted them to Azure Cosmos DB, which serves as both a document store and vector database.

For employee processing, the orchestrator divided the roster into batches and fanned out classification tasks across parallel workers. Each worker enriched employee records with hierarchical context, generated embeddings, executed hybrid search queries combining

vector similarity and BM25 keyword matching against the Cosmos DB taxonomy index, and invoked the LLM classifier with the top-k candidate taxonomies. The orchestrator fanned in all classification results to produce the complete roster mapping. This serverless implementation enabled horizontal scaling to handle organizations of arbitrary size while maintaining fault tolerance through automatic retries and distributed state management, ensuring robust processing even when individual operations failed.

B HR Embedding Model

We hypothesized that using an embedding model specifically trained on HR data (job listings, resumes, descriptions, etc.) would outperform OpenAI's `text-embedding-3-small` model, which is trained on general text data. To test this hypothesis, we modified the roster mapping model to use the `JobBERT-v2` model instead of `text-embedding-3-small` on the `Construction` dataset.

Although `JobBERT-v2` is a much smaller model than `text-embedding-3-small`, we believed its HR-specific training would give it an edge in capturing the semantics of job functions and sub-functions. However, experimental results showed that performance differences were minimal and not statistically significant across all metrics.

C Detailed Results

This section provides granular experimental results for each dataset across multiple runs, showing the consistency and variability of the model's performance.

Run	Function HitRate@5	Function Accuracy	Sub-function HitRate@5	Sub-function Accuracy	MRR
1	0.95	0.87	0.78	0.56	0.64
2	0.95	0.87	0.78	0.59	0.66
3	0.96	0.87	0.79	0.55	0.65
4	0.95	0.85	0.77	0.55	0.63
5	0.95	0.84	0.79	0.56	0.64

Table 3: Results for the `Historical` dataset

ß

Run	Function HitRate@5	Function Accuracy	Sub-function HitRate@5	Sub-function Accuracy	MRR
1	0.94	0.43	0.81	0.08	0.34
2	0.96	0.84	0.89	0.52	0.65
3	0.94	0.60	0.85	0.25	0.48
4	0.96	0.50	0.89	0.11	0.38
5	0.96	0.64	0.87	0.32	0.54

Table 4: Results for the `Historical, Ambiguous Titles` dataset

Run	Function HitRate@5	Function Accuracy	Sub-function HitRate@5	Sub-function Accuracy	MRR
1	0.82	0.51	0.79	0.47	0.59
2	0.76	0.53	0.77	0.49	0.58
3	0.75	0.50	0.72	0.45	0.56
4	0.79	0.52	0.77	0.46	0.58
5	0.81	0.48	0.78	0.45	0.57

Table 5: Results for the `Construction` dataset

Run	Function HitRate@5	Function Accuracy	Sub-function HitRate@5	Sub-function Accuracy	MRR
1	0.50	0.32	0.48	0.30	0.38
2	0.57	0.31	0.55	0.28	0.39
3	0.51	0.34	0.48	0.32	0.38
4	0.46	0.30	0.42	0.25	0.32
5	0.55	0.34	0.54	0.32	0.41

Table 6: Results for the `Amusement Park, Hierarchical` dataset

Run	Function HitRate@5	Function Accuracy	Sub-function HitRate@5	ßSub-function Accuracy	MRR
1	0.57	0.22	0.47	0.17	0.27
2	0.51	0.20	0.45	0.18	0.25
3	0.58	0.21	0.45	0.17	0.26
4	0.58	0.20	0.50	0.18	0.28
5	0.50	0.21	0.41	0.21	0.26

Table 7: Results for the `Amusement Park, Deduplicated` dataset