Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Bachelor Thesis

Novel Approach for a Job Matching System: Large Language Model-Based Requirements Extraction and Token-Embedding-Driven Similarity Search

Seyed Taha Amirhosseini

seyed.taha.amirhosseini@studium.uni-hamburg.de
Course of Studies: Information Systems
Matriculation Number: 7364370

Primary Referee: Dr. Lothar Hotz
Secondary Referee: Marten Borchers

Submission Date: 30.01.2026

Contents

1 Introduction

In the digital age, online job platforms provide users with access to a wide range of employment opportunities. Yet, despite this abundance, finding positions that genuinely match a user's skills and preferences remains a challenge [MLK+24]. A central cause of this mismatch lies in the fundamental search and filtering mechanisms employed by many platforms, which rely heavily on job titles as the primary means of retrieval [MLK+24]. Title-based search suffers from a well-documented limitation: job titles alone rarely capture the full semantic content of a role. As a result, users frequently encounter postings that are nominally relevant but misaligned with the actual tasks and qualifications required [KAKAH25, WKS25]. In addition, ranking systems on some platforms are influenced by mechanisms such as bid dominance, where less relevant jobs appear prominently simply because employers submit higher bids [PSM+25]. This not only distorts search results but also reduces the overall quality and trustworthiness of the user experience.

These issues demonstrate that existing job-matching systems are often insufficiently effective and motivate the need for more robust, semantically informed approaches—an issue at the core of this thesis.

1.1 Motivation

As industries evolve, job roles and the competencies they require have become increasingly specialized. Musazade et al. observe that "data professionals have started to be required to possess a narrower competence and specialization in particular common, unique and extensive tools and technologies" [Mus23]. This heightened specialization magnifies the weaknesses of title-based search systems: even slight variations in wording can hide otherwise highly similar job roles from search results. Kabir et al. illustrate this issue with queries such as "Software Engineer Java" and "Java backend developer", which differ lexically but are semantically aligned due to overlapping skill requirements [KAKAH25]. Similarly, Womark et al. highlight cases where generic queries like "Human Resources" lead to irrelevant results, such as job descriptions containing phrases like "speak to a Human Resources representative", without actually referring to HR roles [WKS25]. To address these limitations, researchers increasingly advocate for the use of contextual information. Kabir et al., for instance, recommend incorporating job-related skills, industries, and other content-rich signals to improve similarity prediction models [KAKAH25]. Building upon this insight, the central idea of this thesis is to analyze

full job descriptions and extract structured information to enable semantically grounded matching.

This work builds on YourJobFinder [Ami25], an existing aggregation tool that scrapes unstructured job descriptions from multiple online sources. While useful for data collection, its reliance on title-based retrieval results in low precision and many irrelevant postings. To address this, the core contribution of this thesis is a filtering and matching system that processes these unstructured descriptions and identifies postings that genuinely align with a user's profile.

The system developed in this thesis uses an open-source Large Language Model (LLM) to extract key attributes—such as skills, experience, and qualifications—from job descriptions. These extracted requirements are then transformed into vector representations and compared with similarly vectorized user profiles. By relying solely on open-source models and tools, the approach aims to provide a transparent, and cost-effective solution.

1.2 Research Questions

To guide this research and address the stated objectives, the thesis investigates the following questions:

- RQ1: How effectively can open-source LLMs extract structured job requirements (skills, experience, qualifications) from unstructured job descriptions?
- RQ2: How does LLM-based requirement extraction and token-embedding-driven similarity search compare to conventional job search engines with respect to the relevance and accuracy of retrieved job postings?

2 State of the Art

The limitations of current job-matching systems, as outlined in the introduction, have led researchers to explore more sophisticated approaches capable of capturing the semantic relationships between job descriptions, requirements, and user profiles. Over the past years, the field has evolved from simple keyword-based methods toward advanced semantic, transformer-based, and large-scale industrial systems. This chapter provides an overview of these developments, focusing on those techniques that inform the design of the system proposed in this thesis.

2.1 Semantic and Transformer-Based Job Matching

Early improvements over keyword matching emerged through semantic similarity models, which aimed to interpret the conceptual relevance between resumes and job descriptions. Research by Ajjam and Al-Raweshidy demonstrated that these approaches outperform keyword-based methods, achieving significantly higher similarity scores and enabling a more nuanced interpretation of the conceptual relevance between resumes and job descriptions [AAR25].

The adoption of transformer models marked a major shift in this domain. Architectures such as BERT and its derivatives [RG19] enabled richer contextual understanding and improved generalization across career-related tasks. The development of domain-specific models represented a pivotal advancement in this field. For instance, Rosenberger et al. proposed CareerBERT, a novel approach that leverages the power of unstructured textual data sources, that uses jobGBERT [GBC22] to embed resumes and job categories derived from the standardized European Skills, Competences, and Occupations (ESCO) taxonomy [RWW+25]. By pre-training on specialized career data, such models create a shared embedding space that more accurately captures the intricacies of professional qualifications. Their approach demonstrated comparable, though more reliable, performance to that of GPT-4 while exhibiting significantly superior computational efficiency [RWW+25].

These developments highlight the steady shift toward embedding-based architectures capable of encoding job content beyond surface-level keywords. However, most domain-specific transformer models rely heavily on curated data sources or custom pretraining pipelines, which limits accessibility.

2.2 Industrial-Scale Job Matching Architectures

Parallel to academic research, large technology companies have integrated semantic embeddings into production-level job Matching systems. CareerBuilder, a company, that" operates the largest job posting board in the U.S.", developed an embedding-based job matching system that constructs fused embeddings from text, semantic entities, and location data to handle matching at a massive scale [ZWS+21]. Their two-stage process, involving approximate nearest neighbor search followed by a reranking model, led to significant gains in user engagement and match quality. LinkedIn's LinkSAGE framework represents a significant advancement in the field of network analysis, incorporating Graph Neural Networks (GNN) to model the intricate and interwoven relationships within its professional network [LWH+24]. LinkSAGE is using GraphSAGE architecture, which is "a general inductive framework that leverages node feature information (e.g., text attributes) to efficiently generate node embeddings for previously unseen data" [HYL17]. By conceptualizing members, employment opportunities, and competencies as nodes in a comprehensive graph, LinkSAGE is able to capture implicit signals and deliver highly relevant recommendations. Despite its effectiveness, this architecture poses significant barriers for broader adoption: it depends on massive proprietary datasets and requires extensive computational infrastructure, making it unsuitable for open, platform-aggregating systems such as the one developed in this thesis. Nevertheless, LinkSAGE demonstrates a key insight for the field: effective job matching requires modeling the detailed relationships between applicant skills and job requirements. This principle directly informs the methodology of the approach proposed in this thesis.

2.3 Addressing Data Challenges: Augmentation and Hard-Negative Mining

As semantic models grew more advanced, researchers increasingly focused on addressing data sparsity and imbalance—persistent challenges in recruitment datasets. The CONFIT V2 framework introduces a novel strategy to enrich training data by using LLMs to generate hypothetical reference resumes (Hyre) [YXX+25]. These synthetic examples are combined with Runner-Up Hard-Negative Mining, a technique that selects job–resume pairs which appear similar but are incorrect, forcing the model to learn subtle distinctions between suitable and unsuitable matches. This method significantly improves model robustness and highlights the growing role of LLMs in enhancing contrastive learning processes.

Although such methods push performance forward, they require substantial computational resources and specialized training pipelines. This dependency limits their applicability in lightweight, low-budget, or open-source contexts.

2.4 Related Work

Following the broader developments in job matching, this section examines research directly related to the two core techniques used in this thesis: (1) LLM-based requirements extraction and (2) embedding-driven similarity search.

2.4.1 Requirements Extraction using LLMs

LLMs have demonstrated strong capabilities in transforming unstructured job descriptions into structured information. Howison et al. demonstrated the use of LLM to extract structured labor market data, such as education requirements and job types, from real-world job ads with statistically reliable results [HEM+25]. Similarly, Herandi et al. proposed Skill-LLM, a fine-tuned LLM optimized for skill extraction, significantly outperforming standard Named Entity Recognition (NER) baselines in identifying hard skills [HLL+24]. While fine-tuned models achieve strong performance, they require task-specific annotations and training. In contrast, methods such as those explored by Nguyen et al. demonstrate that general-purpose LLMs can achieve competitive results through few-shot prompting, particularly when dealing with syntactically complex skill mentions [NZMB24]. This reinforces the choice of this thesis to rely on general-purpose, open-source LLMs without any fine-tuning, thereby increasing accessibility and adaptability.

2.4.2 Embedding and Similarity Search in Job Matching

Embedding-based similarity search has become a cornerstone of modern job recommendation systems. Domain-specific models such as CareerBERT achieve strong performance by leveraging structured taxonomies [RWW+25]. However, general-purpose models such as sentence-transformers (e.g., all-MiniLM-L6-v2) have also shown excellent retrieval performance without task-specific training, as demonstrated by Kurek et al. [KLB+24]. Building on this foundation, this thesis employs the model to process extracted requirements, enabling the system to capture semantic relationships effectively.

2.5 Research Gap

The literature reviewed in this chapter demonstrates considerable progress in semantic job matching, domain-specific transformer models, graph-based architectures, and data-augmentation techniques. Despite these advancements, several gaps persist:

- **Open-Source:** Most high-performance matching architectures rely on proprietary architectures, datasets,limiting their transparency, reproducibility, and accessibility for further research.
- **Training Required:** State-of-the-art approaches typically depend on extensive supervised training,domain-specific pre-training, or fine-tuning. These methods re-

quire large annotated datasets and significant computational resources, which limits their applicability in lightweight or low-resource contexts where such resources are often unavailable.

- **Utilization of LLMs:** Although recent studies demonstrate the effectiveness of LLMs for extracting structured requirements from unstructured job descriptions, existing job-matching architectures rarely incorporate LLM-based extraction as a central mechanism.

2.5.1 LinkSAGE

LinkSAGE is based on GNNs. The model architecture is defined as an "encoder-decoder GNN model", where the encoder utilizes the GraphSAGE algorithm to generate embeddings by aggregating neighbor information [LWH+24]. Furthermore, model training is an essential component of LinkSAGE. This process is sequential: the system first employs "inductive graph learning on a heterogeneous, evolving graph" to train a GNN encoder, which is subsequently integrated into downstream ranking models via transfer learning [LWH+24]. In terms of accessibility, LinkSAGE is a proprietary system and is not described as open source, as there is no link to a code repository for LinkSAGE provided in the paper.

2.5.2 CONFIT V2

As previously mentioned in section 2.3, LLMs are an important component of CONFIT V2. It employs HYRE, a technique that uses an LLM "to generate a hypothetical resume and augment the job post" [YXX+25]. It also requires a distinct training phase centered on contrastive learning to optimize resume-job compatibility, where the encoder is trained using a specific loss function described as "modified contrastive learning loss" to distinguish between matching and non-matching pairs [YXX+25]. The training process is iterative, involving an initial training phase followed by a RUM step, as explained in section 2.3. Regarding accessibility, the work is designated to be open source. As Yu et al. state, they "will open-source our code and data (under license agreements) to provide a strong baseline for future research" [YXX+25].

2.5.3 CareerBuilder

The framework proposed by CareerBuilder relies on specific training protocols rather than pre-trained, zero-shot inference. First, it trains "an end-to-end Deep Learning Embedding Model (DLEM) on a supervised learning task" utilizing "job application data" [ZWS+21]. The system also employs a representation learning model for the job-skill information graph, which uses "Bayesian personalized ranking and margin-based loss functions to learn the vector representation"[ZWS+21]. The architecture does not employ LLMs. The system is described as a proprietary in-house solution that serves as

the "core technology and service platform in CareerBuilder" [ZWS+21], therfore it is not open source.

To highlight how this work distinguishes itself, Table 2.1 summarizes the gaps explained above by comparing the discussed approaches against the proposed LLM-based requirement extraction and token-embedding-driven similarity search.

Table 2.1: Systematic Comparison of Job Matching Approaches

Tools \ Characteristics	Uses LLMs	Open-Source	Training Required
LinkSAGE	✗	✗	✓
CareerBuilder	✗	✗	✓
CONFIT V2	✓	✓	✓
Approach of this thesis	✓	✓	✗

In summary, although existing systems achieve strong performance in specific settings, there remains no approach that is semantically robust, deployable using lightweight, fully open-source components, and architecturally transparent. This thesis addresses this gap by proposing a job-matching system that employs small open-source LLMs for structured requirements extraction together with embedding-based similarity search, thereby offering an accessible, computationally efficient, and interpretable-by-design alternative.

3 Conception

This chapter presents the conceptual design of the proposed job-matching system, which addresses the limitations identified in the previous chapters. Building on the research gap outlined in section 2.5, the goal of the conception is to translate the methodological requirements into a coherent system architecture that enables reliable requirements extraction and semantically informed similarity search. The system operates as a processing layer on top of the YourJobFinder platform, which serves as the data acquisition component by aggregating raw, unstructured job descriptions based on a user's initial query.

Figure 3.1 illustrates the overall workflow. The YourJobFinder crawler provides raw HTML content, after which the job matching system developed in this thesis is applied. This system consists of two central components:

- a requirements extraction module based on open-source LLMs, and
- an embedding-driven similarity search module that identifies semantically relevant job postings.

The remainder of this chapter details the architectural design and theoretical foundations of the proposed system. Section 3.1 defines the unified data structure to requirements extraction and user profiles. Section 3.2 through section 3.6 describe the requirements extraction pipeline, focusing on handling raw data, constraining LLM outputs via Finite-State Machines, and evaluating extraction quality. Finally, subsection 3.7.1 presents the embedding-driven similarity search strategy, including the adaptation of the MaxSim algorithm for weighted field aggregation.

3.1 Data Structure for Job Matching

The core of the proposed system relies on a structured comparison between job requirements and user profiles. Figure 3.2 illustrates the fundamental data model that enables semantic matching. Both job postings and user profiles are decomposed into three key dimensions:

- **Skills** — technical and professional competencies
- **Experience** — work history and domain expertise

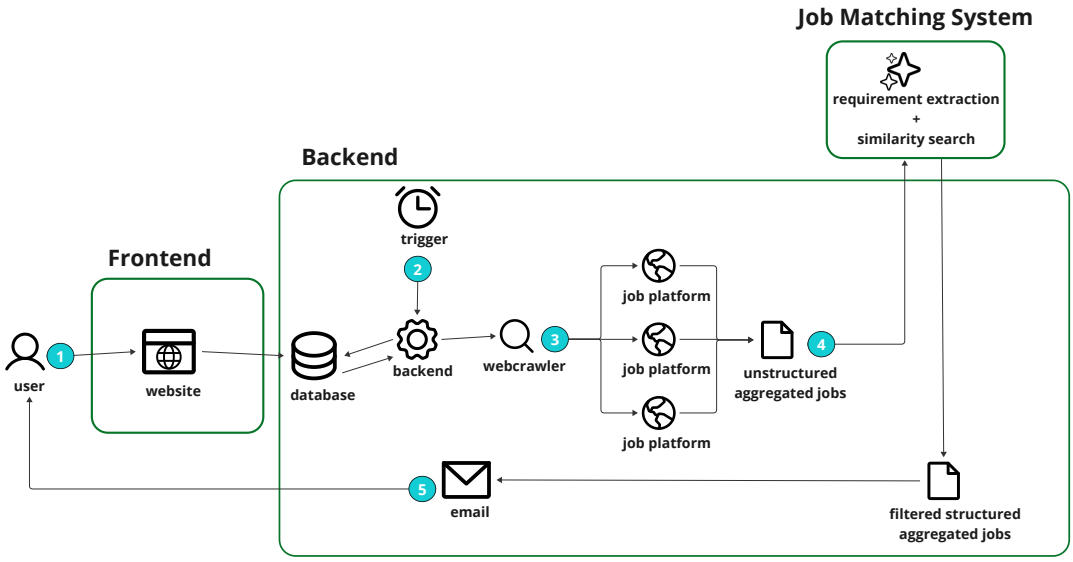

Figure 3.1: System architecture integrating YourJobFinder with the proposed job-matching system.

- **Qualification** — formal education and certifications

These structured representations are encoded in a common Pydantic Model [CJR+] for both job descriptions and user profiles. The requirements extraction module converts unstructured job descriptions into this schema so that they share the same structure as user profiles. With this common schema, the system can compare skills, experience, and qualification in a consistent way, forming the basis for the LLM extraction step and the subsequent embedding-driven similarity search.

3.2 Handling Raw Data

The pipeline begins with raw HTML job postings retrieved by the YourJobFinder crawler. In their original form, these documents often contain large amounts of non-informative content, such as JavaScript, CSS, advertisements, and unrelated boilerplate text. To create a structured and semantically meaningful input for the LLM, the HTML is converted into Markdown using the *markdownify* library [DT25]. This preprocessing step ensures that only the textual content relevant to the job description is retained, thereby reducing noise and improving extraction quality.

3.3 Schema-Guided LLM Output

The requirements extraction module must produce output that conforms to the predefined Pydantic schema described in section 3.1. However, LLMs may produce syntactically malformed JSON, ignore required fields, or include extraneous content that violates

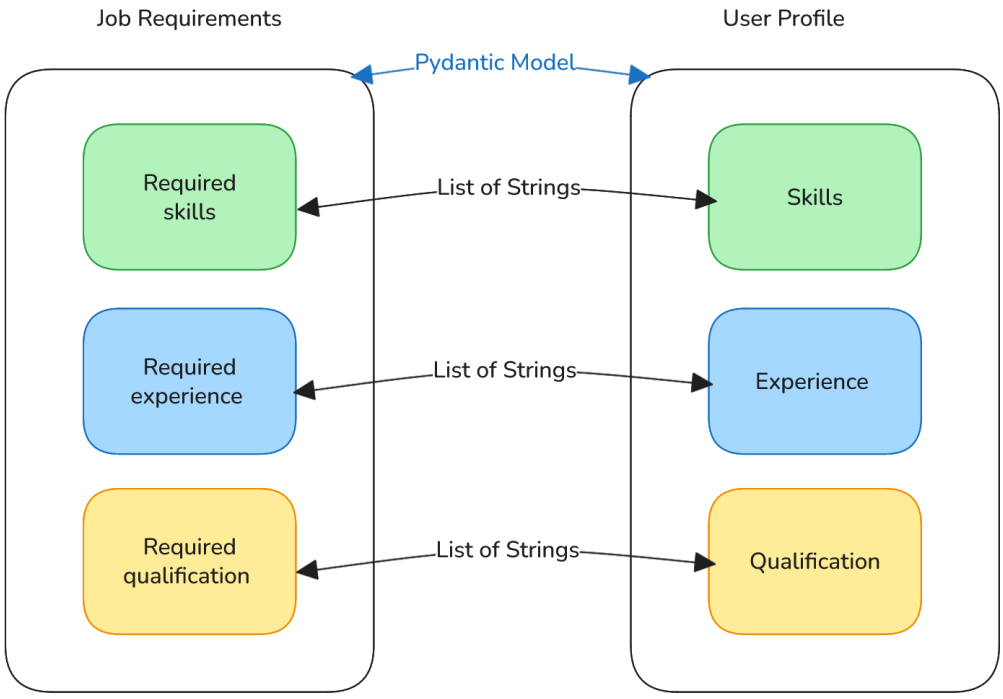

Figure 3.2: Data structure mapping between job requirements and user profile for semantic matching.

the schema [SPS+24]. To address this challenge, the system employs guided generation, a technique that constrains the LLM's output at inference time to ensure conformance with a formal specification.

The theoretical foundation for this approach is provided by Willard and Louf [WL23], who reformulate the problem of constrained text generation in terms of finite-state machines (FSMs). Their insight is that regular expressions and context-free grammars (CFGs) can be used to construct an index over the language model's vocabulary, enabling efficient determination of valid next tokens at each generation step.

3.3.1 Finite-State Machine Formulation

The approach relies on fundamental concepts from automata theory. A finite automaton (FA) is formally defined as a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is a finite alphabet, $\delta : Q \times \Sigma \to Q$ is the transition function, $q_0 \in Q$ is the start state, and $F \subseteq Q$ is the set of accept states [Sip96]. Regular expressions, which describe patterns such as digit sequences or identifiers, can be converted into equivalent finite automata that recognize exactly the strings matching the pattern [WL23].

In standard LLM inference, the model produces a probability distribution over the entire vocabulary for each token position. Guided generation modifies this process by applying a boolean mask m that restricts the support of the distribution to only those tokens that are valid according to the constraints:

$$\boldsymbol{\alpha} = \text{LLM}(S_t, \boldsymbol{\theta})$$
$$\tilde{\boldsymbol{\alpha}} = m(S_t) \odot \boldsymbol{\alpha}$$
$$\tilde{s}_{t+1} \sim \text{Categorical}(\tilde{\boldsymbol{\alpha}})$$

where $\boldsymbol{\alpha}$ represents the logits produced by the LLM, m is the mask function, and $\odot$ denotes element-wise multiplication. The resulting masked distribution $\tilde{\boldsymbol{\alpha}}$ assigns zero probability to invalid tokens while preserving the relative probabilities of valid continuations [WL23].

The efficiency of this approach stems from pre-computing an index $\sigma : Q \to \mathcal{P}(\mathcal{V})$ that maps each FSM state $q \in Q$ to the subset of vocabulary tokens that would be accepted in that state. This index is constructed once per schema and reused across all generation calls, reducing the per-token computational cost from $\mathcal{O}(N)$ to $\mathcal{O}(1)$ on average, where N is the vocabulary size [WL23].

Figure 3.3 illustrates this process of sampling tokens for the regular expression `[0-9]:[0-9]`, which could represent, for example, a match score. Assume the vocabulary $\mathcal{V}$ contains only the set of strings {"2","14","7","T","\",":","=","i"}. In the beginning, the FSM is in state q_0 and only tokens representing a one-digit number are valid; therefore, "14" and "T" are masked and their logits are set to zero. After sampling a valid digit, the FSM transitions to state q_1, where only the colon token is valid. Finally, after sampling the colon, the FSM reaches state q_2, where again only digit tokens are valid. At each generation step, the current FSM state determines which vocabulary tokens are valid continuations. Invalid tokens receive a mask value of zero, effectively removing them from consideration during sampling.

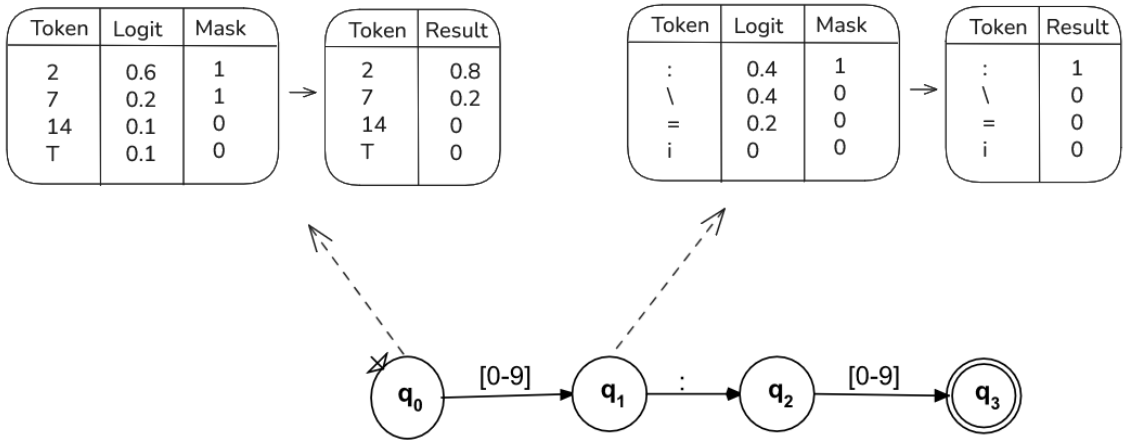

Figure 3.3: FSM-guided token generation for the regular expression `[0-9]:[0-9]`. Inspired by [Sou24]

3.3.2 Extension to Context-Free Grammars

While finite automata suffice for regular languages, structured data formats such as JSON require the expressive power of context-free grammars (CFGs). CFGs can describe nested structures with arbitrary depth, such as arrays containing objects that themselves contain arrays. To recognize such languages, Willard and Louf extend their approach using pushdown automata (PDAs).

A pushdown automaton extends a finite automaton with a stack, enabling it to recognize context-free languages. Formally, a PDA is defined as a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Γ is the stack alphabet and the transition function $\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \to \mathcal{P}(Q \times \Gamma_\epsilon)$ additionally reads from and writes to the stack [Sip96]. The stack allows the automaton to remember an unbounded amount of information, which is essential for matching nested brackets and enforcing hierarchical structure.

In the context of this thesis, this mechanism guarantees that the extracted job requirements strictly adhere to the schema defined in Listing 4.1. By maintaining the stack configuration of the PDA, the system ensures that keys (e.g., "experience") are followed by valid values (e.g., arrays or strings) and that all braces and quotes are correctly closed. This eliminates the need for error-correction logic and ensures the extracted data is immediately machine-readable.

3.4 LLM Models

A key design constraint of this system is that no fine-tuning is used. This means the quality of the requirements extraction depends entirely on the pre-trained, zero-shot abilities of the selected LLM. However, general-purpose models differ significantly in how well they follow instructions, produce structured output, and handle schema-guided generation. Choosing a model without evaluation would therefore risk weak or inconsistent extraction performance.

Therefore, a comparison of multiple models is required to determine which open-source LLM is best suited for this task. The models evaluated in this thesis represent state-of-the-art open-source architectures available as of October 2025. They were selected based on their parameter size, availability, and their compatibility with constrained inference.

Table 3.1: Model Parameter Sizes

Model Name	**Parameter Size**
Qwen3 [Qwe25]	8B
GLM-4-9B-0414 [GZX+24]	9B
mistral-7B-instruct [JSM+23]	7B
Llama-3.1-8B-Instruct [WKN+25]	8B

3.5 Managing Input Data Chunking Strategies

A limitation of Transformer-based language models is their restricted context size [JA24], therefore to be able to process job descriptions independently of their length, the input text should be split into smaller chunks. There are different methods of chunking text data for LLM processing. It can be grouped into different categories. The following categories are used in this work: :

- **Fixed-size chunking** It involves splitting text into chunks based on a predetermined number of tokens, with the option to include overlapping sections to preserve semantic context [Sou24]. It is widely used because it is simple, computationally inexpensive, and does not require specialized tools [Sou24].
- **Structure-based chunking**: This method splits the text based on its structure and hierarchy [Pin25], in this case newlines, paragraphs, or Markdown headers. This method also aims to preserve the semantic integrity of the content [Pin25].

This work uses a hybrid approach that combines the semantic preservation of structure-based chunking with the operational safety of fixed-size chunking. The primary strategy involves splitting the job descriptions by Markdown headers to maintain the coherence of logical sections. To address the limitation of variable chunk sizes, a fixed-size slicing mechanism is employed as a fallback constraint, ensuring that no single chunk exceeds the context window regardless of its structural length.

3.6 LLM Output Evaluation

Evaluating the correctness of extracted requirements poses a unique methodological challenge, as semantically equivalent requirements may be expressed in lexically diverse forms. For example, 'Proven ability to manage projects simultaneously' could be correctly summarized as 'Project management'. Even though the wording differs significantly. Relying on exact string matching would thus penalize valid extractions.

To address this, the system employs an automated evaluation strategy based on the 'LLM-as-a-judge' paradigm [ZCS+23]. This approach uses an independent, more capable LLM to assess the faithfulness of extracted JSON fields relative to the original job description. As Huyen mentions, "AI judges are fast, easy to use, and relatively cheap compared to human evaluators. They can also work without reference data" [Huy25]. The evaluation framework used in this thesis is implemented with DeepEval [YLD+24].

3.6.1 LLM-as-a-Judge

"The approach of using AI to evaluate AI is called AI as a judge or LLM as a judge" [Huy25]. Evaluations may produce numeric scores or qualitative assessments [WDN+24]. Using LLM-as-a-Judge is not just cost-effective, " Studies have shown that certain AI

judges are strongly correlated to human evaluators." [Huy25]. Zheng et al. found in their work that LLM judges and humans can achieve 85% agreement, while the agreement among human evaluators was 81% [ZCS+23].

3.6.2 G-Eval

The evaluation relies on the G-Eval metric, a prompt-based scoring method with three components: (1) task definition and criteria, (2) a chain-of-thought reasoning process used internally by the judge model, and (3) a probabilistic scoring function [LIX+23]. This mechanism enables controlled and interpretable evaluation of structured JSON outputs.

3.6.3 Metrics

Four criteria adapted from Yuan et al. [YRZ+24] guide the evaluation:

- **Correctness** — assesses whether extracted requirements are accurate and present in the input text.
- **Completeness** — evaluates whether all mandatory requirements have been captured.
- **Alignment** — measures whether the model avoids including 'preferred', 'bonus', or nice-to-have requirements.
- **Readability** — examines whether the output adheres to the defined schema, is syntactically valid, and correctly categorizes extracted items into the appropriate fields (skills, experience, or qualifications)

These metrics collectively ensure that extraction quality is judged not only by semantic fidelity but also by structural reliability.

3.7 Embedding and Similarity Search

Once the job requirements have been extracted and validated, the second component of the system performs semantic similarity matching. Both the structured job postings and the user query are encoded as vector representations using the *all-MiniLM-L6-v2* sentence transformer model. As discussed in subsection 2.4.2, this model offers strong zero-shot performance for retrieval-based tasks in job matching.
The skills fields are weighted more because as studies show skills are more important in job application context. A study by Bone et al., which analysed online job postings from 2018 to 2024, points out "We show that individual skills, rather than formal education requirements, have become increasingly important features of job advertisements for AI roles." [ES23].

Task of similarity search is to give a similarity score by embedding the extracted requirements and user profile, which have the same structure, and comparing them using MaxSim approach.

3.7.1 Vector Embedding

An embedding effectively maps discrete objects, like words or documents, to points in a continuous vector space [Ras24]. Figure 3.4 illustrates this transformation from raw input to a vector representation, which serves as a good way to translate human language into computer language. [JA24]

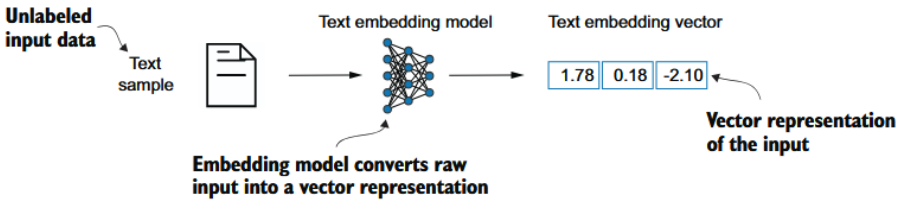

Figure 3.4: the process of converting raw data into a three-dimensional numerical vector (Source: [Ras24]).

Embeddings are fixed in size, with their values representing various properties that collectively capture the meaning of a word in a way that is interpretable by a computer [JA24]. A key advantage of using embeddings is the ability to measure semantic similarity. By calculating distance metrics in the vector space, it is possible to determine how closely related two words are; words sharing similar meanings tend to be clustered closer together [JA24], Figure 3.5 illustrates how words with similar meaning tend to be closer to each other, if it was possible to compress the embeddings into a two-dimensional representation [JA24].

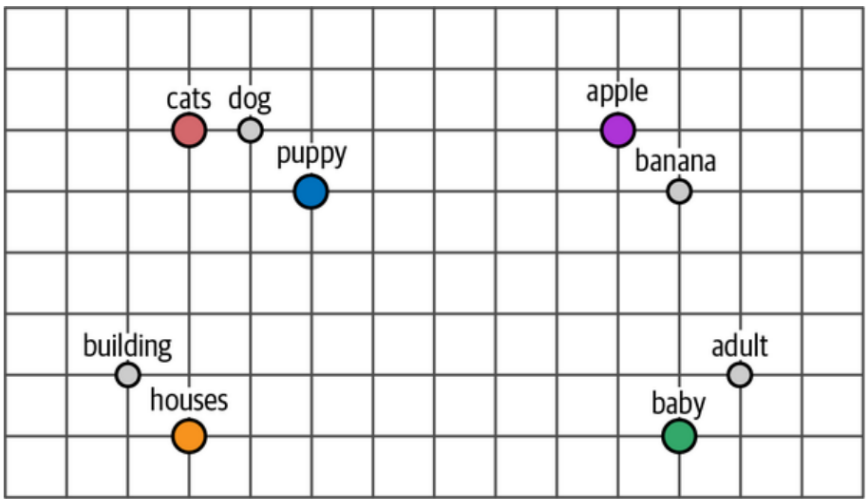

Figure 3.5: Embeddings of words that are similar will be close to each other in two-dimensional space (Source: [JA24]).

3.7.2 MaxSim

Computing a meaningful similarity score between extracted job requirements and a user profile is non-trivial. A naive pairwise comparison fails because lists often differ in length and items may appear in different orders. To address this, this thesis adopts a matching strategy inspired by the MaxSim framework originally introduced by Chan and Ng [CN08] for machine translation evaluation.

In the original MaxSim paper, the authors utilize the Kuhn-Munkres algorithm to find a maximum weight matching in a bipartite graph, ensuring a strict 1-to-1 alignment between items. In contrast, this thesis employs a row-wise maximum aggregation over the similarity matrix. This modification is a deliberate design choice: in job matching, the number of requirements often exceeds the number of profile items, and a single broad competency in a user profile may semantically satisfy multiple specific job requirements.

The algorithm proceeds in the following steps:

1. **Embedding:** Both the job requirements $J = \{j_1, \ldots, j_n\}$ and user attributes $U = \{u_1, \ldots, u_m\}$ are mapped to a high-dimensional vector space.

2. **Similarity Matrix:** A similarity matrix C is computed, where each entry C_{ik} represents the cosine similarity between the i-th job requirement and the k-th user profile item.

3. **Max-Over-Rows Aggregation:** For every specific job requirement j_i, the system identifies the maximum similarity score found across all user items U:

The similarity score for a specific data field (e.g., Skills) is defined as the mean of these maximum satisfaction scores:

$$Score_{field} = \frac{1}{|J|} \sum_{i=1}^{|J|} \max_{k}(C_{ik}) \tag{3.1}$$

In the context of Figure 3.6, the calculation would proceed as follows:

- Requirement *Python*: Max match is 1.0 (from User: Python).
- Requirement *Java*: Max match is 0.4 (from User: Python).
- Requirement *Teamwork*: Max match is 0.8 (from User: Team player).

The final score for this section would be the average of these maximums: $(1.0 + 0.4 + 0.8)/3 \approx 0.73$.

This approach ensures that every job requirement is evaluated against the user's best matching attribute, effectively measuring how much of the job description is "covered" by the user's profile.

	User Profile		
Job Requirements		Python	Team player
	Python	1.00	0.4
	Java	0.4	0.2
	Teamwork	0.1	0.8

Figure 3.6: Visualization of the similarity matrix. The system identifies strong semantic alignment (green cells) between user skills and job requirements, even when exact wording differs.

3.7.3 Weighted Field Aggregation

Recognizing that not all data categories hold equal value in recruitment, the system segments the comparison into three dimensions: *Skills*, *Experiences*, and *Qualifications*. The final semantic match score is a weighted linear combination of these field scores:

$$Score_{total} = w_{skills} \cdot Score_{skills} + w_{exp} \cdot Score_{exp} + w_{qual} \cdot Score_{qual} \tag{3.2}$$

By assigning higher weights to skills, the model aligns with recent findings on the increasing importance of skills-based hiring as explained above.

4 Implementation

This chapter details the technical realization of the concept presented in the previous chapter. It describes the development environment, the data processing pipeline, the implementation of the requirements extraction using LLMs, the evaluation methodology, and the realization of the similarity search algorithms.

4.1 Development Environment

The system was implemented using the Python programming language (Version 3.10), chosen for its extensive ecosystem of data science and machine learning libraries. The core dependencies include:

- **Markdownify**: For converting raw HTML content into structured Markdown text.
- **Outlines**: To enforce structured JSON / Pydantic Model generation from the LLMs, ensuring deterministic output formats.
- **FastAPI**: For serving the inference engine as a scalable web service.
- **Sentence-Transformers**: For generating vector embeddings of requirements and user profile.
- **DeepEval**: For the automated "LLM-as-a-judge" evaluation pipeline.

All experiments and inference tasks were conducted on a dual GPU setup (2x 12GB NVIDIA RTX 3080), ensuring consistent performance benchmarks across all tested models. Figure 4.1 illustrates the high-level architecture of the implemented system.

4.2 Data Preprocessing

As outlined in the conception, the raw data consists of HTML documents. The preprocessing pipeline transforms this unstructured input into clean, chunked text suitable for LLM processing.

4.2.1 HTML to Markdown Conversion

The `markdownify` library is configured to strip unnecessary tags (such as scripts, styles, and images) while preserving the structural hierarchy of the document. Headings, lists,

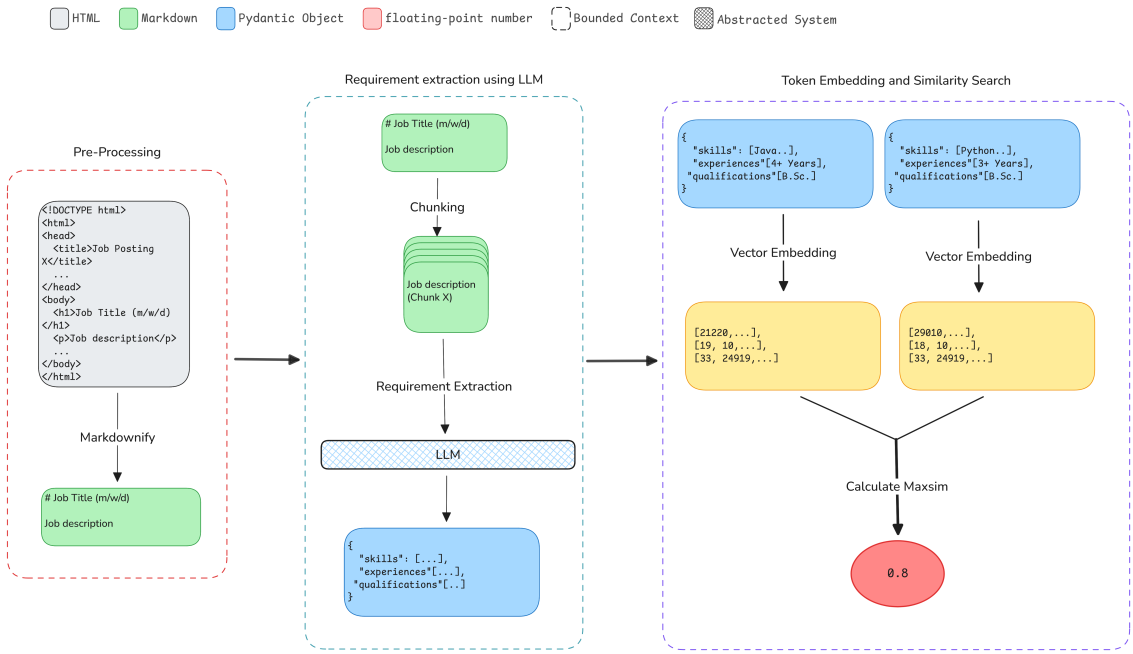

Figure 4.1: System Architecture

and paragraphs are retained, as these structural elements provide critical semantic cues for the extraction model. This reduction significantly lowers the token count while maintaining the semantic structure of the job posting.

4.2.2 Hybrid Chunking Strategy

To process job descriptions that exceed the context window of smaller LLMs, a hybrid chunking strategy is implemented. The text is primarily split based on Markdown headers (using the regex `r'(#{1,6}\s+.*?\n)'`). This ensures that logical sections—such as 'Responsibilities' or 'Requirements'—remain intact within a single context window.

A fallback mechanism is implemented for cases where a section exceeds the maximum token limit. If a semantic chunk exceeds the configured size (defaulting to approximately 12,000 tokens, estimated via character count), it is further divided based on a strict character limit to prevent memory overflows during inference.

4.3 Requirements Extraction

The core component of the system is the extraction of structured data. To address the non-deterministic nature of LLMs, the `outlines` library is employed to constrain the generation process using Finite State Machine (FSM) logic.

4.3.1 Output Structure Definition

A strict Pydantic model defines the output schema. This ensures that every processed job description yields a consistent data structure, regardless of the underlying LLM used. The schema is defined as follows:

Listing 4.1: Pydantic Schema for Requirements Extraction

```
1
2  from typing import List
3  from pydantic import BaseModel, Field
4
5  class Requirements(BaseModel):
6      """Normalized requirements schema for job extraction outputs."""
7      skills: List[str] = Field(
8          default_factory=list,
9          description="List of required skills"
10     )
11     experiences: List[str] = Field(
12         default_factory=list,
13         description="List of experience requirements"
14     )
15     qualifications: List[str] = Field(
16         default_factory=list,
17         description="List of qualifications/certifications"
18     )
```

4.3.2 Prompt Engineering

The prompt provided to the LLM consists of a system instruction defining the persona and the task, followed by the specific job description chunk. Figure 4.2 shows the system prompt, which requires the LLM to only extract must-have requirements, be concise, and only respond in the defined categories.

You are an expert job requirements extractor. Analyze the following job description and extract ONLY the mandatory requirements.
Instructions:
1. Focus ONLY on clearly stated MUST-HAVE requirements (required, essential, necessary)
2. IGNORE all 'nice-to-have', 'preferred', or 'bonus' skills/qualifications
3. Be specific and concise - extract exact requirements, not general topics
4. Categorize each requirement as either: 'skills', 'experiences', or 'qualifications'

Figure 4.2: System prompt used for requirements extraction

4.3.3 Inference

The inference logic is encapsulated within a generic `LLMExtractor` class, designed to be model-agnostic. This allows for seamless switching between different open-source models (e.g., Llama, Mistral, Qwen) by simply updating a configuration dictionary.

The extractor initializes the model and tokenizer, and wraps them with the `outlines.Generator` class. The generation process is constrained to strictly follow the schema defined in Listing 4.1.

Listing 4.2: LLM Extractor Implementation

```
class LLMExtractor:
    def __init__(self, model_id, chunk_size, device_kwargs=None):
        self.model_id = model_id
        self.chunk_size = chunk_size
        self.device_kwargs = device_kwargs or {}
        self.generator = None
        self._load_model()

    def _load_model(self):
        # ... Memory management and cleanup ...
        self.tokenizer = AutoTokenizer.from_pretrained(self.model_id)
        self.model = AutoModelForCausalLM.from_pretrained(
            self.model_id,
            **self.device_kwargs,
            trust_remote_code=True,
        )
        # Initialize Outlines generator with schema constraint
        outlines_model = from_transformers(self.model, self.tokenizer)
        self.generator = Generator(outlines_model, Requirements)
```

This design simplifies the process of managing models; adding a new model or removing an existing one only requires adding a JSON entry. For example, Listing 4.3 shows how to add the Qwen3-8B model.

Listing 4.3: LLM Definition

```
{
  "qwen3-8b": {
    "model_id": "Qwen/Qwen3-8B",
    "chunk_size": 12000,
    "device_kwargs": {
      "device_map": "auto",
      "dtype": "torch.bfloat16"
    }
  }
}
```

4.3.4 Exception handling

After the output is generated, the output structure is validated against the Pydantic model using the built-in `validate_json()` method. While the outlines library is designed to ensure that the model output follows the schema shown in Listing 4.1, issues can still occur during generation. For example, if the model reaches its `max_new_tokens` limit, it may produce incomplete or invalid JSON that does not pass validation. When this happens, the system handles the exception and returns a Requirements object in which the fields are empty lists. This approach allows the rest of the pipeline to run without failing, but it also means that there is no clear indication that the output was invalid, which makes troubleshooting more challenging.

4.3.5 Result Aggregation

Since job descriptions are processed in chunks, a single job posting may yield multiple `Requirements` objects. An aggregation step is implemented to merge these partial results. A `defaultdict(set)` is utilized to collect skills, experience, and qualifications across all chunks. The use of sets automatically handles deduplication, ensuring that if a skill is mentioned in multiple sections, it appears only once in the final output.

4.3.6 G-Eval Metrics

To integrate the evaluation strategy described in subsection 3.6.2, this work implements four metrics defined in subsection 3.6.3. Each metric specifies a natural-language description of the goal, a sequence of evaluation steps, the relevant test case fields (job description and/or extracted requirements), and a fixed success threshold of 0.7. Listing 4.4 shows an example metric definition.

Listing 4.4: Implementation of a G-Eval metric

```
from deepeval.metrics import GEval

def create_evaluation_metrics(model: DeepEvalBaseLLM) -> List[GEval]:
    correctness_metric = GEval(
        name="Correctness",
        criteria=(
        "Evaluate␣whether␣the␣extracted␣job␣requirements␣are"
        "accurate␣and␣actually␣present␣in␣the␣input␣text."
        ),
        evaluation_steps=[
            "Read␣the␣input␣job␣description␣carefully",
            "Compare␣each␣extracted␣requirement␣with␣the␣input␣text",
            # further evaluation steps ...
        ],
```

```
        evaluation_params=[
            LLMTestCaseParams.INPUT,
            LLMTestCaseParams.ACTUAL_OUTPUT,
        ],
        model=model,
        threshold=0.7,
    )
    # additional metrics: Completeness, Alignment , Readability
    return [correctness_metric, ...]
```

The evaluation pipeline first loads all JSON outputs from the requirements extraction stage and converts them into `LLMTestCase` objects, where the original job description serves as the input and the serialized `Requirements` object is used as `actual_output`. Using DeepEval's `evaluate` function together with an external judge model, these metrics are executed asynchronously over all test cases, and the resulting scores, rationales are collected.

4.4 Embedding and Similarity Search

This section describes the implementation of the similarity matching component introduced in the conception. The goal is to compute a semantic similarity score between extracted job requirements and a user profile.

4.4.1 Vector Embedding

The embedding functionality is implemented using the `sentence-transformers` library, which provides pre-trained transformer models optimized for semantic similarity tasks.[RG19] The model *all-MiniLM-L6-v2* is loaded once as a singleton to avoid repeated initialization overhead during inference.

4.4.2 MaxSim

The MaxSim algorithm described in subsection 3.7.2 is implemented in the `compute_maxsim` function. The implementation directly follows the conceptual formulation: for each job requirement, find the maximum similarity to any user profile item, then average these maximum scores.

Listing 4.5: Implementation of MaxSim Logic

```
def compute_maxsim(user_items: List[str],
                   job_items: List[str],
                   model: SentenceTransformer) -> float:
    if not user_items or not job_items:
        return 0.0
```

```
6
7       # 1. Encoding
8       user_embeddings = model.encode(user_items, convert_to_tensor=True)
9       job_embeddings = model.encode(job_items, convert_to_tensor=True)
10
11      # 2. Similarity Matrix
12      cosine_scores = util.cos_sim(job_embeddings, user_embeddings)
13
14      # 3. Max-Over-Rows (dim=1)
15      max_scores, _ = torch.max(cosine_scores, dim=1)
16
17      # 4. Aggregation
18      return max_scores.mean().item()
```

The `util.cos_sim` function from `sentence-transformers` computes the full similarity matrix C between all job and user embeddings. The `torch.max` operation along dimension 1 extracts the maximum similarity for each job requirement (row), and `mean()` computes the final field score as defined in Equation 3.1.

4.4.3 Weighted Aggregation

The final similarity score combines the MaxSim scores from all three fields using configurable weights. By default, skills receive a weight of 0.5, experiences 0.3, and qualifications 0.2, reflecting the prioritization of skills-based matching discussed in the conception.

Listing 4.6: Weighted score aggregation

```
1  def compute_similarity(user_profile, extracted_requirements,
2                         weights=None) -> SimilarityScore:
3      if weights is None:
4          weights = {"skills": 0.5, "experiences": 0.3,
5                     "qualifications": 0.2}
6
7      model = get_model()
8      overall_scores = {}
9
10     for field in ['skills', 'experiences', 'qualifications']:
11         user_items = getattr(user_profile, field, [])
12         job_items = getattr(extracted_requirements, field, [])
13         overall_scores[field] = compute_maxsim(user_items,
14                                                job_items, model)
15
16     weighted_score = sum(
17         overall_scores[field] * weights.get(field, 0.0)
```

```
18          for field in overall_scores
19      )
20      return SimilarityScore(score=weighted_score)
```

This design allows the weights to be adjusted via the API without modifying the codebase, enabling experimentation with different prioritization strategies.

5 Evaluation

This chapter presents the empirical evaluation of the system developed in this thesis and directly addresses the two research questions introduced in chapter 1. The evaluation is structured into three components that build upon one another.

First, this chapter analyzes the characteristics of the dataset used for evaluation. This includes an examination of the distribution of job postings across sources, industries, and description lengths, providing the necessary context for understanding model behavior and extraction performance.

Secondly, evaluates how effectively open-source LLMs extract structured job requirements from unstructured job descriptions (RQ1). This is assessed through a quantitative analysis using G-Eval metrics within an automated "LLM-as-a-Judge" framework, complemented by a qualitative error analysis that identifies systematic strengths and weaknesses of the tested models.

Third, the chapter examines how the proposed job-matching approach compares to conventional job search engines in terms of relevance and accuracy of retrieved postings (RQ2). This includes an evaluation of the similarity search pipeline and a comparison against traditional title-based retrieval strategies.

Together, these three components provide a comprehensive empirical foundation for assessing the system's performance, its practical utility, and its limitations, thereby preparing the ground for the discussion in the Outlook.

5.1 Dataset

The empirical evaluation is based on a corpus of $N = 209$ real-world job descriptions. To ensure the dataset reflects the practical challenges of a live job aggregator, links to the postings were retrieved directly from the YourJobFinder database and scraped using the platform's existing scraping tools.

The raw HTML content of each posting was subsequently converted into Markdown .

5.1.1 Source and Industry Distribution

Figure 5.1 illustrates the distribution of job postings by source platform. The dataset is heavily dominated by major job aggregators, with Indeed ($n = 158$) and LinkedIn ($n = 40$) accounting for approximately 95% of the corpus. The remaining postings originate from niche platforms or individual company career pages.

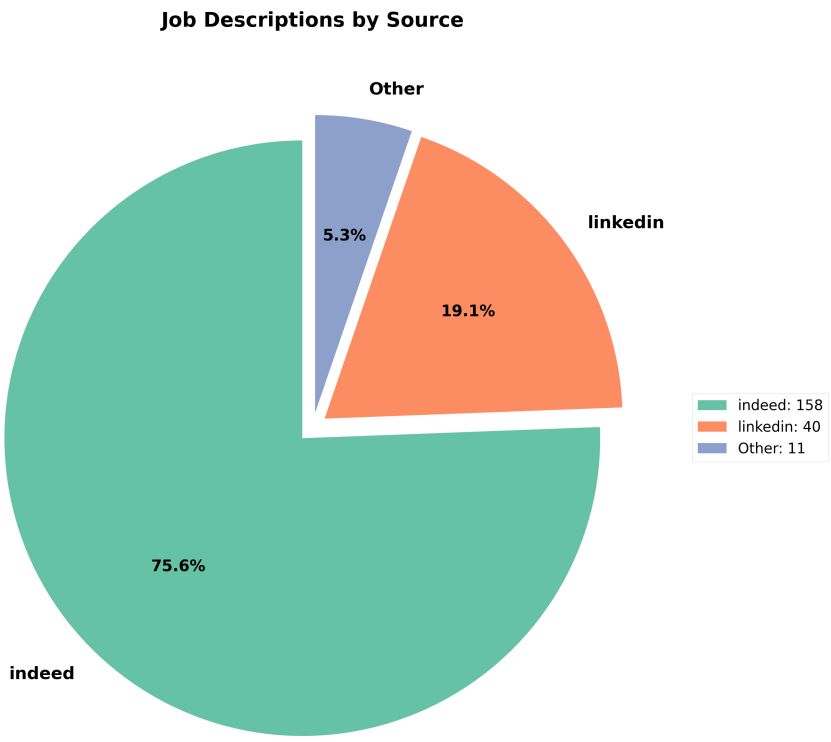

Figure 5.1: Distribution of job postings by source platform

In terms of sectoral distribution, Figure 5.2 shows that the dataset is imbalanced, as the technology sector is the largest single category (comprising roughly one-third of the dataset). However, there is significant representation from healthcare, business, and engineering. This diversity is essential for evaluating the model's generalization capabilities, ensuring that the extraction logic is not overfitted to the specific terminology or formatting conventions of the tech industry.

Notably, the 'Other' category in Figure 5.2 represents a specific edge case involving an expired job posting. While expired listings typically return an HTTP 404 status code, this specific page remained active, displaying a list of related job recommendations instead of the original description. Because the page still contained significant textual content, it bypassed the scraper's emptiness filters. Although unintentional, this instance serves as a valuable adversarial example, testing the extraction model's behavior when presented with valid HTML structure but semantically irrelevant content, which will be discussed in subsection 5.3.3.

5.1.2 Length and Characteristics

Beyond content distribution, the length of the job descriptions is a critical variable for evaluating LLM behavior, particularly regarding context window limitations. Figure 5.3 presents a histogram of character counts across the dataset.

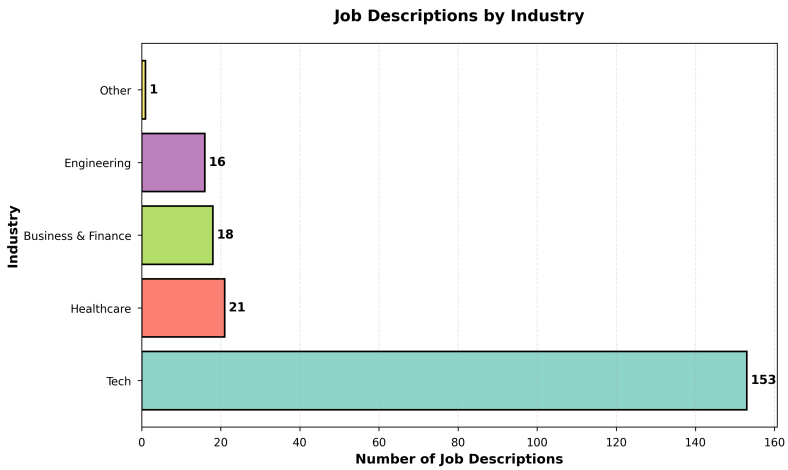

Figure 5.2: Distribution of job postings by industry sector

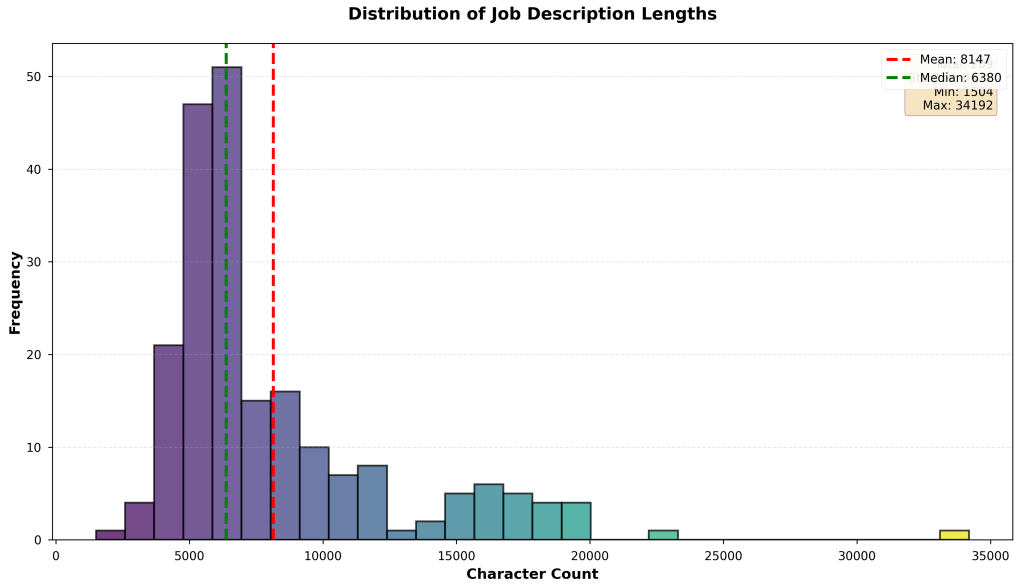

Figure 5.3: Histogram of job description lengths showing a right-skewed distribution

The distribution is strongly right-skewed, characterized by a significant disparity between the median length (6,380 characters) and the mean length (8,147 characters). While the majority of postings fall within the 4,000 to 8,000 character range, the dataset contains extreme outliers, with the longest single posting reaching 34,192 characters. This extreme variance—where the maximum length is nearly six times the median—validates the necessity of the hybrid chunking strategy introduced in subsection 4.2.2.

Figure 5.4 further decomposes these statistics by source. A distinct structural difference is observable between platforms: Indeed postings exhibit a highly concentrated distribution with lower variance, suggesting a more standardized listing format. In contrast, LinkedIn postings display a much wider distribution with a significantly higher median length. This indicates that LinkedIn descriptions are not only more verbose but also less

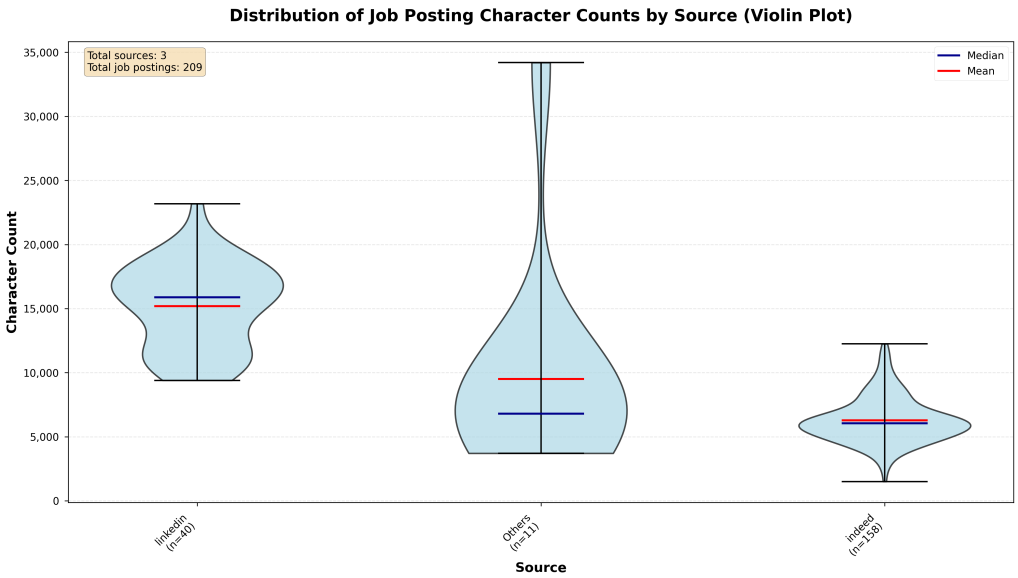

Figure 5.4: Violin plot comparing character count distributions by source

predictable in structure.

5.2 Evaluation Framework

To systematically assess extraction quality as described in subsection 3.6.2 using the DeepEval library, NVIDIA's hosted `gpt-oss-120b` model as an independent evaluator was employed. Each output is rated along the four criteria introduced in subsection 3.6.3, then model outputs are averaged across the entire dataset to compute final performance scores.

5.3 Performance Overview

Figure 5.5 presents the aggregated model ranking derived from the evaluation metrics. Qwen3-8B achieves the highest overall performance, followed by GLM-4-9B-0414 and Mistral-7B-Instruct-v0.3, with Llama-3.1-8B-Instruct showing the lowest aggregated scores.

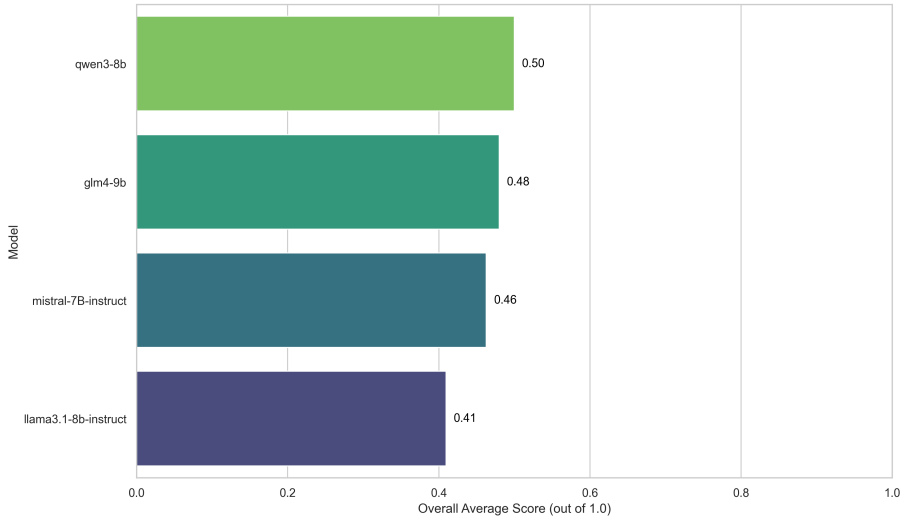

Figure 5.5: Aggregated overall ranking across all evaluation metrics, rounded to two decimal places.

To understand the drivers behind these overall scores, Table 5.1 details the breakdown across the specific G-Eval metrics. Each value represents a mean score across the evaluation corpus, where 1.0 signifies perfect compliance with the metric.

Table 5.1: Model performance across G-Eval metrics.

Model	Correctness	Completeness	Alignment	Readability
Qwen3-8B	**0.45**	0.45	**0.50**	**0.61**
GLM-4-9B-0414	0.43	**0.48**	0.43	0.58
Mistral-7B-Instruct-v0.3	0.44	0.40	0.46	0.55
Llama-3.1-8B-Instruct	0.38	0.37	0.39	0.49

Scores represent mean values over all 209 evaluated job descriptions, rounded to two decimal places.

5.3.1 Readability vs. Semantic Metrics

An important observation from Table 5.1 is that the Readability metric, designed to assess whether the output is valid JSON, became effectively redundant due to the system's architectural design. By employing the outlines library and a strict exception handling mechanism, the pipeline guarantees that the output is always a valid JSON object, even if it is an empty one returned upon generation failure. Consequently, high Readability scores were a structural artifact of the engineering pipeline rather than a reflection of the model's capabilities. This suggests that for constrained generation pipelines, evaluation should focus exclusively on semantic metrics rather than syntactic validity.

However, despite these architectural guarantees of structural validity, the Readability scores do not reach the maximum value of 1.0. This gap highlights a critical distinction captured by the G-Eval metric, which evaluates not only whether the output is structured, but also whether requirements are "properly categorized into skills, experiences, and qualifications". The failure to reach a perfect score indicates that while the models are forced to generate valid JSON, they still struggle with the semantic logic required to populate that structure correctly. The models frequently miscategorize requirements—for example, placing a formal degree under skills or a software tool under qualifications. Therefore, the Readability results lead to a clear conclusion: the problem of syntactic structure has been effectively solved through constraints (outlines and exception handling), leaving semantic reasoning and categorization as the primary bottleneck for open-source models.

5.3.2 Precision vs. Completeness Trade-offs

A clear trade-off emerges between Qwen3-8B and GLM-4-9B-0414. Qwen3-8B generally follows a conservative extraction pattern, rarely adding unsupported items and therefore achieving the highest precision. However, this sometimes results in missed implicit requirements, slightly lowering its completeness score. In contrast, GLM-4-9B-0414 adopts a more liberal extraction strategy, capturing a broader range of potential requirements—including many optional or contextually weakly supported items—boosting completeness at the expense of precision. For downstream filtering systems, conservative extraction is typically preferred to avoid incorrectly rejecting qualified candidates.

5.3.3 Copying Bias

The Edge case mentioned in subsection 5.1.1 caused an interesting behavior, where models such as Qwen3-8B and Llama-3.1-8B frequently hallucinated requirements that were not present in the input. Notably, the extracted requirements, such as "Bachelor's degree in Computer Science" and "PMP Certification", were identical to the examples provided in the system prompt's instructions. This behavior is best described as *copying bias*, where LLMs "copy answers from provided examples instead of learning the underlying pat-

terns" [AWT24]. Such behaviors pose a systematic risk in extraction pipelines, as models are prone to outputting artifacts from the prompt itself, leading to false negatives in the matching system.

5.4 Case Study

Based on section 5.3, to address RQ2 (section 1.2), Qwen3-8B was chosen for LLM-based requirement extraction and token-embedding-driven similarity search to compare job matching performance against LinkedIn's built-in job recommendation feature. LinkedIn serves as a practical baseline, as it suggests postings based on activity on LinkedIn and the information available in a user's profile (e.g., skills, experience) [Lin25].
A dedicated LinkedIn account was created to approximate a recent graduate seeking a full-time position after completing a bachelor's degree. The profile contained the following information:

- **Skills:** Machine Learning, Domain-Driven Design (DDD), Data Science, Spring Boot, Java, Python, Hibernate, PostgreSQL, R, Project Management
- **Qualifications:** Domain Driven Design, B.Sc. Wirtschaftsinformatik
- **Experience:** 4 years of software development

LinkedIn's recommended jobs for this profile are illustrated in Figure 5.6.

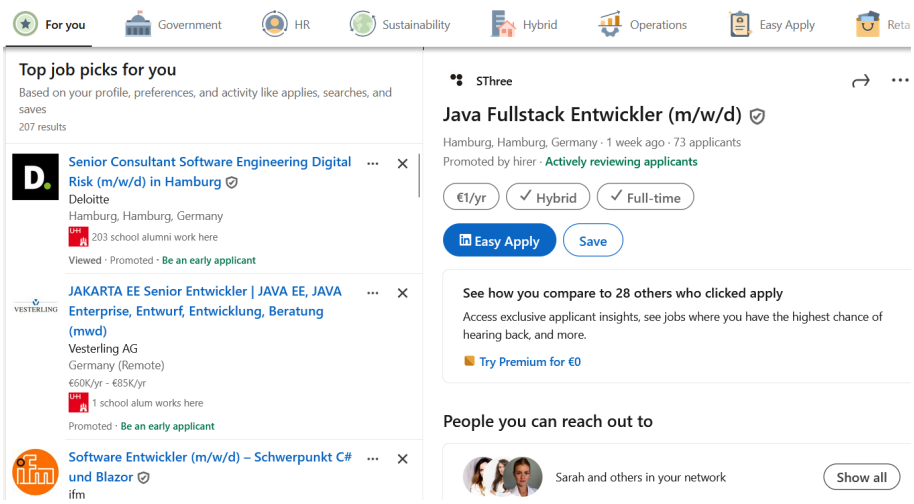

Figure 5.6: Example of LinkedIn job recommendations for the hypothetical profile.

5.4.1 Comparison Procedure

The comparison followed three steps:

1. The first 22 job recommendations by LinkedIn were collected.

2. Each job posting was manually marked as *suitable* or *not suitable* with a reason for the ground truth, reflecting the preference of the hypothetical user.

3. The same job descriptions were processed by the system developed in this thesis and similarity scores were computed for each job against the user profile.

5.4.2 Similarity Score Distribution

Figure 5.7 shows the bucketed similarity score distribution for the 22 LinkedIn-recommended jobs when using Qwen3-8B as the extraction model.

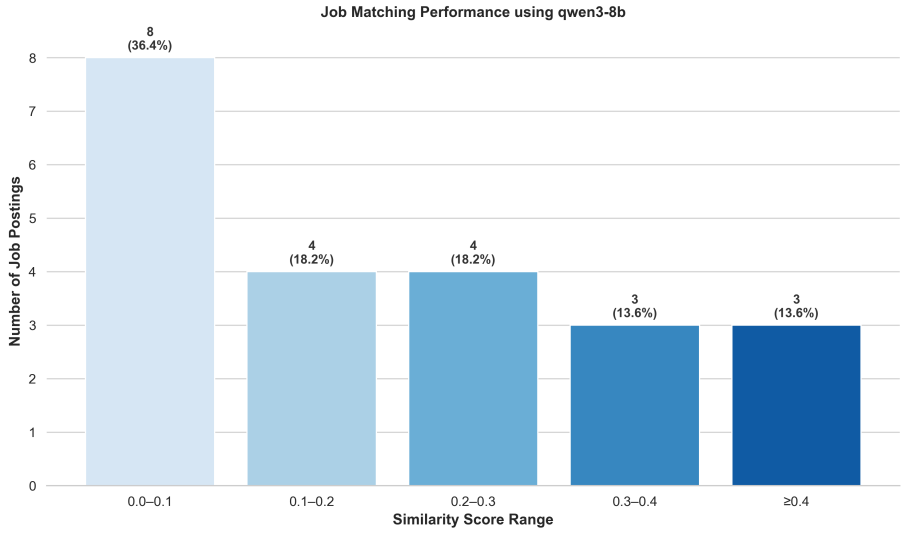

Figure 5.7: Bucketed similarity score distribution for LinkedIn-recommended jobs (Qwen3-8B).

The distribution in Figure 5.7 is strongly right-skewed: 9 out of 22 postings (39.1%) fall into the 0.0–0.1 bucket, while only 3 postings (13.0%) achieve a score of ≥ 0.4. Given that a similarity score of 1.0 indicates full alignment of requirements with the user profile and 0 indicates no semantic overlap, these results suggest that, the majority of LinkedIn's recommendations constitute weak matches for the hypothetical profile. However, as detailed insubsection 5.3.1,the 0.0–0.1 bucket likely includes cases where requirements extraction failed, resulting in empty extracted requirement and consequently near-zero similarity scores. Therefore, values in this range reflect a combination of genuine semantic mismatches and extraction failures.

As illustrated in Figure 5.8 only 6 of the 22 LinkedIn-recommended jobs were manually marked as suitable in the ground truth. Among these six, only a single posting `job6` obtained a comparatively high similarity score (≈ 0.43).

This leads to a critical consideration regarding the implementation of a fixed acceptance threshold as a filtering mechanism. If a strict threshold of 0.7 (implying a 70% semantic overlap) were applied, the results in Figure 5.8 demonstrate that no job postings would be retrieved. Even if the threshold were lowered to 0.4, only three postings

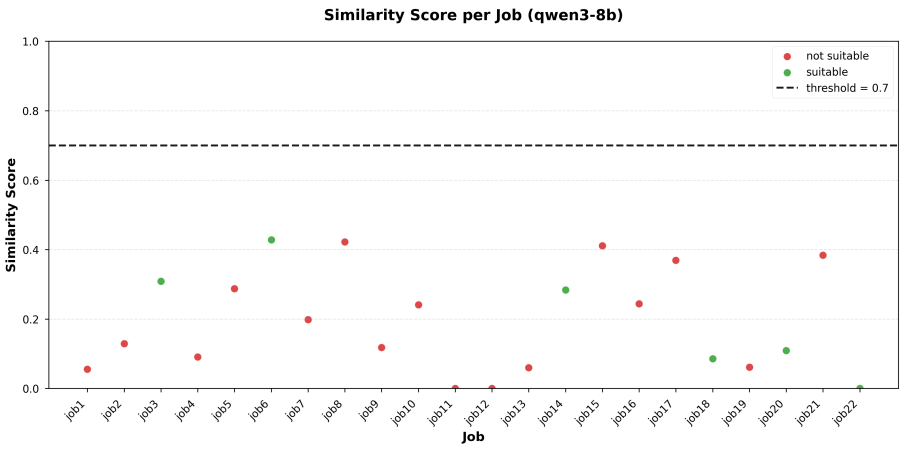

Figure 5.8: Similarity score per job

would qualify: `job6`, `job8`, and `job15`. While `job6` represents a valid match, `job8` and `job15` were manually annotated as unsuitable due to their employment type (specifically Ausbildung and Werkstudent).

Because the similarity search concentrates on skills and experience matching without explicitly filtering for employment type (e.g., full-time vs. internship), these false positives are an expected limitation. More concerning, however, is that five other job postings identified as suitable in the ground truth failed to reach the 0.4 threshold. This indicates a high rate of false negatives, suggesting that the system currently underestimates the relevance of certain valid job postings. This can be linked to Garbage-in-Garbage-out (GiGo) principle, which state that "good inputs generally result in good outputs, and bad inputs, barring design intervention, generally result in bad outputs" [LHB10]. In context of similarity search, the incomplete or incorrect extraction of requirements results a "bad input", leading to an artificially low similarity score. GiGo combined with lack of granularity in the user profile, which reflects the real-world challenge of users providing incomplete or non-exhaustive data, caused the false negatives.

6 Conclusion

This thesis presented a novel, open-source approach to job matching that addresses the semantic limitations of traditional keyword-based search engines and the opacity of industrial recommendation systems. By integrating LLM-based requirements extraction with an embedding-driven similarity search, the proposed system demonstrates a viable path toward more transparent and semantically grounded job retrieval.

Regarding the first research question (RQ1), the evaluation confirmed that open-source LLMs can effectively extract structured job requirements from unstructured text without the need for task-specific fine-tuning. Integration of the outlines library successfully solved the challenge of syntactic validity, ensuring machine-readable output. However, the results indicate that the system is not yet at a production-ready level. While Qwen3-8B emerged as the best-performing model, it only achieved an aggregated overall score of 0.50 out of 1.0. Furthermore, critical semantic metrics such as Completeness and Alignment remained below 0.50 (0.45 and 0.50 respectively). This highlights a significant gap: while lightweight open-source models can be constrained to produce valid JSON structure, they still struggle with the semantic reasoning required to capture all mandatory requirements accurately without missing context or miscategorizing terms.

Regarding the second research question (RQ2), the comparison against LinkedIn's recommendation engine showed that the proposed approach offers a more transparent and quantifiable matching logic. However, the system did not reach high similarity scores for the analyzed job postings. This low scoring can be linked to the weak requirement extraction process. As observed in the section 5.3, the models frequently hallucinated requirements or included artifacts from the prompt instructions. These false negatives in the extracted data created a mismatch against the user profile, artificially lowering the similarity scores even for relevant job postings. Consequently, while the architecture provides a solid theoretical foundation, the current extraction quality introduces noise that lowers the reliability of the matching score. Ultimately, this work contributes a modular, cost-effective architecture for job matching. By relying exclusively on lightweight open-source components, it democratizes access to advanced recruitment technologies, though significant improvements in extraction fidelity are required before it can compete with industrial-grade systems.

6.1 Discussion

This thesis set out to build a transparent, open-source job matching system. Throughout the conception and implementation, several key architectural decisions were made to address the gaps identified in the state of the art. This section summarizes those decisions and evaluates their effectiveness in light of the evaluation results.

6.2 Outlook

While the proposed system demonstrates the efficacy of this approach, several limitations identified during implementation and evaluation point toward avenues for future research and optimization

6.2.1 Integration of Reasoning and "Thinking" Models

The current implementation relied on standard instruction-tuned models (e.g.,mistral-7B-instruct, Llama-3.1-8B-Instruct). It did not utilize Chain-of-Thought (CoT) prompting or emerging "thinking" models that dedicate compute time to reasoning before generation. Future work should investigate if these architectures can improve the Completeness and Alignment scores by allowing the model to better distinguish between mandatory requirements and optional "nice-to-haves".

6.2.2 Scaling to Larger Models

The experiments were constrained to models with 7B–9B parameters to ensure consumer hardware compatibility. Given that extraction performance was the primary bottleneck, future research should evaluate larger open-source models (e.g., 70B+ parameters). These larger models typically exhibit superior instruction-following and semantic reasoning capabilities, which could significantly reduce the false positives and hallucinations observed in this study.

6.2.3 Performance and Latency Analysis

This thesis focused primarily on the accuracy and quality of extraction. The analysis of latency, to determine how fast LLM models extracted requirements, was out of scope of this work. For this system to be deployed as a real-time web service, a thorough performance benchmark is necessary, along with the exploration of optimization techniques such as quantization and batch processing.

6.2.4 Data Engineering and Imbalance

The data engineering process had its own limitations . Dataset contained more job postings from the technology industry than from all other industries combined. Future work

should expand the dataset to include a balanced representation of industries, ensuring the system generalizes beyond tech-centric job descriptions.

6.2.5 Explainability Features

Finally, a key advantage of this structured approach is its potential for transparency. Future development should implement an explainability layer that translates the mathematical similarity scores into natural language. This feature would show the user exactly why a job is relevant (e.g., "Matched based on your experience in Python and Project Management"), thereby bridging the gap between the numerical vector space and the user experience.

Bibliography

[AAR25] AJJAM, Mohammed-Hassan ; AL-RAWESHIDY, Hamed: *AI-Driven Semantic Similarity-Based Job Matching Framework for Recruitment Systems*. http://dx.doi.org/10.2139/ssrn.5293689. Version: 2025

[Ami25] AMIRHOSSEINI, Taha: *YourJobFinder*. https://github.com/whoistahito/YourJobFinder. Version: 2025. – Accessed: 2026-01-21

[AWT24] ALI, Ameen ; WOLF, Lior ; TITOV, Ivan: Mitigating Copy Bias in In-Context Learning through Neuron Pruning. In: *CoRR* abs/2410.01288 (2024). http://dx.doi.org/10.48550/ARXIV.2410.01288. – DOI 10.48550/ARXIV.2410.01288

[CJR+] COLVIN, Samuel ; JOLIBOIS, Eric ; RAMEZANI, Hasan ; GARCIA BADARACCO, Adrian ; DORSEY, Terrence ; MONTAGUE, David ; MATVEENKO, Serge ; TRYLESINSKI, Marcelo ; RUNKLE, Sydney ; HEWITT, David ; HALL, Alex ; PLOT, Victorien: *Pydantic Validation*. https://docs.pydantic.dev/latest/

[CN08] CHAN, Yee S. ; NG, Hwee T.: MAXSIM: A Maximum Similarity Metric for Machine Translation Evaluation. In: MCKEOWN, Kathleen R. (Hrsg.) ; MOORE, Johanna D. (Hrsg.) ; TEUFEL, Simone (Hrsg.) ; ALLAN, James (Hrsg.) ; FURUI, Sadaoki (Hrsg.): *ACL 2008, Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics, June 15-20, 2008, Columbus, Ohio, USA*, The Association for Computer Linguistics, 2008, 55–62

[DT25] DAPENA-TRETTER, Matthew: *python-markdownify*. https://github.com/matthewwithanm/python-markdownify. Version: 2025. – Accessed: 23-10-2025

[ES23] EHLINGER, Eugenia G. ; STEPHANY, Fabian: Skills or Degree? The Rise of Skill-Based Hiring for AI and Green Jobs. In: *CoRR* abs/2312.11942 (2023). http://dx.doi.org/10.48550/ARXIV.2312.11942. – DOI 10.48550/ARXIV.2312.11942

[GBC22] GNEHM, Ann-Sophie ; BÜHLMANN, Eva ; CLEMATIDE, Simon: Evaluation of Transfer Learning and Domain Adaptation for Analyzing German-Speaking Job Advertisements. In: CALZOLARI, Nicoletta (Hrsg.) ; BÉCHET,

Frédéric (Hrsg.) ; BLACHE, Philippe (Hrsg.) ; CHOUKRI, Khalid (Hrsg.) ; CIERI, Christopher (Hrsg.) ; DECLERCK, Thierry (Hrsg.) ; GOGGI, Sara (Hrsg.) ; ISAHARA, Hitoshi (Hrsg.) ; MAEGAARD, Bente (Hrsg.) ; MARIANI, Joseph (Hrsg.) ; MAZO, Hélène (Hrsg.) ; ODIJK, Jan (Hrsg.) ; PIPERIDIS, Stelios (Hrsg.): *Proceedings of the Thirteenth Language Resources and Evaluation Conference, LREC 2022, Marseille, France, 20-25 June 2022*, European Language Resources Association, 2022, 3892–3901

[GZX+24] GLM, Team ; ZENG, Aohan ; XU, Bin ; WANG, Bowen ; ZHANG, Chenhui ; YIN, Da ; ROJAS, Diego ; FENG, Guanyu ; ZHAO, Hanlin ; LAI, Hanyu ; YU, Hao ; WANG, Hongning ; SUN, Jiadai ; ZHANG, Jiajie ; CHENG, Jiale ; GUI, Jiayi ; TANG, Jie ; ZHANG, Jing ; LI, Juanzi ; ZHAO, Lei ; WU, Lindong ; ZHONG, Lucen ; LIU, Mingdao ; HUANG, Minlie ; ZHANG, Peng ; ZHENG, Qinkai ; LU, Rui ; DUAN, Shuaiqi ; ZHANG, Shudan ; CAO, Shulin ; YANG, Shuxun ; TAM, Weng L. ; ZHAO, Wenyi ; LIU, Xiao ; XIA, Xiao ; ZHANG, Xiaohan ; GU, Xiaotao ; LV, Xin ; LIU, Xinghan ; LIU, Xinyi ; YANG, Xinyue ; SONG, Xixuan ; ZHANG, Xunkai ; AN, Yifan ; XU, Yifan ; NIU, Yilin ; YANG, Yuantao ; LI, Yueyan ; BAI, Yushi ; DONG, Yuxiao ; QI, Zehan ; WANG, Zhaoyu ; YANG, Zhen ; DU, Zhengxiao ; HOU, Zhenyu ; WANG, Zihan: *ChatGLM: A Family of Large Language Models from GLM-130B to GLM-4 All Tools*. http://dx.doi.org/10.48550/ARXIV.2406.12793. Version: 2024

[HEM+25] HOWISON, Mark ; ENSOR, William O. ; MAHARJAN, Suraj ; PARIKH, Rahil ; SENGAMEDU, Srinivasan H. ; DANIELS, Paul ; GAITHER, Amber ; YEATS, Carrie ; REDDY, Chandan K. ; HASTINGS, Justine S.: Extracting Structured Labor Market Information from Job Postings with Generative AI. In: *Digit. Gov.: Res. Pract.* 6 (2025), Februar, Nr. 1. http://dx.doi.org/10.1145/3674847. – DOI 10.1145/3674847

[HLL+24] HERANDI, Amirhossein ; LI, Yitao ; LIU, Zhanlin ; HU, Ximin ; CAI, Xiao: *Skill-LLM: Repurposing General-Purpose LLMs for Skill Extraction*. https://arxiv.org/abs/2410.12052. Version: 2024

[Huy25] HUYEN, Chip: *AI Engineering*. USA : O'Reilly Media, 2025. – ISBN 978–1801819312

[HYL17] HAMILTON, William L. ; YING, Zhitao ; LESKOVEC, Jure: Inductive Representation Learning on Large Graphs. In: GUYON, Isabelle (Hrsg.) ; LUXBURG, Ulrike von (Hrsg.) ; BENGIO, Samy (Hrsg.) ; WALLACH, Hanna M. (Hrsg.) ; FERGUS, Rob (Hrsg.) ; VISHWANATHAN, S. V. N. (Hrsg.) ; GARNETT, Roman (Hrsg.): *Advances in Neural Information Processing Sys-*

tems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, 2017, 1024–1034

[JA24] JAY ALAMMAR, Maarten G.: *Hands-On Large Language Models*. Beijing : O'Reilly Media, Inc, 2024. – OCLC: 1459898028

[JSM+23] JIANG, Albert Q. ; SABLAYROLLES, Alexandre ; MENSCH, Arthur ; BAMFORD, Chris ; CHAPLOT, Devendra S. ; LAS CASAS, Diego de ; BRESSAND, Florian ; LENGYEL, Gianna ; LAMPLE, Guillaume ; SAULNIER, Lucile ; LAVAUD, Lélio Renard ; LACHAUX, Marie-Anne ; STOCK, Pierre ; SCAO, Teven L. ; LAVRIL, Thibaut ; WANG, Thomas ; LACROIX, Timothée ; SAYED, William E.: Mistral 7B. In: *CoRR* abs/2310.06825 (2023). http://dx.doi.org/10.48550/ARXIV.2310.06825. – DOI 10.48550/ARXIV.2310.06825

[KAKAH25] KABIR, Md A. ; ABDELFATAH, Kareem ; KORAYEM, Mohammed ; AL HASAN, Mohammad: Ordinal Regression for Job Search Keyword Similarity Prediction. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Pennsylvania, USA : Association for the Advancement of Artificial Intelligence (AAAI), 2025, S. 1–5. – https://compjobs.github.io/assets/paper_202509.pdf

[KLB+24] KUREK, Jarosław ; LATKOWSKI, Tomasz ; BUKOWSKI, Michał ; ŚWIDERSKI, Bartosz ; ŁĘPICKI, Mateusz ; BARANIK, Grzegorz ; NOWAK, Bogusz ; ZAKOWICZ, Robert ; DOBRAKOWSKI, Łukasz: Zero-Shot Recommendation AI Models for Efficient Job–Candidate Matching in Recruitment Process. In: *Applied Sciences* 14 (2024), Nr. 6. http://dx.doi.org/10.3390/app14062601. – DOI 10.3390/app14062601. – ISSN 2076–3417

[LHB10] LIDWELL, W. ; HOLDEN, K. ; BUTLER, J.: *Universal Principles of Design, Revised and Updated: 125 Ways to Enhance Usability, Influence Perception, Increase Appeal, Make Better Design Decisions, and Teach through Design*. Rockport Publishers, 2010 https://books.google.de/books?id=l0QPECGQySYC. – ISBN 9781610580656

[Lin25] LINKEDIN: *Jobs recommended for you*. https://www.linkedin.com/help/linkedin/answer/a512279. Version: 2025. – Accessed: 2026-01-21

[LIX+23] LIU, Yang ; ITER, Dan ; XU, Yichong ; WANG, Shuohang ; XU, Ruochen ; ZHU, Chenguang: G-Eval: NLG Evaluation using Gpt-4 with Better Human Alignment. In: BOUAMOR, Houda (Hrsg.) ; PINO, Juan (Hrsg.) ; BALI, Kalika (Hrsg.): *Proceedings of the 2023 Conference on Empirical Methods in Nat-*

ural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023, Association for Computational Linguistics, 2023, 2511–2522

[LWH+24] LIU, Ping ; WEI, Haichao ; HOU, Xiaochen ; SHEN, Jianqiang ; HE, Shihai ; SHEN, Kay Q. ; CHEN, Zhujun ; BORISYUK, Fedor ; HEWLETT, Daniel ; WU, Liang ; VEERARAGHAVAN, Srikant ; TSUN, Alex ; JIANG, Chengming ; ZHANG, Wenjing: *LinkSAGE: Optimizing Job Matching Using Graph Neural Networks*. https://arxiv.org/abs/2402.13430. Version: 2024

[MLK+24] MASHAYEKHI, Yoosof ; LI, Nan ; KANG, Bo ; LIJFFIJT, Jefrey ; BIE, Tijl D.: A Challenge-based Survey of E-recruitment Recommendation Systems. In: *ACM Comput. Surv.* 56 (2024), Nr. 10, 252. http://dx.doi.org/10.1145/3659942. – DOI 10.1145/3659942

[Mus23] MUSAZADE, Nurlan: Tools and Technologies Utilized in Data-Related Positions: An Empirical Study of Job Advertisements, University of Maribor, University Press, 12 2023 (36th Bled eConference: Digital Economy and Society: The Balancing Act for Digital Innovation in Times of Instability, BLED 2023 - Proceedings), 155–170. – 36th Bled econfrence ; Conference date: 25-06-2023 Through 28-08-2023

[NZMB24] NGUYEN, Khanh C. ; ZHANG, Mike ; MONTARIOL, Syrielle ; BOSSELUT, Antoine: Rethinking Skill Extraction in the Job Market Domain using Large Language Models. In: *CoRR* abs/2402.03832 (2024). http://dx.doi.org/10.48550/ARXIV.2402.03832. – DOI 10.48550/ARXIV.2402.03832

[Pin25] PINECONE: *Chunking Strategies for LLM Applications*. https://www.pinecone.io/learn/chunking-strategies/. Version: 2025. – Accessed: 2025-01-12

[PSM+25] POURBABAEE, Farzad ; SHENG, Sophie Y. ; MCCRORY, Peter ; SIMON, Luke ; MO, Di: Trading off Relevance and Revenue in the Jobs Marketplace: Estimation, Optimization and Auction Design. In: *CoRR* abs/2504.03618 (2025). http://dx.doi.org/10.48550/ARXIV.2504.03618. – DOI 10.48550/ARXIV.2504.03618

[Qwe25] QWEN TEAM: *Qwen3 Technical Report*. https://arxiv.org/abs/2505.09388. Version: 2025

[Ras24] RASCHKA, Sebastian: *Build a Large Language Model (From Scratch)*. Shelter Island, NY : Manning Publications, 2024 https://www.manning.com/books/build-a-large-language-model-from-scratch. – ISBN 978–1633437166

[RG19] REIMERS, Nils ; GUREVYCH, Iryna: Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In: INUI, Kentaro (Hrsg.) ; JIANG, Jing (Hrsg.) ; NG, Vincent (Hrsg.) ; WAN, Xiaojun (Hrsg.): *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, Association for Computational Linguistics, 2019, 3980–3990

[RWW+25] ROSENBERGER, Julian ; WOLFRUM, Lukas ; WEINZIERL, Sven ; KRAUS, Mathias ; ZSCHECH, Patrick: CareerBERT: Matching resumes to ESCO jobs in a shared embedding space for generic job recommendations. In: *Expert Syst. Appl.* 275 (2025), 127043. http://dx.doi.org/10.1016/J.ESWA.2025.127043. – DOI 10.1016/J.ESWA.2025.127043

[Sip96] SIPSER, Michael: Introduction to the Theory of Computation. In: *SIGACT News* 27 (1996), Nr. 1, 27–29. http://dx.doi.org/10.1145/230514.571645. – DOI 10.1145/230514.571645

[Sou24] SOUZA, Tharsis T. P.: *Taming LLMs: A Practical Guide to LLM Pitfalls with Open Source Software*. https://github.com/souzatharsis/tamingLLMs. Version: 2024

[SPS+24] SHORTEN, Connor ; PIERSE, Charles ; SMITH, Thomas B. ; CARDENAS, Erika ; SHARMA, Akanksha ; TRENGROVE, John ; LUIJT, Bob van: StructuredRAG: JSON Response Formatting with Large Language Models. In: *CoRR* abs/2408.11061 (2024). http://dx.doi.org/10.48550/ARXIV.2408.11061. – DOI 10.48550/ARXIV.2408.11061

[WDN+24] WAGNER, Nico ; DESMOND, Michael ; NAIR, Rahul ; ASHKTORAB, Zahra ; DALY, Elizabeth M. ; PAN, Qian ; COOPER, Martin S. ; JOHNSON, James M. ; GEYER, Werner: Black-box Uncertainty Quantification Method for LLM-as-a-Judge. In: *CoRR* abs/2410.11594 (2024). http://dx.doi.org/10.48550/ARXIV.2410.11594. – DOI 10.48550/ARXIV.2410.11594

[WKN+25] WEERAWARDHENA, Sajana ; KASSIANIK, Paul ; NELSON, Blaine ; SAGLAM, Baturay ; VELLORE, Anu ; PRIYANSHU, Aman ; VIJAY, Supriti ; AUFIERO, Massimo ; GOLDBLATT, Arthur ; BURCH, Fraser ; LI, Ed ; HE, Jianliang ; KEDIA, Dhruv ; OSHIBA, Kojin ; YANG, Zhouran ; SINGER, Yaron ; KARBASI, Amin: Llama-3.1-FoundationAI-SecurityLLM-8B-Instruct Technical Report. In: *CoRR* abs/2508.01059 (2025). http://dx.doi.org/10.48550/ARXIV.2508.01059. – DOI 10.48550/ARXIV.2508.01059

[WKS25] WOMARK, Gabriel ; KHARKAR, Ritvik ; SHRIVASTRAVA, Ishan: Lessons Learned — Building ML Models to Remove Irrelevant Results in Job Search.

In: *Proceedings of the AAAI International Workshop on Computational Jobs Marketplace* Ziprecruiter, Inc., 2025, S. 1–5. – https://compjobs.github.io/assets/paper_202505.pdf

[WL23] WILLARD, Brandon T. ; LOUF, Rémi: Efficient Guided Generation for Large Language Models. In: *CoRR* abs/2307.09702 (2023). http://dx.doi.org/10.48550/ARXIV.2307.09702. – DOI 10.48550/ARXIV.2307.09702

[YLD+24] YANG, Yixin ; LI, Zheng ; DONG, Qingxiu ; XIA, Heming ; SUI, Zhifang: Can Large Multimodal Models Uncover Deep Semantics Behind Images? In: KU, Lun-Wei (Hrsg.) ; MARTINS, Andre (Hrsg.) ; SRIKUMAR, Vivek (Hrsg.): *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, Association for Computational Linguistics, 2024, 1898–1912

[YRZ+24] YUAN, Dong ; RASTOGI, Eti ; ZHAO, Fen ; GOYAL, Sagar ; NAIK, Gautam ; RAJAGOPAL, Sree P.: Evaluate Summarization in Fine-Granularity: Auto Evaluation with LLM. In: *CoRR* abs/2412.19906 (2024). http://dx.doi.org/10.48550/ARXIV.2412.19906. – DOI 10.48550/ARXIV.2412.19906

[YXX+25] YU, Xiao ; XU, Ruize ; XUE, Chengyuan ; ZHANG, Jinzhong ; MA, Xu ; YU, Zhou: *ConFit v2: Improving Resume-Job Matching using Hypothetical Resume Embedding and Runner-Up Hard-Negative Mining*. https://arxiv.org/abs/2502.12361. Version: 2025

[ZCS+23] ZHENG, Lianmin ; CHIANG, Wei-Lin ; SHENG, Ying ; ZHUANG, Siyuan ; WU, Zhanghao ; ZHUANG, Yonghao ; LIN, Zi ; LI, Zhuohan ; LI, Dacheng ; XING, Eric P. ; ZHANG, Hao ; GONZALEZ, Joseph E. ; STOICA, Ion: Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In: OH, Alice (Hrsg.) ; NAUMANN, Tristan (Hrsg.) ; GLOBERSON, Amir (Hrsg.) ; SAENKO, Kate (Hrsg.) ; HARDT, Moritz (Hrsg.) ; LEVINE, Sergey (Hrsg.): *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023

[ZWS+21] ZHAO, Jing ; WANG, Jingya ; SIGDEL, Madhav ; ZHANG, Bopeng ; HOANG, Phuong ; LIU, Mengshu ; KORAYEM, Mohammed: *Embedding-based Recommender System for Job to Candidate Matching on Scale*. https://arxiv.org/abs/2107.00221. Version: 2021