The Evolving Role of AI Specialists in Academia: A Job Market Analysis

by

Santiago Dominguez Ham

Bachelor Thesis in Computer Science

Submission: May 19, 2025

Supervisor: Prof. Dr. I. Frumin

Constructor University | School of Computer Science and Engineering

Statutory Declaration

Family Name, Given/First Name	Dominguez Ham, Santiago
Matriculation number	30006130
Kind of thesis submitted	Bachelor Thesis

English: Declaration of Authorship

I hereby declare that the thesis submitted was created and written solely by myself without any external support. Any sources, direct or indirect, are marked as such. I am aware of the fact that the contents of the thesis in digital form may be revised with regard to usage of unauthorized aid as well as whether the whole or parts of it may be identified as plagiarism. I do agree my work to be entered into a database for it to be compared with existing sources, where it will remain in order to enable further comparisons with future theses. This does not grant any rights of reproduction and usage, however.

This document was neither presented to any other examination board nor has it been published.

German: Erklärung der Autorenschaft (Urheberschaft)

Ich erkläre hiermit, dass die vorliegende Arbeit ohne fremde Hilfe ausschließlich von mir erstellt und geschrieben worden ist. Jedwede verwendeten Quellen, direkter oder indirekter Art, sind als solche kenntlich gemacht worden. Mir ist die Tatsache bewusst, dass der Inhalt der Thesis in digitaler Form geprüft werden kann im Hinblick darauf, ob es sich ganz oder in Teilen um ein Plagiat handelt. Ich bin damit einverstanden, dass meine Arbeit in einer Datenbank eingegeben werden kann, um mit bereits bestehenden Quellen verglichen zu werden und dort auch verbleibt, um mit zukünftigen Arbeiten verglichen werden zu können. Dies berechtigt jedoch nicht zur Verwendung oder Vervielfältigung.

Diese Arbeit wurde noch keiner anderen Prüfungsbehörde vorgelegt noch wurde sie bisher veröffentlicht.

May 19, 2025 [illegible]
Date, Signature

Abstract

This study emerges from a desire to further understand which academic fields and disciplines are seeking the largest amount of AI-related talent. In response to a lack of research focusing exclusively on academia, this study seeks to uncover trends specifically within a doctoral/postdoctoral context. Web scraping, the process of automatically extracting data from websites by parsing their underlying code structure, was used to collect data from two academic job posting websites: FindAPhD and AcademicPositions. The data from a total of 1,819 postings was subjected to a quantitative analysis that revealed patterns and trends in geographic distribution, departmental affiliation, skill demands, posting length, posting time, and funding type, accounting for different structural and regional paradigms between the two websites. A novel LLM-based approach was used to generate additional semantic attributes, uncovering more nuanced metrics and trends to further enrich the dataset. Findings identify a strong institutional interest for interdisciplinary research, painting a clearer picture of the broadening role of AI in academia, as well as offering actionable insights for institutions, policymakers, and individuals alike.

Contents

1 Introduction

The increasing presence of Artificial Intelligence (AI) in professional environments over the last few years has been matched by a growing demand for AI specialists who can connect their technical expertise in this relatively novel topic with other discipline-specific knowledge. This is not only revolutionizing industry and workforce, but also transforming the landscape of research and academia. Progressively more sophisticated AI technologies and services have enabled new collaborative approaches that challenge long-established disciplinary boundaries, reflecting a broader trend towards interdisciplinarity. To derive meaningful insights into what skills and dexterities are most in demand in the current job market, and which fields and institutions are leading this integration, it is imperative to understand where these AI specialists are being hired.

Existing research mainly concentrates on AI skill demands in industry, leaving trends explicitly related to academia comparatively underexplored [8]. This study aims to address this gap by employing web-scraping techniques to analyze job postings for PhD and Post-PhD positions on two websites: FindAPhd and AcademicPositions. As such, this research is guided by—and aims to answer—the following questions:

1. Which fields or domains are looking to hire the most significant number of AI specialists?
2. What skills are most in demand for AI-related doctoral/postdoctoral positions in research and academia?

Precursory research and literature review resulted in two hypotheses, respectively. First, fields closely related to computer science, such as data science and engineering, are expected to have the highest demand for AI specialists. Second, machine learning, data processing, statistics, and programming are hypothesized to be the most frequently requested skills.

Some cornerstone concepts valuable to understanding the motivation behind this research and interpreting the results of experimentation include AI Specialist, interdisciplinarity, and job market analysis. In this study, **AI Specialist** refers to a professional with expertise in fields comprising the broader notion of "AI," such as machine learning and natural language processing, often working on designing and developing new technologies or implementing existing ones. This ties to the idea of **interdisciplinarity**, which refers to the invocation of knowledge from different disciplines as a holistic approach to research or team-building, to ultimately facilitate solving problems in nature, scale, and scope similar to those AI aims to tackle. The **job market analysis**, also the primary research deliverable, involves methodically examining collected data to uncover patterns in hiring practices.

This research employs a quantitative approach for analysing academic job postings. Findings are expected to deepen understanding of how academia is accommodating AI, providing actionable insights to institutions, policymakers, and individuals alike; for example, enabling academic institutions to integrate AI into their research teams more effectively, as well as the researchers comprising these teams to align their skill sets with current academic demands.

2 Statement and Motivation of Research + Literature Review

2.1 Why Web Scraping?

Despite the increasingly dominant presence of AI in academia, the understanding of how this is changing the doctoral and postdoctoral job markets remains limited. This, in turn, served as motivation for the establishment of the two aforementioned research questions. The internet is already home to a myriad of information about employment opportunities, amassed in numerous job posting websites targeted at distinct audiences. However, to derive meaningful and generalizable insights about trends, extremely large volumes of data are required—manually collecting this information would take an unreasonable amount of time. Web scraping, the process of automatically extracting data from websites by parsing the underlying HTML, immediately jumps as an attractive alternative.

The practice of web scraping has existed for nearly as long as the World Wide Web itself, but has only come to be seen as a truly powerful data collection tool in recent years. This is largely due to streamlined web scraping libraries for programming languages such as Python becoming widely available. The idea of scraping job posting websites is also not novel, as these sites have also had a consistent presence on the internet for decades; researchers and analysts alike have recognized the potential of job postings as a rich source of information and have conducted studies delving into the creation of procedures for transforming job posting website information into valuable intelligence, including data collected specifically via web scraping. Scraping techniques have proven effective at collecting data from multiple job posting portals for industry positions, further validating the method considered for this research [8]. Hence, the relevance of this study is reinforced, given that the application of these methods to analyze academic job markets remains meagre. Other studies have stressed the importance of careful input data sanitation, structuring, and standardizing to produce high-quality analysis and use Python libraries to their full potential [12].

2.2 Implementations of Web Scraping

This is not to say, however, that Python is the only language with web scraping capabilities; it is merely the most accessible. For instance, the effectiveness of the R programming language (specifically employing the `rvest` library) has already been thoroughly explored within the context of medicinal informatics, with web scraping as a primary collection method, followed by tokenization, lemmatization, stemming (these three are often collectively referred to as *Natural Language Processing*, or NLP for short), and term frequency-inverse document frequency to effectively categorize collected data [10]. Above all, the importance of sanitizing collected data was further emphasized: the volume of scraped information was reduced by 73% after removing duplicate data. In such studies, great success has been found by employing keywords corresponding to key skills and attributes as means of categorizing domain-specific skills, and has been used as a method for enabling NLP to analyze skill demands in postdoctoral industry job postings [14]. Naturally, these practices for handling data, independently of programming language, are extremely valuable insights. A significant hurdle was the confounding qualities of a bilingual job posting website, something that must be kept in mind when choosing a target to scrape, given that lemmatization of languages other than English is not as strictly standardized. Other detrimental factors in output quality stemmed from the small amount of time allocated for data collection and the small number of websites used for scraping. Being aware of limitations and challenges faced by other studies must be kept in mind when developing further research.

Three libraries of particular noteworthiness, `BeautifulSoup`, `Scrapy`, and `Selenium`, are designed specifically for the Python programming language, which already sees widespread use in fields like data science and machine learning. These libraries achieve a similar purpose but differ in their capabilities, and thus must be further scrutinized before any methodology is decided on. Dedicated research has resulted in general recommendations for future studies: `BeautifulSoup` is the most adept at carrying out minor exercises, and `Scrapy` is deemed the overall best due to its scalability and performance. `Selenium`, meanwhile, excels in its capabilities for scraping website content loaded dynamically via methods such as JavaScript [6].

Primary observations of the libraries' respective documentations unveil the reasoning behind these recommendations. `BeautifulSoup` is often favored by beginners due to its overall ease of use and efficiency at parsing simple HTML, but its capabilities do not allow for intuitive scraping of websites that load content dynamically [9]. `Scrapy` is specifically designed to handle more complex and large-scale projects that incorporate asynchronous requests into their scraping or attempt to scrape multiple websites simultaneously. These capabilities make `Scrapy` an attractive choice for many, but can be immoderate for the focus of this study due to the comparatively unassuming volume of data contained in job posting websites. Finally, `Selenium` distinguishes itself by being able to scrape content not contained within raw HTML; that is, information that requires input, like scrolling or clicking to access [11]. These observations concur with the aforementioned recommendations.

2.3 Interdisciplinarity, Challenges, and Concerns

AI's multifaceted nature and versatility grant it unparalleled applicability in a wide range of fields, fostering the integration of AI specialists that this study aims to investigate. However, studies have identified significant challenges within this interdisciplinary potential. Chief among them is the difficulty of AI expertise integration into fields distant from computer science, an unfortunate result of a communication gap between AI specialists and experts in other disciplines. Healthcare practitioners, for instance, have faced great difficulties aligning technical AI concepts with desired practical applications [1]. This phenomenon occurs due to differences in technical expertise. Fields and disciplines closely related to computer science can integrate AI more seamlessly on account of their similar tools and methodologies. More distant fields find AI integration to be a more formidable challenge due to varying levels of technical literacy and research paradigms. This finding is particularly pertinent to the first research question, as understanding these challenges is essential for holistic interpretation of the findings and making sure the presented insights are actionable.

In tandem to web scraping, Large Language Models (LLMs)—arguably the quintessential building blocks of present-day generative AI—have brought profound change to data collection and analysis methods, posing both favorable and obstructive prospects. On one hand, LLMs promote inclusion of online collaborators, accelerate knowledge generation, excel at mediating between humans, and effectively summarize information; on the other, they deter consumers from engaging with source material, cultivate pluralistic ignorance, homogenize individuals' private beliefs, and often aid in the dispersal of false information [2]. Insights from this research bring attention to certain specific risks of integrating machine learning technologies into research; risks which, without a doubt, influence hiring and research team formation practices in academia, something indispensable to keep in mind for this research. As such, the importance of comprehensive training and institutional support as a means of bridging these gaps in communication and technical proficiency cannot be overstated. Research suggests that cross-disciplinary training initiatives play a crucial role in promoting collaboration [5]. The lack of mutual understanding of shared frameworks, or a shared language altogether, is a serious hindrance for interdisciplinary AI projects. Such insights assist in the establishment of a framework for understanding how academic institutions can better integrate AI. These aforesaid challenges remain daunting obstacles and pose great relevance to this study as they are a key part of the conversation to which contributions wish to be made.

Several ethical concerns and debates surround web scraping, given its status as a uniquely powerful and scalable method of data collection, further exacerbated by its widespread use. These predominantly pertain to adherence to terms of service and user privacy guidelines; naturally, complying with these is essential to avoid ethical violations with potentially serious legal repercussions. A large number of websites outright forbid the use of web crawlers and spiderbots, most often as a means of preventing private information from being collected without consent. A detailed breakdown of what is and isn't allowed can be found on a website's "robots.txt" file [4]. Adhering to this is perchance the single most important way to minimize the risk of ethical infractions, harming website functionality, and overloading server capacities. For this study, an even higher level of caution is required, given that job posting websites can contain non-public personal information. When researching, transparency and caution are vital to ensure only public data is scraped through interactions allowed by the service providers. Incorporating these principles of responsible research into the design of this study ensures compliance with ethical standards and can aid in developing standardized practices for web scraping research in all fields.

To summarize, this study aims to confront a significant gap in the literature by investigating the demand for AI specialists in doctoral and postdoctoral academic job markets, thus extending the state of the art by providing relevant insights into AI's changing role in academia. Prior studies primarily focus on the industry job market, leaving academic job positions mostly unexplored or under-analyzed. Web scraping as a method of collecting information has likewise already been used for job market analysis, but has yet to find more widespread use in academia in the context of AI-related job positions. The results and findings of this research possess great utility and practical implications for many. Academic institutions, primarily universities, can use the insights to better inform their curriculum development, ensuring that their pupils' education remains competitive and can continue to prepare them for careers in research and academia in the current job market landscape. The same holds for developing more informed procedures for research group composition. Policymakers, both in academia and otherwise, can also harness this knowledge to promote interdisciplinary initiatives that adopt AI more efficiently. Finally, the results of the research can provide invaluable guidance to individual students, researchers, or professionals wishing to better tailor their skill development and career planning. Naturally, a study such as this is in itself a valuable contribution to the broader conversation around AI in academia.

3 Description of the Investigation / Methodology

A quantitative approach was espoused to answer the research questions and deemed appropriate given their multifaceted essence. This quantitative study leverages web scraping techniques to collect large volumes of data from two academic job posting websites: **FindAPhD** and **AcademicPositions**. The websites were chosen as scraping targets due to their appealing combination of fittingness to the study's research aims (a job posting website geared to doctoral and postdoctoral applicants) and ease of scraping. This means that they do not rely excessively on dynamic JavaScript content. The two websites also had a similar offer of core metadata for postings, particularly fit for identifying trends and highlights through frequency analysis of scraped features like skills, fields, and location. Data exclusive to FindAPhD included department, supervisor, and funding information; data exclusive to AcademicPositions included posting date.

3.1 Choice of Programming Language, Libraries, and Data Format

To carry out the scraping, Python was chosen as a programming language due to its accessibility, ease of use, the availability of web scraping libraries that proved helpful in other studies (`BeautifulSoup` and `Selenium`), and the availability of data analysis and visualization libraries (`Pandas, Matplotlib, Seaborn`) [7], [3], [13]. Although `Scrapy` is another widely used web scraping library, the combination of `BeautifulSoup` and `Selenium` was deemed more favorable because `Selenium` has better capabilities for handling JavaScript, which both websites use to load content dynamically.

All these tools' capabilities are adequate for handling the work and challenges of this study, allowing for the scraping of dynamic and static content and processing both structured and unstructured text. The following is a high-level overview of the steps involved in the scraping of each website:

1. Before writing or testing any scraping scripts, the HTML structure of the website was carefully inspected via the "inspect" tool characteristic of most modern web browsers (Google Chrome was utilized for this experiment). This is an essential step, as an effective web scraper cannot be designed without knowing exactly what HTML attributes it should be looking for; failure to do so will either make scraping impossible or drastically increase the number of necessary subsequent data cleaning operations. Collecting the necessary data without any unnecessary requests is good coding practice.

2. A Python script was written for each website, leveraging the `BeautifulSoup` and `Selenium` libraries. These scripts were first tested on a single page, followed by a small quantity of pages. This was done to ensure accuracy.

3. Each script's capabilities were expanded to account for pagination, enabling it to scrape all results for a given search term in one go.

4. Extracted data was stored in a CSV file, ensuring a consistent format for all: each row in the file representing a job posting, with columns designating the desired attributes.

Mindfulness towards important ethical concerns often raised by web scraping was upheld throughout the study. At all moments, the guidelines and terms of use established in the websites' robots.txt files were abided by. Considerable time (a random amount between

3 and 7 seconds) between server requests was enforced to respect network traffic and avoid overloading servers. Finally, only publicly available information was subjected to the scraping process; any non-public information that could identify a person was neither scraped nor stored. The results of this study are not being sold or used for profit in any form.

3.2 Formulation of Search Term and Skill Lists

This investigation's overall design and workflow were crafted around the research questions. Consequently, the process of gathering data commenced with establishing two crucial lists: one consisting of desired search terms or keywords, and another of the "AI Skills" the web scrapers are programmed to search for. The first list of AI-related searches was derived from self-sourced terms, terms from the literature review, terms from peer review feedback, and additional suggestions by DeepSeek R1. The complete search term list is the following, with synonyms grouped together and DeepSeek's contributions italicized. The prompt used to obtain them is listed in Appendix Listing 19.

- artificial intelligence, AI
- machine learning, deep learning, neural network
- LLM, large language model
- VLM
- GPT
- generative AI, *generative model*, genAI
- transformer model, pre-trained model
- reinforcement learning, supervised learning, unsupervised learning, *statistical learning*, few-shot learning, *zero-shot learning*
- AI agent
- natural language processing
- computer vision
- *computational intelligence*
- *expert system*

The second list, consisting of AI-related skill keywords, is particularly pertinent to answering the second research question. It was obtained in a similar manner to the first list. It is categorized and summarized below. The complete list of keywords can be found in Appendix Table 5, and the prompt used to generate some terms in Appendix Listing 20.

3.3 Design of Web Scraper and Data Cleaning

Following the list preparations, two Python scripts (one per website) were designed to carry out the web scraping in a manner as straightforward as possible. Upon execution, the user is prompted to enter keywords they wish to use as a search query. Then, `Selenium` is used to initialize a Chrome web driver, appending the search parameters to the scraped website's base URL. From this initial results page, valuable information

Category	**Sample Skills**
Programming + Libraries	Python, R, MATLAB, Rust, tensorFlow, etc.
Core Ideas	Machine learning, deep learning, supervised/unsupervised learning, etc.
Data Science + Engineering	Data analysis, data mining, data visualization, etc.
Math + Theory	Statistics, Bayesian methods, optimization, information theory, etc.
NLP + Computer Vision	Natural language processing, speech recognition, sentiment analysis, etc.
Deployment + Production	MLOps, Hugging Face, AWS, Docker, Kubernetes, etc.
Soft Skills	Research, algorithm design, peer review

Table 1: Examples of AI Skills for Each Category

is already extracted, namely the countries and disciplines with postings available for the search. The crawler then accesses a user-specified amount of individual job posting pages (the maximum number of pages must be checked manually) where the complete list of tags and posting descriptions are contained. To obtain the top skills for each posting, a “relevance score” is calculated using a weighted approach for every skill in the “AI Skills” dictionary, awarding one point for each mention in the posting’s description and two points for each mention in the title and tags.

```
skillScores = defaultdict(int)
    for skill in self.aiSkills:
        displaySkill = skill.replace("_programming", "")

# Score 2 points per title mention
titleMatches = self.countSkillOccurrences(result['Title'].lower(),
↪   skill)
skillScores[displaySkill] += titleMatches * 2

# Score 2 points per tag mention
tagMatches = sum(self.countSkillOccurrences(tag.lower(), skill) for
↪   tag in tags)
skillScores[displaySkill] += tagMatches * 2

# Score 1 point per description mention
descMatches = self.countSkillOccurrences(fullDescription.lower(),
↪   skill)
skillScores[displaySkill] += descMatches
```

Listing 1: Skill Scoring

The scraper was run once for each of the search terms, per website. For each search, the collected data was organized in two CSV files: an individual file for each search containing the all the information scraped from a posting; and a general search log file containing the timestamp, keywords, available countries, disciplines with most postings, and skills with the largest relevance score of each search. A line of code can be uncommented to enable “headless mode”, meaning that the script’s functions will be carried out without opening a separate Chrome window. While cleaner overall, headless mode sometimes causes bot-detection mechanisms to be triggered more easily. Complete scraper code fragments for both websites is found in Appendix Listings 7 - 13. Functions with similar functionality have different structures to accommodate for differences in HTML code between the two websites.

```python
def __init__(self):
    options = webdriver.ChromeOptions()

    ↪  options.add_argument('--disable-blink-features=AutomationControlled')
    # options.add_argument('--headless=new') # Un-commenting enables
    ↪  headless mode
    self.driver = webdriver.Chrome(
        service=Service(ChromeDriverManager().install()),
        options=options
    )
    self.baseUrl = "https://academicpositions.com"
```

Listing 2: Example `Selenium` WebDriver Setup, for AcademicPositions

The collected data was subjected to a “preprocessing” stage consisting of validation, cleaning, and enhancement. This was meant to eliminate—or at the very least, substantially mitigate—the impact of incomplete or inconsistent data on subsequent data analysis and statistical calculations. The `Pandas` library for Python, a data manipulation and analysis library already widely used in machine learning, contains methods apt for handling large volumes of data in CSV files and a handful of practical methods for handling missing data gracefully (namely, `fillna()`). The first preprocessing step was concatenating all CSV search results files into one large CSV file for each website to work with as few DataFrames as possible. Then, read through the two DataFrames and eliminate duplicate postings, adding a new column to track which search terms returned a posting.

```
for file in allFiles:
    # Get search term from filename
    baseName = os.path.basename(file)
    searchTerm = os.path.splitext(baseName)[0].replace("_", " ")

    df = pd.read_csv(file) # Load CSV
    df['Search Term'] = searchTerm # Add search term column
    dataframes.append(df)

 # Combine CSVs
 combinedDf = pd.concat(dataframes, ignore_index=True)
 combinedDf = combinedDf.groupby('Title', as_index=False).agg(
 lambda x: x.iloc[0] if x.name != 'Search Term' else set().union(*x)
 )
```

Listing 3: Concatenation of Scraping Results

3.4 Enrichment of Data Via LLM

To further enrich the data before any further cleanup operations and descriptive statistical calculations take place, a Large Language Model's (LLM) capabilities were leveraged to derive additional features and insights that simple keyword matching might not be able to capture. The LLM of choice was Mistral 7B, an open-source, pre-trained transformer model that can be run locally. Its lightweight resource requirements and multilingual capabilities made it an ideal candidate for this study's scope. Although the model runs completely offline, this did not influence the quality or usability of the output. In this use case, LLM creativity is secondary to raw reasoning abilities. Using Python, all posting descriptions for both websites were sequentially fed into Mistral, alongside a prompt asking it to carry out the following two tasks:

a) Classify a posting's "focus" as either involving the development of new AI technologies, the implementation and application of existing ones, both, or neither.

b) Formulate a list of 3-7 short tags that capture the topics, methods, and application domains of the posting.

The prompt included instructions for encoding responses in JSON format to facilitate parsing; it can be found in full alongside the Python code used to feed it to Mistral in Appendix Listing 15. With the resources and equipment used in this study (listed under Appendix Table 6), processing all 1,819 postings took approximately four hours. Using `Pandas`, the previously cleaned-up CSV file was loaded into a DataFrame, to which the focus, LLM tags, and raw JSON response (deemed worthy of keeping around in case of parsing errors) were appended. This DataFrame was again converted into a CSV file for data persistence and subjected to a few more sanitization steps.

Missing data was universally dealt with by filling with empty strings, an appropriate method given that the entire dataset consists of categorical string data. To avoid label overlapping in graphs and display data more cleanly, some data was abbreviated; for example, "United Arab Emirates" was shortened to "UAE". Posting dates stored as strings were converted into `Pandas` DateTime objects for more straightforward computation. Finally, two columns with no useful information in the FindAPhD scraper, "Program Link" and "Program Title", were dropped from the DataFrame. The scraper was coded to handle these fields, but only a negligible amount of postings contained parent program information.

```
df['Title'] = df['Title'].fillna("").str.lower()
df['Tags'] = df['Tags'].fillna("").str.lower()
df['Full Description'] = df['Full Description'].fillna("").str.lower()
df = df.drop(columns=["Program Title", "Program Link"])
df['Country'] = df['Country'].replace("United Arab Emirates", "UAE")
```

Listing 4: Aforementioned `Pandas` Cleaning Operations

Following cleanup, the final dataset contained the following information columns for FindAPhD: Title, country, university, department, supervisor, funding, top skills, tags, link, full description, search terms, focus, LLM tags, and full LLM response. AcademicPositions, on the other hand, contained the following fields: Title, country, employer, posting date, top skills, tags, link, full description, search terms, LLM tags, and full LLM response. The `Matplotlib` and `Seaborn` libraries for Python were used as the primary means of data visualization. For most statistics—namely those that can be visualized using simple bar charts, line plots, histograms, or box plots—data could be directly plotted without any further processing.

Some figures in section 4 contain a “similarity index” between two datasets. This measure refers to the Jaccard Similarity Index, the ratio of the intersection over union of two sets. Below is a short demonstration on how to calculate it:

```
def jaccardSimilarity(set1, set2):
  if not set1 or not set2:
      return 0.0
  return len(set1 & set2) / len(set1 | set2)
```

Listing 5: Calculating Jaccard Similarity

Creating word clouds and heatmaps was also straightforward, but required importing additional dedicated Python libraries for these purposes. There were, however, three graphs that required supplementary data manipulation. The first was the bar chart of the most common departments in FindAPhD (listed as Figure 6). The extra step emerged in response to the need to “normalize” department names and account for slight differences in names of analogous schools and departments among different universities and employers. This was achieved by defining a mapping of keyword patterns to department labels, which was then applied to the DataFrame’s “Department” column to form a new “Normalized Department” attribute. For example, the following keywords, if found on a department’s name, would lead it to be classified as “computer science”: ”computer science”, “computing”, “computational”, “informatics”, “computer”, and “machine learning”. If no match was found, as was often the case with departments corresponding to niche or emerging subjects, the posting was classified as “other”. A comparatively small “other” group indicates a good quality mapping.

```python
def normalizeDepartment(dpt):
  dpt = str(dpt).lower()
  for label, patterns in dptPatterns.items():
      for pattern in patterns:
          if re.search(rf'\b{pattern}\b', dpt):
              return label
  return "other"
```

Listing 6: Department Name Normalization

The second and third were the tables depicting semantic clusters of LLM-generated tags for each website (listed as Tables 3 and 4). These clusters were obtained using the following process:

1. Using `Pandas`, the list of LLM tags was cleaned and standardized by removing duplicates and missing values, reordering alphabetically, and concatenating into a single "Tag Sentence" DataFrame column to be used as embedding input.
2. Using `all-MiniLM-L6-V2`, a pre-trained model from the `sentence_transformers` Python library, each tag sentence was converted into a numerical vector that retains semantic meaning.
3. Using a Python implementation of UMAP, a dimensionality reduction technique, from the `umap_` library, the vectors were reduced down to 2D for clustering and visualization.
4. Using an implementation of K-Means, a clustering method, from the `scikit-learn` Python library, similar tag groupings were grouped into k = 6 clusters. Cluster info for each posting was kept in a new DataFrame column.
5. The clusters were visualized in a scatter plot using `Seaborn` and `Matplotlib`.
6. A selection of "tag sentences" from each cluster was printed on the standard python output. Based on these, each cluster was then manually given a name.

After experimenting with different cluster amounts for step 4, six was deemed optimal. It consistently produced a selection of easily distinguishable clusters, minimizing overlap, overfitting, and oversimplification. Because the intermediary scatter plots do not hold any immediate semantic meaning to humans, they are not featured in section 4 and are kept in the appendix (Figures 27 and 28).

3.5 Limitations

The investigation outline, while robust, carries limitations. Although the two target websites for scraping are suitable for this study's research goals, limiting search and scraping to just two websites can place a limit on how generalizable the findings are, especially given that some information fields were only available on FindAPhD (department, supervisor, and funding) whilst others only on AcademicPositions (posting date). Similarly regional biases can also affect how accurately the results represent the entire academic job market. Additionally, the self-sourced aspects of the search term and AI skill lists are vulnerable to researcher bias; the use of DeepSeek R1 to aid in creating the lists aimed to act as an additional quality check to reduce these proclivities. Even without this corrective measure, bias can be substantially mitigated by ensuring transparency in outlining methodology and careful assessment of results, potentially involving multiple rounds of scraping per term. Specific and careful prompt design and cross-validating AI-generated information is also paramount to avoiding biases and ensuring correctness. Lastly, certain sections of websites that are more reliant on JavaScript prove more difficult to scrape and require additional effort. However, with careful examination of their HTML structure, as outlined above, this hindrance became less pertinent.

4 Evaluation of the Investigation Results

This section presents the scraping results from both websites in tables, graphs, and plots, with accompanying interpretation and analysis. To provide structure and reflect the order and manner in which data was collected, all tables and figures are grouped into five main groups, or subsections. Section 4.1 explores general descriptive statistics obtained from data available on both websites, which was therefore easier to compare. Section 4.2 contains graphs showing trends about postings' funding type (salary), supervisor, and department found on FindAPhD, as it was the only website displaying these attributes in their postings. In contrast, information related to the posting date was only provided on AcademicPositions. This information is presented in Section 4.3. Finally, Sections 4.4 and 4.5 inspect findings derived from the additional data generated by Mistral.

4.1 General Insights

Country	Posting Count
United Kingdom	88
China	83
New Zealand	24
Canada	18
Australia	17
USA	7
Ireland	7
Singapore	6
Germany	6
UAE	5
Rest	53
Total	1194

(a) FindAPhD

Country	Posting Count
Morocco	124
The Netherlands	104
Belgium	93
Luxembourg	72
Sweden	57
Switzerland	50
France	40
Germany	26
Israel	24
Finland	17
Rest	38
Total	645

(b) AcademicPositions

Table 2: Top 10 Countries by Posting Count

Both websites provided a similar number of unique countries. Regional biases are evident; 84.67% of FindAPhD's postings are based in the United Kingdom. Other Commonwealth countries like Australia, Canada, and New Zealand also dominate the top five. AcademicPositions, while not having any outliers, collects most of its postings from the European Union, particularly Western Europe, and particularly from the Benelux region (Belgium, Netherlands, Luxembourg).

Generally speaking, university/employer data mirrors the patterns in Tables 2a and 2b. Nine of the top ten universities in FindAPhD are in the UK (Appendix Table 7), while the top employers list in AcademicPositions (Appendix Table 8) strongly resembles its respective top country list (Table 2b), particularly amongst the top four.

As expected, search terms corresponding to broad and foundational AI concepts such as AI, machine learning, and computer vision yielded the most results on both websites. The latter term, in particular, demonstrated surprisingly high relevance in European positions, gathering almost as many mentions as artificial intelligence itself. For a complete list of top search terms, see Appendix Figures 13 and 14.

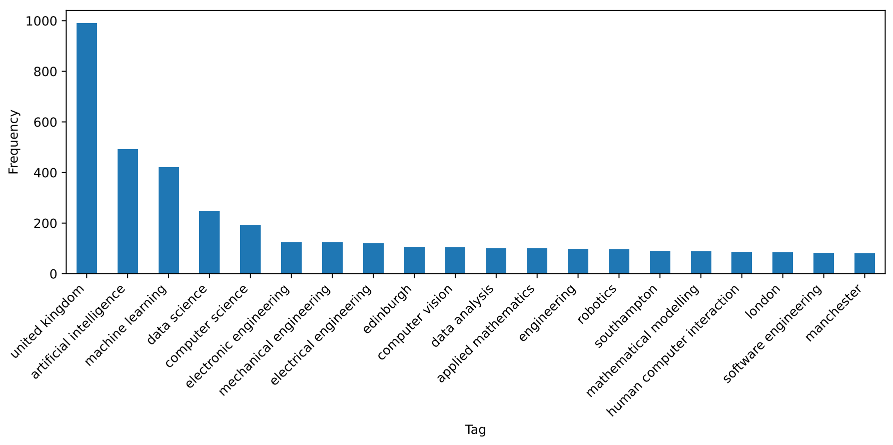


Figure 1: Top 20 Tags — FindAPhD

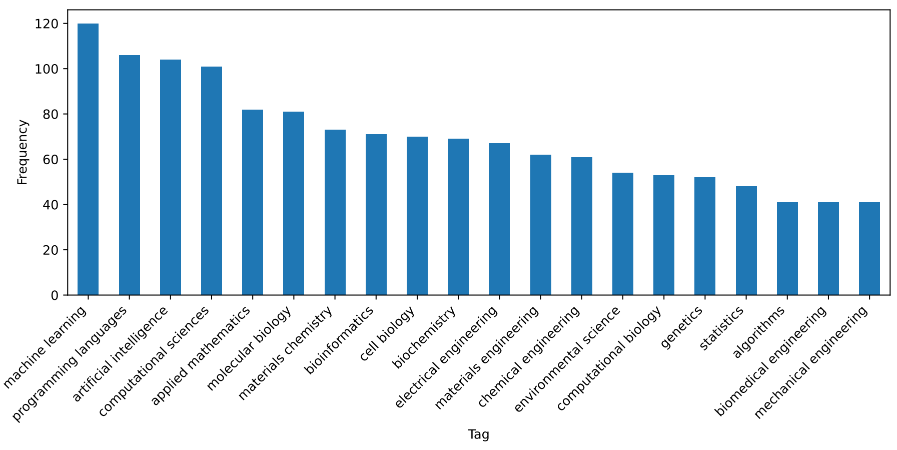


Figure 2: Top 20 Tags — AcademicPositions

The presence of tags like ”United Kingdom” and ”Edinburgh” in Figure 1 further confirms FindAPhD’s strong UK bias. Besides this, engineering (and its subfields) is well-represented. Tags in AcademicPositions (Figure 2) contain frequent mentions of life sciences (biology, chemistry, etc.), but have a broader scope on the whole—a strong hint of interdisciplinary trends. Core AI keywords (e.g., “machine learning”, “computer science”) prove amongst the most relevant on both sites. The similarity index of all tags (not just the top 20) between both websites was 0.14; an undoubtedly favorable result signalling a good choice of websites, as there is minimal overlap between the two.

On the other hand, the similarity index between the two top-skill sets was much higher, at 0.625. For this metric, this is indicative of a quality selection of skills with a good balance between shared and unique skills. FindAPhD contained 13 unique skills, some of which relate to model architecture (transformer, reinforcement learning), whereas AcademicPositions had 5 unique skills, some related to deployment and application (model deployment, data preprocessing). Core AI competencies like machine learning, data analysis, and natural language processing are nevertheless prioritized across both websites. Appendix Figures 16 and 17 contain word clouds of these skill distributions.

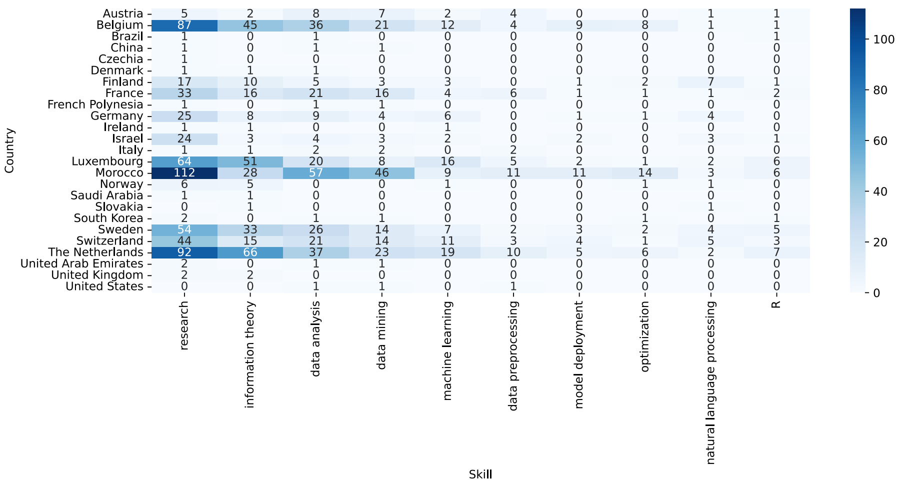
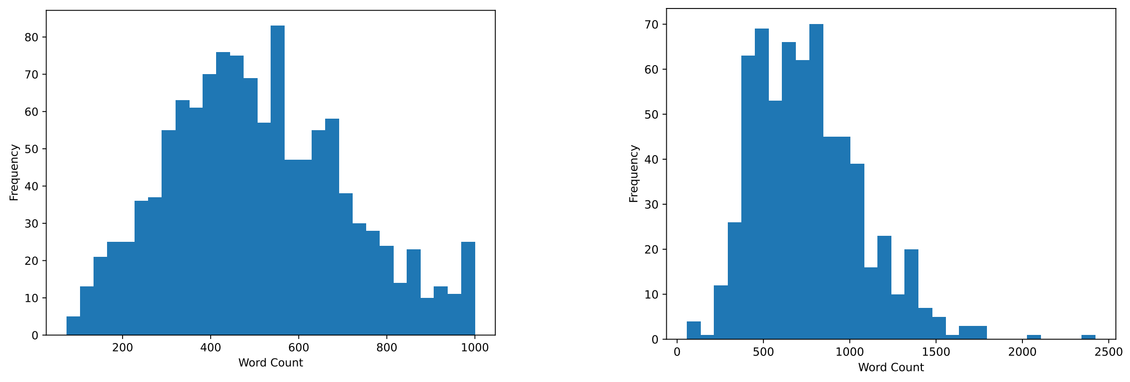


Figure 3: Skill Mentions by Country — AcademicPositions

FindAPhD's regional bias limits the utility of a heatmap as means of visualizing the proportion of skill distribution across countries. Nearly all skills in FindAPhD are clustered on UK postings, an expected outcome given the platform's regional focus. Some skills, like natural language processing and reinforcement learning, are also nearly exclusive to the UK. Despite this, core foundational AI skills like machine learning, python, data analysis, and deep learning still garner some mentions in other countries. In contrast, AcademicPositions shows a more evenly distributed skillset (Figure 3), with competencies like model deployment and data preprocessing exclusively found here. These two skills hint at a prevalence of application-related AI roles in Europe, one of the two established "key tasks" of an AI Specialist.

Certain skills show higher propensity to appear alongside each other, providing further clarity on what kind of roles and structures are dominant in each website/region. Postings in FindAPhD demand general AI literacy, with high co-occurrence between broad concepts like machine learning and research alongside central data analysis tools such as Python, MATLAB, and R. Moreover, European positions on AcademicPositions matched skills relevant on data processing and deployment, applied roles that, while still central to data science, emphasize interdisciplinary research. Appendix Figures 18 and 19 show detailed skill co-occurrence matrices.



(a) FindAPhD (b) AcademicPositions

Figure 4: Word Count Distributions Across Platforms

Figure 4 shows a pair of histograms that illustrate the distribution of posting description word counts. Both websites peak around 550 words, but have different distributions. Figure 4a (FindAPhD) is approximately normally distributed with no outliers, a pattern that extends to individual countries such as the UK, Australia, Israel, and Germany. This is a strong indicator of template usage or the existence of standardized posting formats with word count limits, which is within reason given that most postings originate from the same country.

In contrast, Figure 4b (AcademicPositions) is right-skewed, with greater variance and several notable outliers in the upper extreme. This suggests a broader variety of role types, job tasks, and discipline-specific context—a notion cross-validated by AcademicPositions's tags (Figure 2). The Netherlands and some Nordic countries (Finland, Sweden, and Norway) average around or above 1,000 words per posting, hinting at more complex tasks with wordier, discipline-specific vocabulary. For candidates and/or applicants, this translates to postings easier to read and compare in FindAPhD, and more detailed and insightful postings in AcademicPositions. For an alternative visualization of each website's variance, see Appendix Figures 20 and 21 for box plots of per-country distributions.

4.2 Insights from Department, Supervisor, and Funding Type

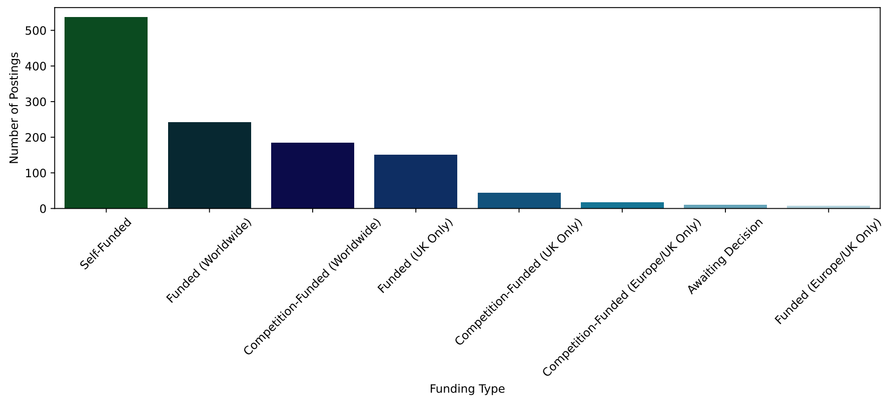


Figure 5: Distribution of Funding Types

Information about funding type was only available on FindAPhD. Figure 5 illustrates how self-funded positions dominate the platform, accounting for about half of all postings. While this raises questions about inequitable access to these opportunities (particularly amongst foreigners or lower-income individuals), most funded positions are available to applicants worldwide, suggesting a push towards international inclusivity beyond the United Kingdom. However, funded opportunities are still not the majority. Postings' funding type was largely irrelevant to their length and verbosity, with only positions labeled ”Europe/UK Only” having a slightly smaller word count average. This comparison is thus dismissed.

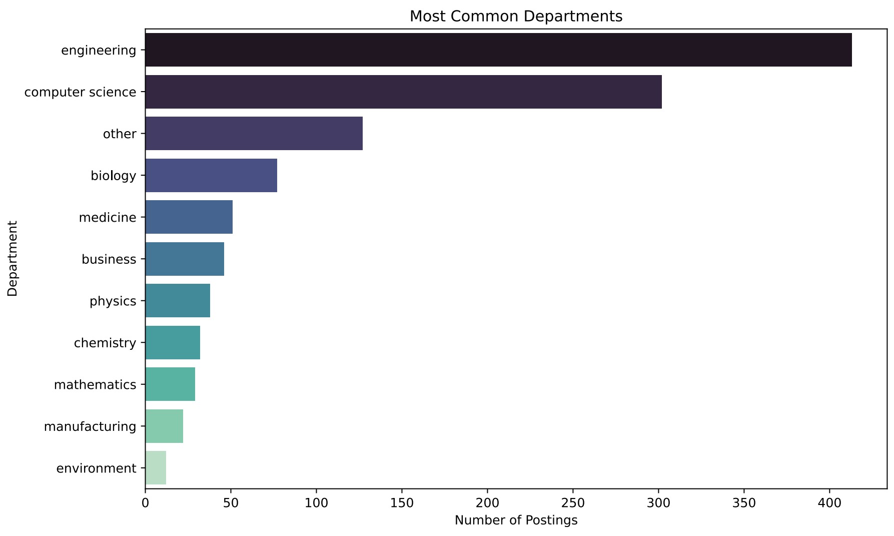


Figure 6: Most Common Departments

Figure 6 depicts the 10 largest groupings of FindAPhD postings by academic department. The classification process, which accounted for differences in department names across different institutions, is explained in detail in Section 3. Engineering and computer science account for over 60% of all postings, highlighting that AI Specialists are still most in demand in technical and applied science departments. Despite this, the notable presence of life and health sciences in the data further substantiates the notion that AI in academia is spreading beyond STEM fields strictly related to computing. The substantial amount of “other” postings is also suggestive of interdisciplinary or emerging fields.

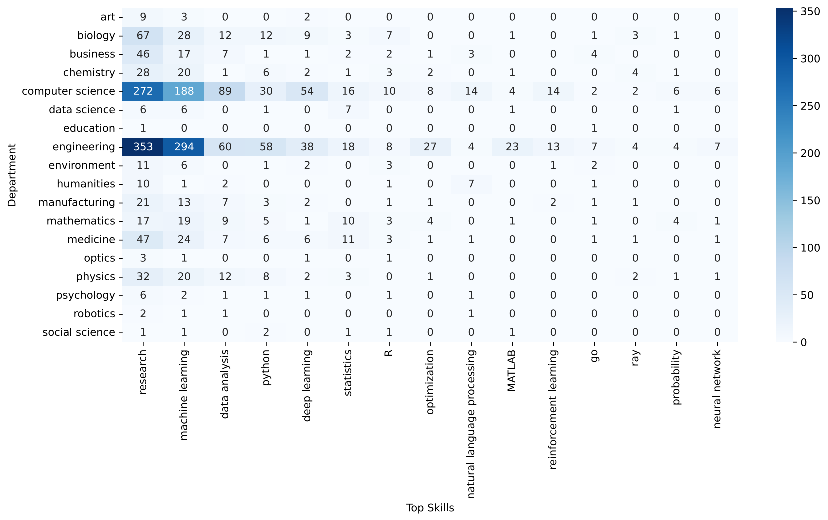


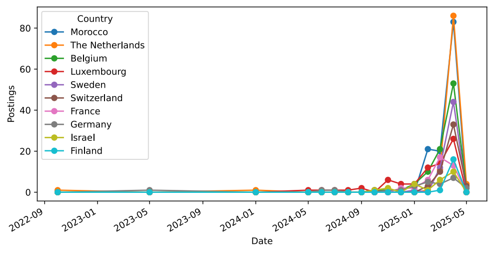
Figure 7: Skill Demand by Department

Figure 7 plots the distribution of the top 15 skills across departments. Computer science and engineering, the largest departments overall (Figure 6), also display the largest share of AI-related skills. Other non-computing science departments, including biology and medicine, show the notable need for skills like machine learning, data analysis, and

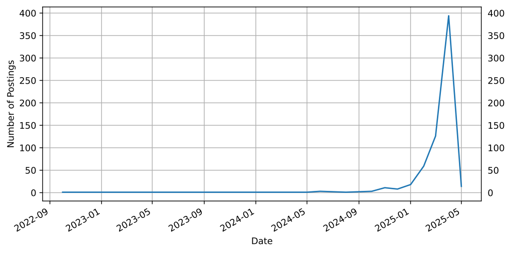
Python; this strongly indicates that postings in these departments are targeted at AI users rather than developers. Some non-STEM interdisciplinary fields, namely business, are highly represented, whereas humanities-based departments, grouped under “art”, “humanities”, and “social sciences” in the heatmap, remain starkly underrepresented.

Information about posting supervisors was also exclusive to FindAPhD. While not as relevant to the aims of this study, it’s worth noting that 31.3% of listings have more than one supervisor associated with them. A table of the top ten most mentioned supervisors can be found in Appendix Table 9.

4.3 Insights from Temporal Data



(a) FindAPhD



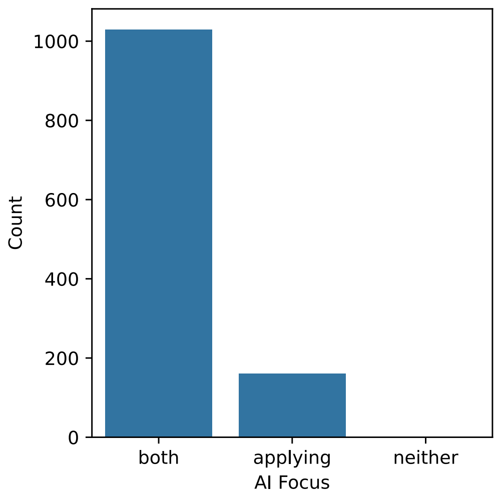
(b) AcademicPositions

Figure 8: Time Distributions

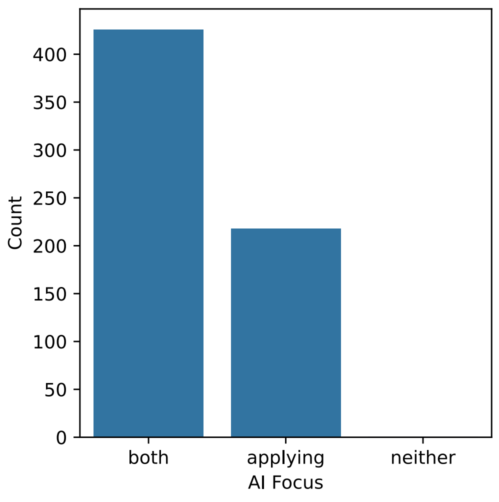
Using information about postings’ posting dates, which is only available on AcademicPositions, Figure 8 illustrates temporal trends in posting volume and per-country distributions, which are largely analogous. The vast majority of listings were posted in the calendar year 2025, particularly in April (the period of time in which data was scraped). This is indicative of three things: firstly, job postings do not tend to remain up for longer than 3-4 months, hinting at how long these take to get filled; secondly, the April spike corresponds to a “spring hiring season” ahead of fall/winter semester 2025, which in Europe tends to begin in September or October; lastly, posting patterns are likelier to be explained by macro-level trends instead of individual institution choices.

What day of the week a posting was published offers minor insights: most occur in the first half of the week, ensuring visibility for the rest of the week and following standard university workflows. This additionally confirms that the scraper or analysis introduced no date bias, as these trends reflect expected human behavior. The corresponding graph is listed as Figure 22 in the appendix.

4.4 Insights from LLM Posting Focus



(a) FindAPhD



(b) AcademicPositions

Figure 9: Focus Distribution

AI positions chiefly follow a dual focus on both developing new AI technologies and implementing existing ones. The difference is more predominant for FindAPhD (Figure 9a), whereas AcademicPositions has a higher proportion of "applying"-focused postings, as shown in Figure 9b. This aligns with the website's trends towards interdisciplinarity and integration of AI into non-computing fields. Lastly, the negligible amount of postings classified as "neither" on both websites reinforces the effectiveness of the scraping method and search term list, successfully collecting only postings related to AI.

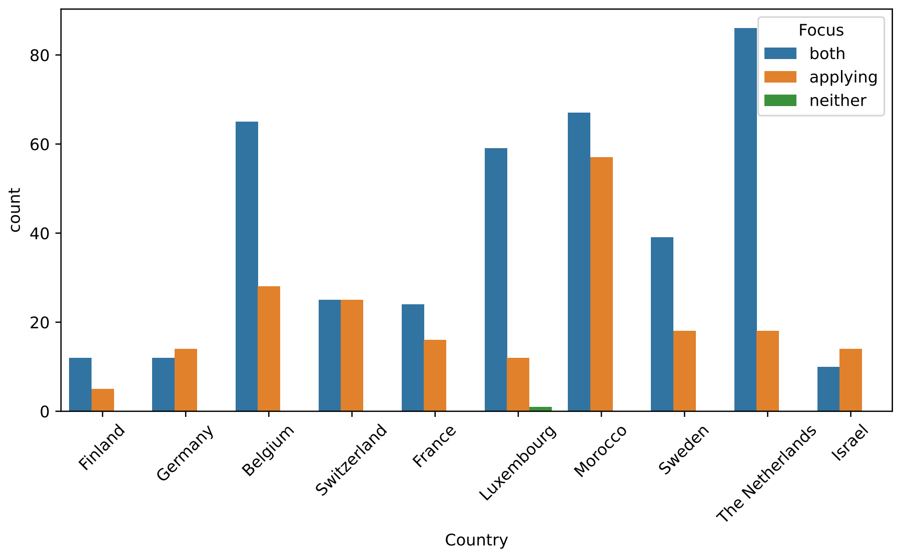


Figure 10: Focus by Country — AcademicPositions

Variation of the posting focus across countries in FindAPhD is next to none and unusable for meaningful comparisons. Conversely, AcademicPositions (Figure 10) shows a much more diverse and balanced distribution; interestingly, the focus distributions for Germany, Israel, and Switzerland are the only instances of this statistic with more "applying" positions. For most remaining countries, the ratio of "both" postings to "applying" postings hovers between 2:1 and 3:2, except for the Benelux region, which zeroes in heavily on dual-focus positions.

The overwhelming presence of postings in FindAPhD involving both developing and implementing AI technologies holds true for every department, with the strongest "both" bias occurring in computer science and engineering, the two largest departments by number of postings (Figure 6). The "both" bias is still present in non-computing fields, presumably due to employers seeking to develop customized AI tools for their research. There are, however, departments with a comparatively higher proportion of application-exclusive postings, namely medicine and biology. For an accompanying visual see Appendix Figure 23.

Per-focus analysis results in observations largely analogous to overall word trends: word counts in FindAPhD are, on average, shorter and less variant than those in AcademicPositions (Figure 4). However, those postings classified as both developing and applying new AI technologies are slightly longer—a logical corollary of the increased complexity associated with a "both" job.

4.5 Insights from LLM Tags

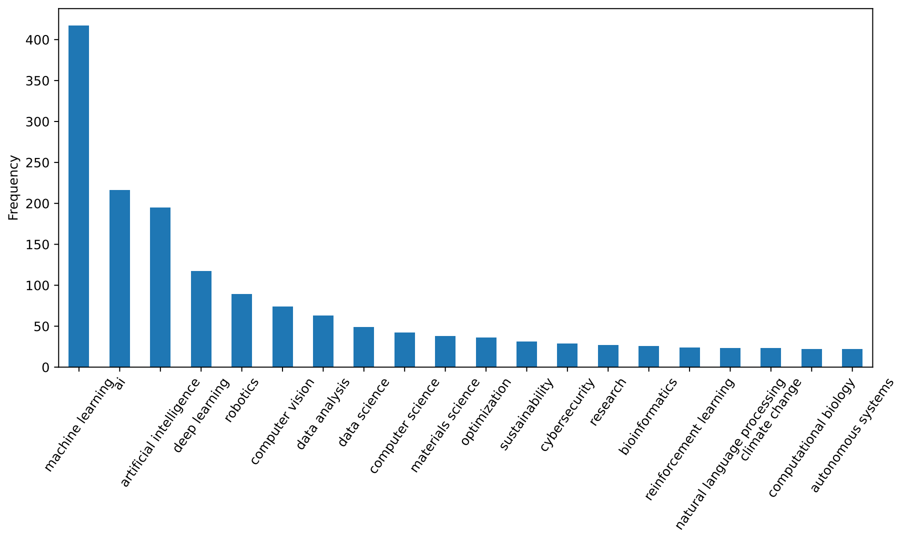


Figure 11: Top LLM Tags - FindAPhD

Figures 11 and 12 show each website's most common LLM-generated tags, captivating nuances and latent content that simple tag scraping and keyword matching cannot derive as easily. The top tags in FindAPhD are dominated by concepts central to AI, such as machine and deep learning, whereas AcademicPositions shows a much broader range of disciplines, demonstrating a demand for AI implementations amongst life sciences (biology, chemistry, etc.). The presence of bioinformatics, climate change, computational biology, and sustainability in both websites designates them as topics/fields with particularly outstanding convergence with AI.

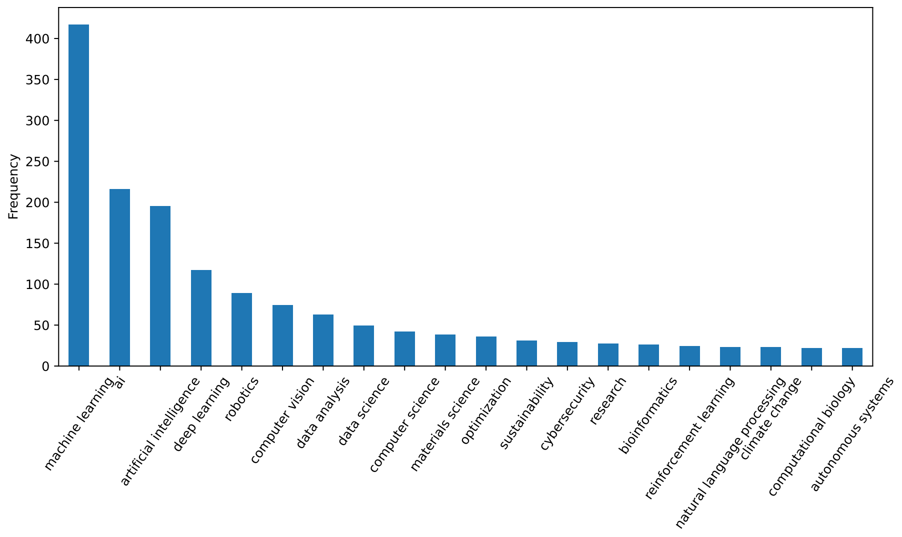


Figure 12: Top LLM Tags - AcademicPositions

The Jaccard similarity indexes between the human-assigned and LLM-generated sets of tags across all postings for both websites were mostly between 0 and 0.15. Both distributions indicate a strong divide between their respective sets, corresponding to a high grade of semantic enrichment provided by the LLM; 97% and 94.5% of LLM tags were not contained in the human tag set. This demonstrates that the LLM tags are not redundant and offer utility in filling gaps where the human tags are inconsistent or insufficient for thematic analysis. How useful the tags will be depends on how meaningful and/or relevant the human tag labeling is. In this case, higher tag overlap in AcademicPositions (Appendix Figure 24b) means the LLM tags will be less transformative than in FindAPhD, which included tags irrelevant to the focus of the posting (Figure 1).

The LLM Tag word clouds for FindAPhD (Appendix Figure 25) and AcademicPositions (Appendix Figure 26) visually represent the high enrichment mentioned earlier. Both websites show a broad variety of tags encapsulating both skills and departments/fields, which is particularly beneficial for FindAPhD, whose postings' human tags did not tend to vary thematically.

Cluster Name	**LLM Tag Examples**
Innovation and Society	Entrepreneurship, start-up, digital humanities, etc.
Chemistry and Materials	Nanotechnology, cheminformatics, 2D materials, etc.
Infrastructure and Environment	Sustainability, climate change, smart city, etc.
Theory and Human-Computer Interaction	Machine learning, mathematical theory, control systems, etc.
Biology and Medicine	Molecular biology, medical imaging, protein engineering, etc.
Engineering and Industry	Industrial automation, astrophysics, fluid dynamics, etc.

Table 3: LLM Tag Cluster Labels and Content Sample — FindAPhD

Table 3 displays the results of the vector mapping, dimensionality reduction, clustering, and plotting process described in Section 3 for FindAPhD. Postings here feature a great variety of themes, with both STEM-heavy groupings (clusters 2, 5, and 6) and groupings concerned with applicability in society (clusters 1 and 3). Cluster 4's title, ”Theory and Human-Computer Interaction”, is in agreement with the overwhelming number of ”both”-focused postings for this website. Despite previously appearing to be focused on highly technical fields like engineering, LLM tags reveal that there is still a substantial degree of interdisciplinarity being fostered among the top departments, both those STEM-related and otherwise. Similarly, AcademicPositions's cluster names (Table 4) largely align with and validate earlier findings about disciplines and skills. While some clusters remain clearly technical and domain-speciic (clusters 1 and 6), others gather enough cross-discipline or application-based themes to garner a label explicitly referring to ”interdisciplinarity” (cluster 4).

Cluster Name	**LLM Tag Examples**
Computation and Robotics	Robotics, sensor networks, edge computing, etc.
Chemistry and Materials	Chemistry, materials science, energy, etc.
Environment and Agriculture	Environmental science, water treatment, precision agriculture, etc.
Interdisciplinary and Academia	Data science, human-computer interaction, higher education, etc.
Biology and Medicine	Molecular biology, bioinformatics, computational biology, etc.
Engineering and Communication	Signal processing, wireless communication, fluid dynamics, etc.

Table 4: LLM Tag Cluster Labels and Content Sample — AcademicPositions

The number of clusters (6) also resulted in surprisingly similar clusters between the two websites: both websites had clusters where the names “chemistry and materials” and “biology and medicine” were deemed appropriate. Other direct matches include mentions of ”engineering” and ”environment”. These largely correlate with the departments found to be the most mentioned (Figure 6).

Some of the LLM Tags in each cluster appear to point at incredibly specific or nuanced domains and topics (e.g., “Cyber-Physical Systems”, “Digital Humanities”, “Green Chemistry”). These should not be dismissed, as they are indicators of possible emerging fields and topics. A straightforward way to investigate this is to feed the LLM tags back into the web scraper, which would produce new results that can be cross-validated with the original ones. Overall, LLM tags reveal a level of semantic cohesion that is not as easy to discern from simple frequency counts or human-assigned tags, which do not follow a set format.

5 Conclusions

This study aimed to answer two questions pertaining to AI hiring practices in academia: which fields or domains are looking to hire the most significant number of AI specialists? And, what skills are most in demand for AI-related doctoral/postdoctoral positions in research and academia? By collecting data from two websites—FindAPhD and AcademicPositions—via web scraping and leveraging a variety of data analysis libraries for Python, a thorough quantitative analysis of 1,819 postings was carried out.

Results give enough evidence to confirm the hypothesis that computer science and adjacent STEM fields are looking to hire the most significant number of AI Specialists. Although computer science and engineering comprise the majority of postings, there is still a prominent demand in biology, medicine, business, and natural sciences such as physics and chemistry. Results also strongly support the hypothesis that machine learning, data analysis, statistics and programming are some of the most in-demand skills for AI-related job positions. For the latter, the programming languages Python, R, and MATLAB proved the most mentioned; other frequently-referenced competencies were research, deep learning, information theory, and optimization. Co-occurrence matrices showed how these core skills tend to appear alongside each other.

Particular to this study was the inclusion of LLM-derived focus classifications and posting descriptors (tags). This choice adequately compensated for any possible lack of semantic meaningfulness or detail in any posting. The vast majority of postings are focused on both developing and implementing new AI technologies; such postings also tend to be longer and more detailed. LLM tags also help identify more nuanced and emerging domains that simple pattern-matching on scarce or inconsistent metadata would seldom catch.

Overall, the study procured novel and actionable insights on AI specialists' evolving role and integration in academia. Though fields and skills closely related to computer science still dominate the market, the emerging presence of AI in life sciences and even non-STEM fields reflects macro-level interdisciplinary trends in the academic job market, which will continue to revolutionize how research is conducted and applied for many years.

References

[1] D. Abbonato et al. “Interdisciplinary research in artificial intelligence: Lessons from COVID-19”. In: *Quantitative Science Studies* 5.4 (2024), pp. 922–935. DOI: 10.1162/qss_a_00329. URL: https://doi.org/10.1162/qss_a_00329.

[2] J. W. Burton et al. “How large language models can reshape collective intelligence”. In: *Nature Human Behaviour* 8.9 (2024), pp. 1643–1655. URL: https://www.nature.com/articles/s41562-024-01959-9.

[3] John D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95.

[4] V. Krotov, L. Johnson, and L. Silva. “Legality and ethics of web scraping”. In: *Communications of the Association for Information Systems* 47 (2020), pp. 539–563. DOI: 10.17705/1CAIS.04724. URL: https://doi.org/10.17705/1CAIS.04724.

[5] R. Kusters et al. “Interdisciplinary research in artificial intelligence: Challenges and opportunities”. In: *Frontiers in Big Data* 3 (2020), p. 577974. DOI: 10.3389/fdata.2020.577974. URL: https://doi.org/10.3389/fdata.2020.577974.

[6] C. Lotfi et al. “Web scraping techniques and applications: A literature review”. In: *SCRS Conference Proceedings on Intelligent Systems*. 2021, pp. 381–394. DOI: 10.52458/978-93-91842-08-6-38. URL: https://doi.org/10.52458/978-93-91842-08-6-38.

[7] Wes McKinney. “Data Structures for Statistical Computing in Python”. In: *Proceedings of the 9th Python in Science Conference* (2010). Ed. by Stéfan van der Walt and Jarrod Millman, pp. 51–56. URL: https://pandas.pydata.org/docs/.

[8] L. G. Reddy and P. Viswanath. “A study on web scraping of selected job portals”. In: *Journal of Emerging Technologies and Innovative Research* 9.9 (2022), pp. 317–320. URL: https://www.jetir.org/papers/JETIR2209317.pdf.

[9] Leonard Richardson. *Beautiful Soup Documentation*. Accessed: 2025-05-17. 2023. URL: https://www.crummy.com/software/BeautifulSoup/bs4/doc/.

[10] J. Schedlbauer, G. Raptis, and B. Ludwig. “Medical informatics labor market analysis using web crawling, web scraping, and text mining”. In: *International Journal of Medical Informatics* 150 (2021), p. 104453. DOI: 10.1016/j.ijmedinf.2021.104453. URL: https://doi.org/10.1016/j.ijmedinf.2021.104453.

[11] SeleniumHQ. *Selenium WebDriver*. Accessed: 2025-05-17. 2023. URL: https://www.selenium.dev/documentation/.

[12] G. Tzimas et al. “From data to insight: Transforming online job postings into labor-market intelligence”. In: *Information* 15.8 (2024), p. 496. DOI: 10.3390/info15080496. URL: https://doi.org/10.3390/info15080496.

[13] Michael Waskom. “Seaborn: Statistical Data Visualization”. In: *Journal of Open Source Software* 6.60 (2021), p. 3021. DOI: 10.21105/joss.03021.

[14] J. Zhu et al. “The stated and hidden expectations: Applying natural language processing techniques to understand postdoctoral job postings”. In: *2021 American Society for Engineering Education Annual Conference & Exhibition*. 2021. URL: https://par.nsf.gov/servlets/purl/10324348.

A Methodology Supplements

A.1 Code Fragments

```
def __init__(self):
    options = webdriver.ChromeOptions()

    ↪   options.add_argument('--disable-blink-features=AutomationControlled')
    options.add_argument('--headless=new')
    self.driver = webdriver.Chrome(
        service=Service(ChromeDriverManager().install()),
        options=options
    )
    self.baseUrl = "https://academicpositions.com" # Alternatively,
    ↪   "https://www.findaphd.com"
    self.delayRange = (3, 7)
    self.timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
    self.scriptDir = os.path.dirname(os.path.abspath(__file__))

    # Define file paths
    self.metadataPath = os.path.join(self.scriptDir,
    ↪   "scrapeHistory.csv")
    self.currentListingsPath = os.path.join(self.scriptDir,
    ↪   f"academicpositions_scrape_{self.timestamp}.csv")

    # Initialize metadata file if needed
    if not os.path.exists(self.metadata_path):
        pd.DataFrame(columns=[
            'Search Timestamp',
            'Search Terms',
            'Available Countries',
            'Top Disciplines',
            'Top Demand Skills',
            'Detailed File'
        ]).to_csv(self.metadataPath, index=False)

    self.aiSkills = [] # Full list under Appendix Table 5
```

Listing 7: Full Scraper Initialization Code

```
def scrapeSearchResults(self, keywords, numResults):
    allResults = []
    page = 1
    resultsPerPage = 15

    while len(allResults) < numResults:
        if page == 1:
            searchUrl = f"{self.base_url}/phds/?Keywords={keywords.replace(' ', '+')}"
        else:
            searchUrl = f"{self.base_url}/phds/?Keywords={keywords.replace(' ', '+')}&PG={page}"
        self.driver.get(searchUrl)
        self.randomDelay()

        try:
            WebDriverWait(self.driver, 15).until(
                EC.presence_of_element_located((By.CSS_SELECTOR, "div.phd-result,
                ↪  h3.phd-result__title")))
        except Exception as e:
            print(f"Error waiting for results: {e}")
            break

        soup = BeautifulSoup(self.driver.page_source, 'html.parser')
        listings = soup.select('div.phd-result, div.phd-result-card')

        if page > 1 and len(listings) < resultsPerPage:
            print(f"Only {len(listings)} results found on page {page}, may have reached end of
            ↪  results.")
        for listing in listings:
            try:
                title = listing.find('h3').get_text(strip=True)
                relative_link = listing.find('a')['href']
                link = f"{self.base_url}{relative_link}" if not relative_link.startswith('http')
                ↪  else relative_link
                result = {
                    'Title': title,
                    'Link': link,
                    'Top Skills': [],
                    'Supervisor': [],
                    'Tags': []
                }
                allResults.append(result)
                if len(allResults) >= numResults:
                    break
            except Exception as e:
                print(f"Error parsing listing: {e}")
                continue
        if len(listings) < resultsPerPage or len(allResults) >= numResults:
            break
        page += 1
        self.randomDelay()

    # Get countries + disciplines
    countries = []
    disciplines = []
    if allResults:
        countrySelect = soup.find('select', id='CountryInput')
        if countrySelect:
            countries = [opt.text.split(' (')[0] for opt in
            ↪  country_select.find_all('option')[1:]]
        discSelect = soup.find('select', id='DiscInput')
        if discSelect:
            allDisc = []
            for option in discSelect.find_all('option')[1:]:
                discText = option.text.strip()
                if '(' in discText:
                    discName = discText.split(' (')[0]
                    discCount = int(discText.split(' (')[1].replace(')', ''))
                    allDisc.append((discName, discCount))
            allDisc.sort(key=lambda x: x[1], reverse=True)
            disciplines = allDisc[:5]
    return allResults, countries, disciplines
```

Listing 8: Main Scraping Logic — FindAPhD

```python
def scrapeSearchResults(self, keywords, maxPages):
    baseSearchUrl = f"{self.baseUrl}/find-jobs?search={keywords.replace(' ', '+')}&positions[0]=phd&positions[1]=post-doc"
    self.driver.get(baseSearchUrl)
    self.randomDelay()
    try:
        WebDriverWait(self.driver, 15).until(
            EC.presence_of_element_located((By.CSS_SELECTOR, "div.job-list-item")))
    except Exception as e:
        print(f"Error waiting for results: {e}")
        return [], [], []
    if maxPages == -1:
        try:
            pagination = self.driver.find_element(By.CSS_SELECTOR, "ul.pagination")
            pageLinks = pagination.find_elements(By.TAG_NAME, "a")
            if pageLinks:
                lastPage = int(pageLinks[-2].text)
            else:
                lastPage = 1
        except:
            lastPage = 1
    else:
        lastPage = maxPages
    results = []
    countries = set()
    disciplines = []
    for page in range(1, lastPage + 1):
        if page > 1: # Further pages
            pageUrl = f"{baseSearchUrl}&page={page}"
            self.driver.get(pageUrl)
            self.randomDelay()
            try:
                WebDriverWait(self.driver, 10).until(
                    EC.presence_of_element_located((By.CSS_SELECTOR, "div.job-list-item")))
            except Exception as e:
                print(f"Error waiting for page {page}: {e}")
                continue
        soup = BeautifulSoup(self.driver.page_source, 'html.parser')
        currentPageListings = soup.select('div.job-list-item') # Get job listings from page
        if not currentPageListings:
            print(f"No listings found on page {page} - stopping early")
            break
        for listing in currentPageListings:
            try:
                title = listing.select_one('a.text-dark.text-decoration-none.hover-title-underline.job-link
                ↪  h4').get_text(strip=True)
                link = listing.select_one('a.text-dark.text-decoration-none.hover-title-underline.job-link')['href']
                results.append({
                    'Title': title,
                    'Top Skills': [],
                    'Tags': [],
                    'Link': link if link.startswith('http') else f"{self.baseUrl}{link}"
                })
            except Exception as e:
                print(f"Error parsing listing: {e}")
                continue
        if page == 1: # Get countries and disciplines from first page
            # Get countries
            locationHeading = None
            for h6 in soup.find_all('h6', class_='text-uppercase'):
                if 'Location' in h6.get_text():
                    locationHeading = h6
                    break
            if locationHeading:
                locationList = location_heading.find_next_sibling('ul', class_='category-section')
                if locationList:
                    for continent_li in locationList.find_all('li', recursive=False):
                        countryUl = continentLi.find('ul', class_='child')
                        if countryUl:
                            for countryLi in countryUl.find_all('li', recursive=False):
                                countryLink = countryLi.find('a')
                                if countryLink and 'name' in countryLink.attrs:
                                    countryName = countryLink['name'].replace('-', ' ').title()
                                    if countryName and len(countryName) > 2:
                                        countries.add(countryName)
            fieldHeading = None
            for h6 in soup.find_all('h6', class_='text-uppercase'):
                if 'Field' in h6.get_text():
                    fieldHeading = h6
                    break
            if fieldHeading:
                fieldList = field_heading.find_next_sibling('ul', class_='category-section')
                if fieldList:
                    allDisc = []
                    for item in fieldList.find_all('li', recursive=False):
                        try:
                            name = item.get_text(strip=True).split('(')[0].strip()
                            count = 0
                            countSpan = item.find('span', class_='count')
                            if countSpan:
                                count = int(countSpan.get_text(strip=True).strip('()'))
                            allDisc.append((name, count))
                        except:
                            continue
                    allDisc.sort(key=lambda x: x[1], reverse=True)
                    disciplines = allDisc[:5] # Get top 5 disciplines
        self.randomDelay()
    return results, sorted(countries), disciplines
```

Listing 9: Main Scraping Logic — AcademicPositions

```python
def processRegularProject(self, project):
    # Keep parent university/department if this is from a program
    isProgramSubproject = self.isProgramListing(project.get('Program Link', ''))
    originalUniversity = project.get('University', '')
    originalDepartment = project.get('Department', '')
    self.driver.get(project['Link'])
    self.randomDelay()
    try:
        WebDriverWait(self.driver, 15).until(
            EC.presence_of_element_located((By.CSS_SELECTOR, ".phd-sections__content, .phd-data__container"))
        )
        soup = BeautifulSoup(self.driver.page_source, 'html.parser')

        # Get university/department, tags, country, description
        if not isProgramSubproject:
            department = ""
            university = ""
            providerSpan = soup.find('span', itemprop='provider')
            if providerSpan:
                departmentTag = providerSpan.find('a', class_='phd-header__department')
                if departmentTag:
                    department = departmentTag.get_text(strip=True)
                universityTag = providerSpan.find('a', class_='phd-header__institution')
                if university_tag:
                    university = university_tag.get_text(strip=True)
            project['Department'] = department
            project['University'] = university
        tags = [tag.get_text(strip=True) for tag in soup.select('div.phd-data__container a.phd-data')]
        project['Tags'] = tags
        country = tags[1] if len(tags) > 1 else ""
        project['Country'] = country

        # Get funding info
        fundingInfo = ""
        keyInfoDiv = soup.find('div', class_='key-info')
        if keyInfoDiv:
            for span in keyInfoDiv.find_all('span', class_='key-info__content'):
                icon = span.find('i', class_=lambda x: x and 'fa-wallet' in x)
                if icon:
                    fundingText = span.get_text(strip=True)
                    if "Self-Funded PhD Students Only" in fundingText:
                        fundingInfo = "Self-Funded"
                    else:
                        fundingInfo = fundingText
                    break
        project['Funding'] = funding_info

        # Get supervisor info
        supervisors = []
        keyInfoDiv = soup.find('div', class_='key-info')
        if keyInfoDiv:
            for span in keyInfoDiv.find_all('span', class_='key-info__content'):
                icon = span.find('i', class_=lambda x: x and 'fa-person-chalkboard' in x)
                if icon:
                    supervisorLinks = span.find_all('a', class_='emailLink')
                    if supervisorLinks:
                        for link in supervisorLinks:
                            supervisors.append(link.get_text(strip=True))
                    else:
                        text = span.get_text(', ', strip=True)
                        text = text.replace('&nbsp;', ' ').replace(' , ', ', ').strip()
                        if text:
                            for name in [n.strip() for n in text.split(',')]:
                                if name:
                                    supervisors.append(name)
        project['Supervisor'] = supervisors
        description = ' '.join([p.get_text() for p in soup.select('.phd-sections__content')]).lower()

        # Scoring logic from Listing 1

        # Get full description
        descriptionSection = soup.find('div', class_='phd-sections__content')
        if descriptionSection:
            for script in descriptionSection(["script", "style"]):
                script.decompose()
            descriptionText = descriptionSection.get_text(separator=' ', strip=True)
            descriptionText = ' '.join(descriptionText.split())
            descriptionText = descriptionText.replace('&nbsp;', ' ')
            project['Full Description'] = descriptionText
        else:
            project['Full Description'] = ""
        return project

    except Exception as e:
        print(f"Error analyzing posting: {e}")
        project["Department"] = originalDepartment if isProgramSubproject else ""
        project["University"] = originalUniversity if isProgramSubproject else ""
        project["Country"] = ""
        project["Full Description"] = ""
        return project
```

Listing 10: Analysis and Scoring — FindAPhD

```python
def analyzePosting(self, result):
    self.driver.get(result['Link'])
    self.randomDelay()
    try:
        WebDriverWait(self.driver, 10).until(
            EC.presence_of_element_located((By.CSS_SELECTOR, "div#jobDetails, div#editor"))
        )
        soup = BeautifulSoup(self.driver.page_source, 'html.parser')
        tags = []
        publishDate = None
        employer = None
        country = None
        for row in soup.select('#jobDetails .row'):
            fieldLabel = row.select_one('.font-weight-bold')
            if fieldLabel:
                labelText = field_label.get_text(strip=True)
                if 'Field' in labelText:
                    tagLinks = row.select('.col-auto.col-md-8 a.text-dark')
                    tags = [link.get_text(strip=True) for link in tagLinks]
                elif 'Published' in labelText:
                    dateElement = row.select_one('.col-auto.col-md-8')
                    if dateElement:
                        publishDate = dateElement.get_text(strip=True)
                elif 'Employer' in labelText:
                    employerElement = row.select_one('.col-auto.col-md-8')
                    if employerElement:
                        employer = employerElement.get_text(strip=True)
                elif 'Location' in labelText:
                    locationElement = row.select_one('.col-auto.col-md-8')
                    if locationElement:
                        locationText = locationElement.get_text(strip=True)
                        # Extract country
                        if ',' in locationText:
                            country = locationText.split(',')[-1].strip()

         # Description processing
        descriptionDiv = soup.find('div', {'id': 'editor'})
        if descriptionDiv:
            for script in descriptionDiv(['script', 'style']): # Remove script/style elements
                script.decompose()
            fullDescription = ' '.join(descriptionDiv.stripped_strings) # Clean up text
            fullDescription = ' '.join(fullDescription.split())
        else:
            fullDescription = ""

        # Scoring logic from Listing 1

        result['Top Skills'] = [f"{k} ({v})" for k,v in topSkills]
        result['Tags'] = tags
        result['Posted On'] = publishDate
        result['Employer'] = employer
        result['Country'] = country
        result['Full Description'] = fullDescription
        return result
    except Exception as e:
        print(f"Error analyzing posting: {e}")
        return result
```

Listing 11: Analysis and Scoring — AcademicPositions

```python
def saveResults(self, results, countries, disciplines, keywords):
    # Flatten results
    flatResults = []
    for item in results:
        if isInstance(item, list):  # From programs
            flatResults.extend(item)
        else:
            flatResults.append(item)

    # Prepare CSV
    dataForCSV = []
    for project in flatResults:
        if self.isProgramListing(project['Link']):
            continue # Skip if program
        row = {
            'Title': project['Title'],
            'Country': project.get('Country', ''),
            'University': project.get('University', ''),
            'Department': project.get('Department', ''),
            'Supervisor': ', '.join(project.get('Supervisor', [])),
            'Funding': project.get('Funding', ''),
            'Top Skills': ', '.join(project.get('Top Skills', [])),
            'Tags': ', '.join(project.get('Tags', [])),
            'Link': project['Link'],
            'Full Description': project.get('Full Description', ''),
            'Program Title': project.get('Program Title', ''),
            'Program Link': project.get('Program Link', '')
        }
        dataForCSV.append(row)

    # Save to CSV, update metadata
    df = pd.DataFrame(data_for_csv)
    csvPath = os.path.join(self.script_dir,
    ↪  f"phd_listings_{self.timestamp}.csv")
    df.to_csv(csvPath, index=False)
    skillTotals = defaultdict(int)
    for r in flatResults:
        if not self.isProgramListing(r['Link']):
            for skillEntry in r.get('Top Skills', []):
                skill, score = skillEntry.split(' (')
                skillTotals[skill] += int(score.rstrip(')'))
    newMetadata = pd.DataFrame([{
        'Search Timestamp': self.timestamp,
        'Search Terms': keywords,
        'Available Countries': ', '.join(countries),
        'Top Disciplines': ', '.join(f"{d} ({c})" for d,c in disciplines),
        'Top Demand Skills': ', '.join(f"{k} ({v})" for k,v in sorted(
            skill_totals.items(), key=lambda x: x[1], reverse=True)[:5]),
        'Listings File': os.path.basename(csv_path)
    }])
    metadataDf = pd.read_csv(self.metadataPath)
    metadataDf = pd.concat([metadataDf, newMetadata], ignore_index=True)
    metadataDf.to_csv(self.metadataPath, index=False)
```

Listing 12: Saving Results — FindAPhD

```python
def saveResults(self, results, countries, disciplines, keywords):
# Save to timestamped CSV
listingsDf = pd.DataFrame([{
    'Title': r['Title'],
    'Employer': r.get('Employer', ''),
    'Country': r.get('Country', ''),
    'Posted On': r.get('Posted On', ''),
    'Top Skills': ', '.join(r['Top Skills']),
    'Tags': ', '.join(r['Tags']),
    'Link': r['Link'],
    'Full Description': r.get('Full Description', '')
} for r in results])
listingsDf.to_csv(self.current_listings_path, index=False)

# Update metadata
skillTotals = defaultdict(int)
for r in results:
    for skillEntry in r['Top Skills']:
        skill, score = skillEntry.split(' (')
        skillTotals[skill] += int(score.rstrip(')'))
newMetadata = pd.DataFrame([{
    'Search Timestamp': self.timestamp,
    'Search Terms': keywords,
    'Available Countries': ', '.join(countries),
    'Top Disciplines': ', '.join(f"{d} ({c})" for d,c in
    ↪  disciplines),
    'Top Demand Skills': ', '.join(f"{k} ({v})" for k,v in sorted(
        skillTotals.items(), key=lambda x: x[1],
        ↪  reverse=True)[:5]),
    'Listings File': os.path.basename(self.current_listings_path)
}])
metadataDf = pd.read_csv(self.metadataPath)
metadataDf = pd.concat([metadataDf, newMetadata],
 ↪  ignore_index=True)
metadataDf.to_csv(self.metadataPath, index=False)
```

Listing 13: Saving Results — AcademicPositions

```
dptPatterns = {
        "computer science": [r"computer science", r"computing",
        ↪  r"computational", r"informatics", r"computer", r"machine
        ↪  learning"],
        "data science": [r"data science", r"statistics"],
        "engineering": [r"engineering", r"aerospace", r"civil",
        ↪  r"electrical", r"mechanical", r"aeronautics"],
        "biology": [r"biology", r"biosciences", r"life sciences",
        ↪  r"biological", r"biomedical", r"bioengineering"],
        "chemistry": [r"chemistry", r"biochemistry"],
        "physics": [r"physics", r"fluid dynamics", r"acoustics"],
        "psychology": [r"psychology", r"psychological"],
        "mathematics": [r"mathematical", r"mathematics"],
        "medicine": [r"medical", r"medicine", r"dentistry", r"dental",
        ↪  r"pharmacy", r"pharmaceutical", r"health", r"brain",
        ↪  r"genetics", r"cancer", r"dermatology",
        ↪  r"dermatological"],
        "business": [r"management", r"business", r"economics",
        ↪  r"financial", r"finance"],
        "law": [r"law"],
        "environment": [r"environment", r"environmental",
        ↪  r"sustainability", r"ecological", r"marine", r"ecology",
        ↪  r"agroecology"],
        "robotics": [r"robotics"],
        "manufacturing": [r"manufacturing", r"materials", r"chips"],
        "art": [r"design", r"arts", r"architecture"],
        "education": [r"education"],
        "humanities": [r"humanities", r"history", r"archaeology",
        ↪  r"religion"],
        "social science": [r"social sciences", r"sociology",
        ↪  r"political science", r"politics", r"political",
        ↪  r"people"],
        "optics": [r"optics", r"photonics", r"photonic",
        ↪  r"optogenetic"]
    }
```

Listing 14: Department Name Mapping

```
def buildPrompt(description):
    return f"""
        You're helping me classify PhD/Post-PhD academic job postings.
        ↪  Description of Job:
        \"\"\"
        {description}
        \"\"\"
        Answer the following in JSON format with two keys:
        1. "focus": choose one of the following options based on the main
        ↪  aim of the posting:
            - "developing", if the position focuses on creating new AI,
            ↪  models, techniques, or technologies.
            - "applying", if the position focuses on using existing AI
            ↪  techniques to solve domain-specific problems (e.g., using
            ↪  ML for biology or social sciences).
            - "both", if both of the above are true
            - "neither", if the position is unrelated to AI or only uses
            ↪  it slightly.
        2. "tags": a list of 3{7 short, relevant tags that summarize the
        ↪  topics, methods, or applications of this posting.
        Please do not respond with any explanation, reasoning, or text
        ↪  besides the JSON object. This is an example of how I want you
        ↪  to respond:
        {{
        "focus": "applying",
        "tags": ["machine learning", "NLP", "healthcare"]
        }}
        """
```

Listing 15: Prompt Fed to Mistral 7B

```
def getLLMResponse(prompt,
↪  endpoint="http://localhost:11434/api/generate"):
    payload = {
        "model": "mistral",
        "prompt": prompt,
        "stream": False
    }
    try:
        r = requests.post(endpoint, json=payload)
        return r.json()["response"].strip()
    except Exception as e:
        print("Error: ", e)
        return None
```

Listing 16: Feeding of Prompt to Mistral 7B

```python
import time
import requests
import json
focusList = []
tagList = []
rawResponses = []
for idx, row in df.iterrows():
    print(f"Processing listing {idx + 1}/{len(df)}")
    prompt = buildPrompt(row['Full Description'])
    response = getLLMResponse(prompt)

    rawResponses.append(response)
    focus = None
    tags = None

    if response:
        try:
            data = json.loads(response)
            focus = data.get("focus")
            tags = ", ".join(data.get("tags", []))
        except json.JSONDecodeError:
            print(f"Couldn't decode JSON at {idx}: {response}")

    focusList.append(focus)
    tagList.append(tags)
    time.sleep(1)

df['Focus'] = focusList
df['LLM Tags'] = tagList
df['Full LLM Response'] = rawResponses
```

Listing 17: Parsing of Mistral 7B's Responses

```
from sentence_transformers import SentenceTransformer
from sklearn.cluster import KMeans
import umap.umap_ as umap

# Create "tag sentences" by concatenating a posting's tags
df['Tag Sentence'] = df['LLM Tags'].fillna("").apply(
    lambda x: ", ".join(sorted(set(t.strip() for t in x.split(",") if
    ↪ t.strip())))
)

# Load sentence transformer model, encode tag sentences
model = SentenceTransformer('all-MiniLM-L6-v2')
embeddings = model.encode(df['Tag Sentence'].tolist())

# Reduce dimensions via UMAP
umapReducer = umap.UMAP(n_neighbors=15, min_dist=0.1, metric='cosine',
↪ random_state=42)
umapEmbeddings = umapReducer.fit_transform(embeddings)

# Cluster with K-Means, add cluster label to DataFrame
numClusters = 6
kMeans = KMeans(n_clusters=numClusters, random_state=42, n_init='auto')
clusterLabels = kMeans.fit_predict(umapEmbeddings)
df['Tag Cluster'] = clusterLabels

# Plot
plt.figure(figsize=(10, 6))
palette = sns.color_palette("hls", numClusters)
sns.scatterplot(x=umapEmbeddings[:, 0], y=umapEmbeddings[:, 1],
↪ hue=clusterLabels, palette=palette, s=60, alpha=0.8)
plt.title("Clusters of LLM-Generated Tags (UMAP + KMeans)")
plt.xlabel("UMAP Dimension 1")
plt.ylabel("UMAP Dimension 2")
plt.legend(title='Cluster')
plt.tight_layout()
plt.show()

# Print examples from each cluster
cluster_examples = {}
for cluster in range(numClusters):
    examples = df[df['Tag Cluster'] == cluster]['Tag
    ↪ Sentence'].head(5).tolist()
    cluster_examples[cluster] = examples
cluster_examples
```

Listing 18: UMAP + K-Means Clustering Pipeline.

A.2 Tables

Category	**Skill Keywords**
Programming, Frameworks, Libraries	Python, R_programming, MATLAB, Go, Rust, C++, SQL, Java, PyTorch, TensorFlow, Hugging Face, Scikit-learn, spaCy, OpenCV, NLTK, NumPy, Pandas, *Julia, JAX, Keras, LangChain*
Core ML/AI	machine learning, deep learning, neural network, reinforcement learning, supervised learning, unsupervised learning, few-shot learning, zero-shot learning, generative model, transformer, *GANs*
Data Science, Engineering	data analysis, data mining, data preprocessing, data visualization, *feature engineering, data pipeline*
Math, Theory	linear algebra, probability, statistics, optimization, *Bayesian methods, information theory*
NLP, Computer Vision	natural language processing, speech recognition, text mining, sentiment analysis, computer vision, object detection, image segmentation, *multimodal learning*
Deployment, Production	MLOps, model deployment, distributed training, AWS, GCP, Azure, Docker, Kubernetes, *model quantization, ONNX, Spark, Ray, CI/CD*
Research, Soft Skills	problem solving, research, peer review, critical thinking *algorithm design*

Table 5: Full List of AI-related skill keywords coded into scraper. DeepSeek R1's contributions italicized.

Category	**Skill Keywords**
Hardware Model	2021 MacBook Pro
OS	MacOS Sequoia 15.1.1
RAM	16GB
RAM for LLM	12GB
Chip	Apple M1 Pro
Text Editor	Visual Studio Code
Python Version	Python 3.12.4
LaTeX Installation	MacTeX 2024

Table 6: Details of Machine and Software Used in Study

A.3 Prompts

I am currently designing a list of search terms and keywords for a web scraper I made. The scraper looks at a job posting website for PHD and Post-PHD positions, and I'm specifically interested in scraping positions related to AI. I don't want the list to feel too broad. I want it to focus on just AI. therefore, I think having a list consisting of as many 'synonyms', so to speak, of 'artificial intelligence' and 'large language model' as we can have would be ideal. Something important to keep in mind: since I'm gonna be scraping large amounts of pages, if I search up 'artificial intelligence' we would already get results for all the 'ai for/in ___' searches. so those wouldn't count as 'synonyms'. This is my draft list: Artificial intelligence, AI, machine learning, deep learning, neural network, LLM, large language model, VLM, GPT, generative AI, genAI, transformer model, pre-trained model, reinforcement learning, supervised learning, unsupervised learning, few-shot learning, AI agent, natural language processing, computer vision. What would you suggest I add to this? There is no minimum number of additions; if the list is complete, you can leave it as is.

Listing 19: Prompt Used to Complete AI-Related Search Terms

The following is a list of skills related to AI, grouped by category. These are meant to be used by a web-scraper to identify top-skills in PhD/Post-PhD job positions. Programming, Frameworks, Libraries: Python, R_programming, MATLAB, Go, Rust, C++, SQL, Java, PyTorch, TensorFlow, Hugging Face, Scikit-learn, spaCy, OpenCV, NLTK, NumPy, Pandas Core ML/AI: machine learning, deep learning, neural network, reinforcement learning, supervised learning, unsupervised learning, few-shot learning, zero-shot learning, generative model, transformer Data Science, Engineering: data analysis, data mining, data preprocessing, data visualization Math, Theory: linear algebra, probability, statistics, optimization NLP, Computer Vision: natural language processing, speech recognition, text mining, sentiment analysis, computer vision, object detection, image segmentation Deployment, Production: MLOps, model deployment, distributed training, AWS, GCP, Azure, Docker, Kubernetes Research, Soft Skills: problem solving, research, peer review, critical thinking What would you add to each category, if anything?

Listing 20: Prompt Used to Complete AI-Related Skill List

B Results Supplements

B.1 Tables

University	Posting Count
University of Southampton	88
Edinburgh Napier University	83
University of Sheffield	64
The University of Manchester	58
University of Birmingham	51
University of Bradford	39
Kingston University	38
Xi’an Jiaotong-Liverpool University	37
Durham University	36
Loughborough University	34

Table 7: Top 10 Universities – FindAPhD

Employer	Posting Count
University of Southampton	124
University of Luxembourg	70
KU Leuven	52
Eindhoven Institute of Technology	38
University of Twente	27
Hebrew niversity of Jerusalem	24
KTH Royal Institute of Technology	22
Aix-Marseille Université	20
ETH Zürich	19
Radbout University	18

Table 8: Top 10 Employers – AcademicPositions

B.2 Plots

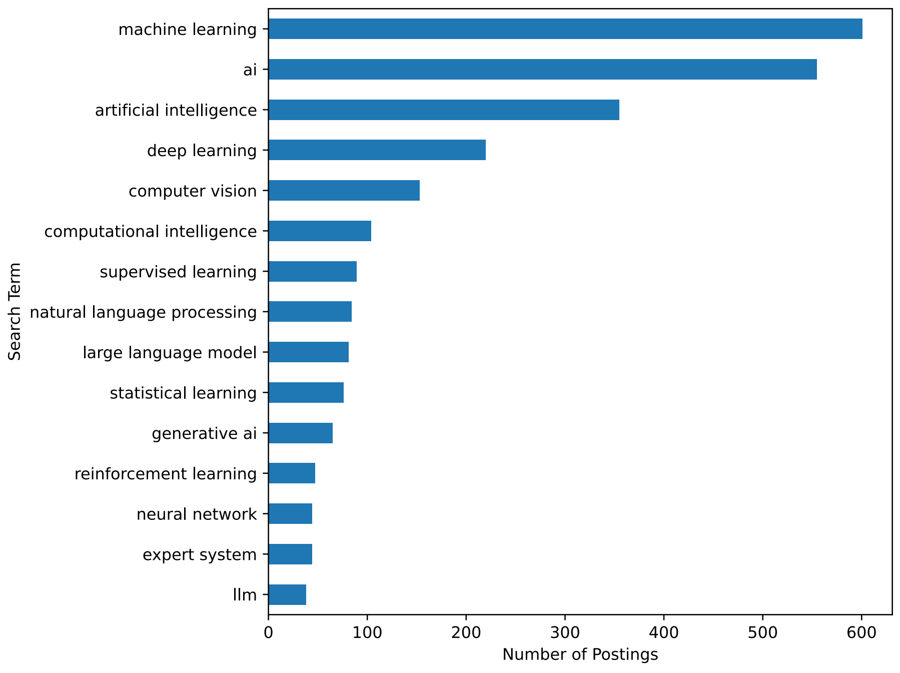


Figure 13: Top Search Terms — FindAPhD

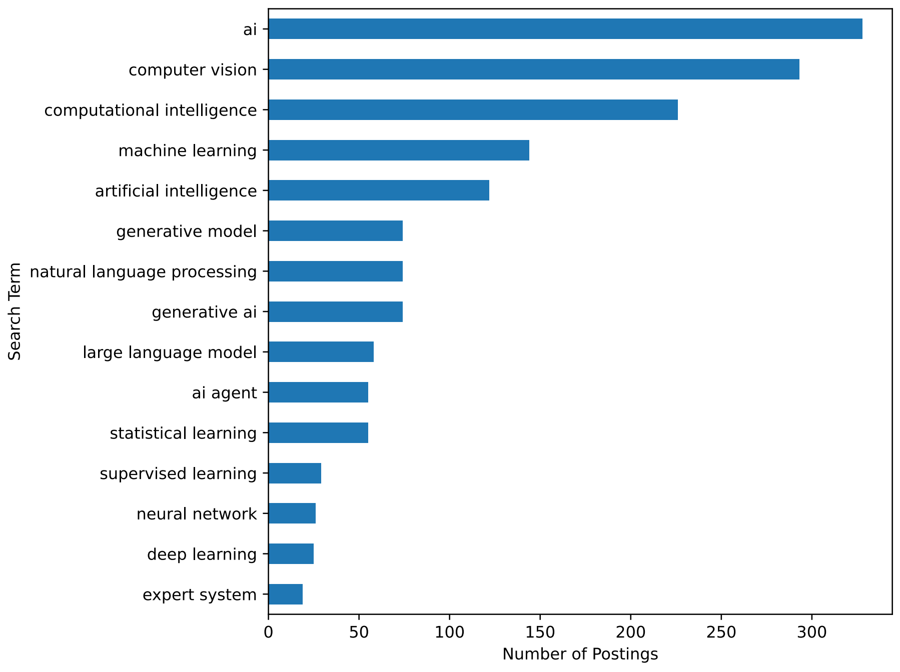


Figure 14: Top Search Terms — AcademicPositions

Supervisor	Posting Count
Dr Y Moshfeghi	8
Prof William Holderbaum	6
Prof Savas Konur	6
Prof Themis Prodromakis	6
Prof Mohamed Pourkashanian	5
Prof X Liu	5
Prof Toby Breckon	5
Dr E Delivopoulos	5
Dr Shivakumara Palaiahnakote	5
Prof Emma Hart	5

Table 9: Most Mentioned Supervisors

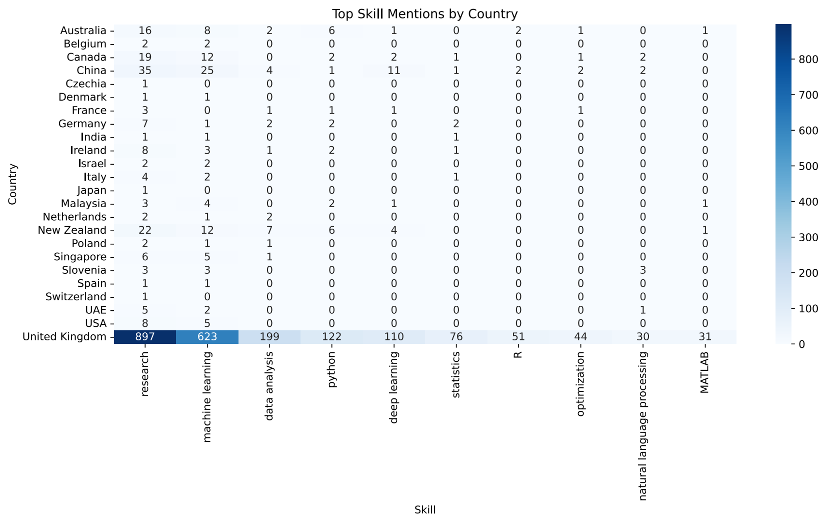


Figure 15: Skill Mentions by Country — FindAPhD

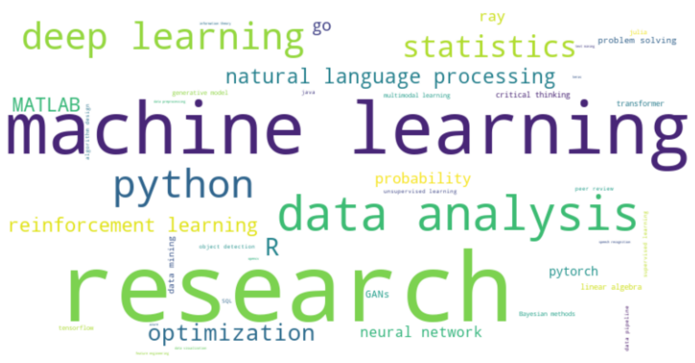


Figure 16: Top Skills World Cloud — FindAPhD

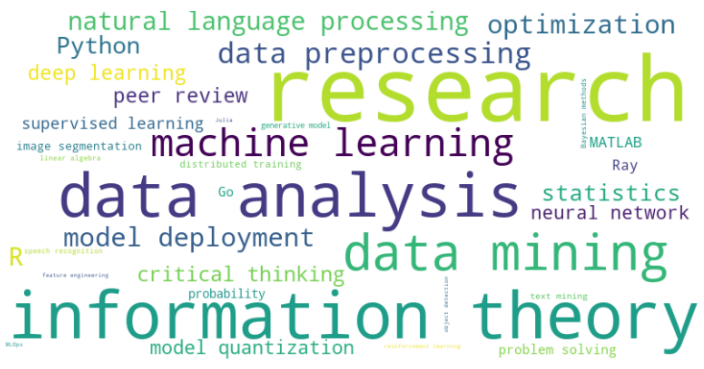


Figure 17: Top Skills Word Cloud — AcademicPositions

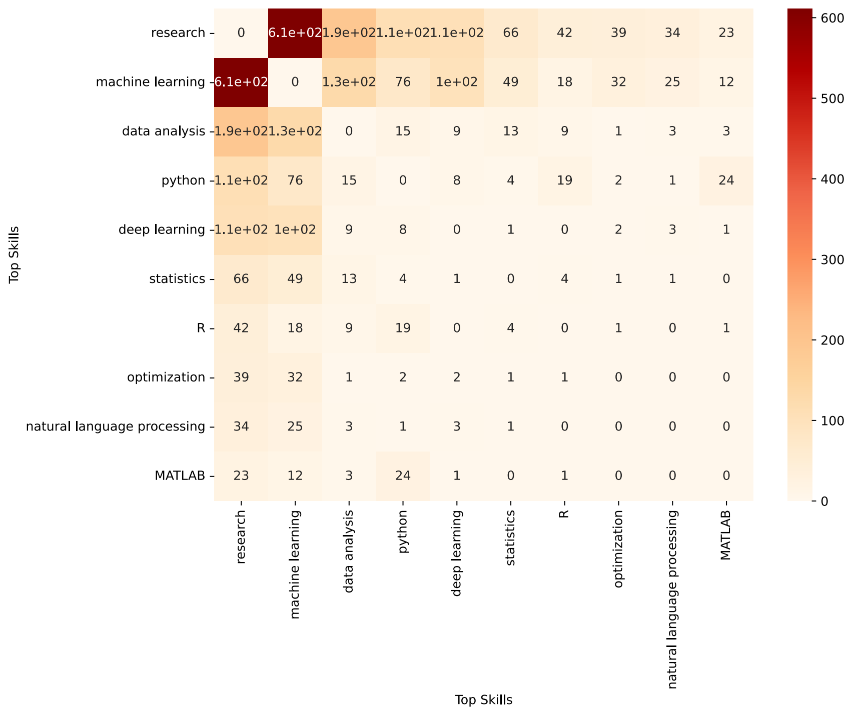


Figure 18: Skill Co-Occurrence Matrix — FindAPhD

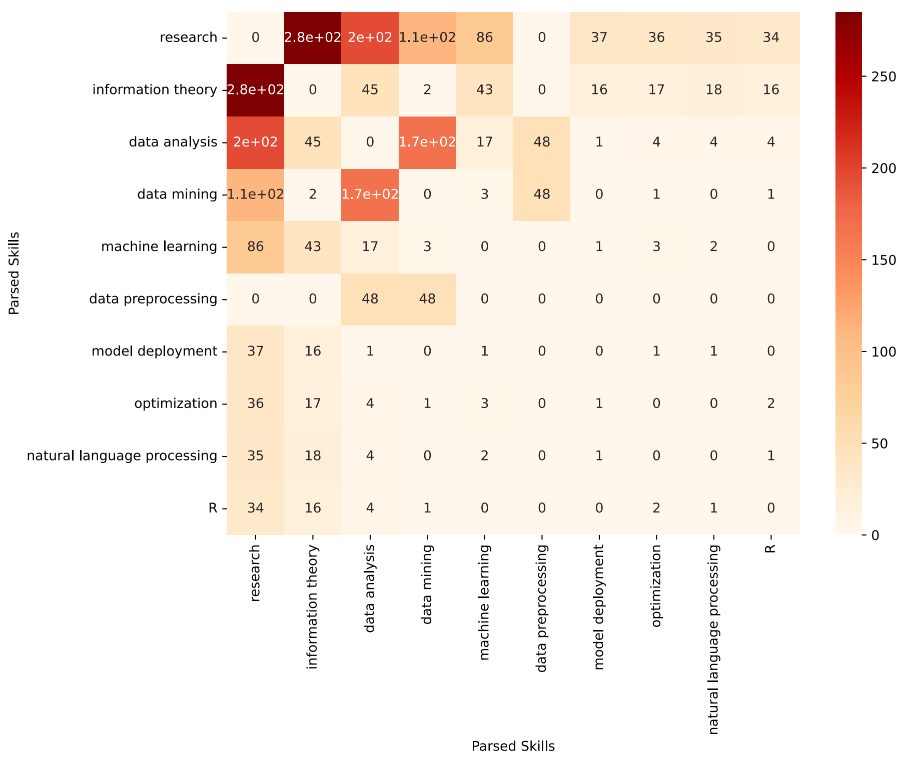


Figure 19: Skill Co-Occurrence Matrix — AcademicPositions

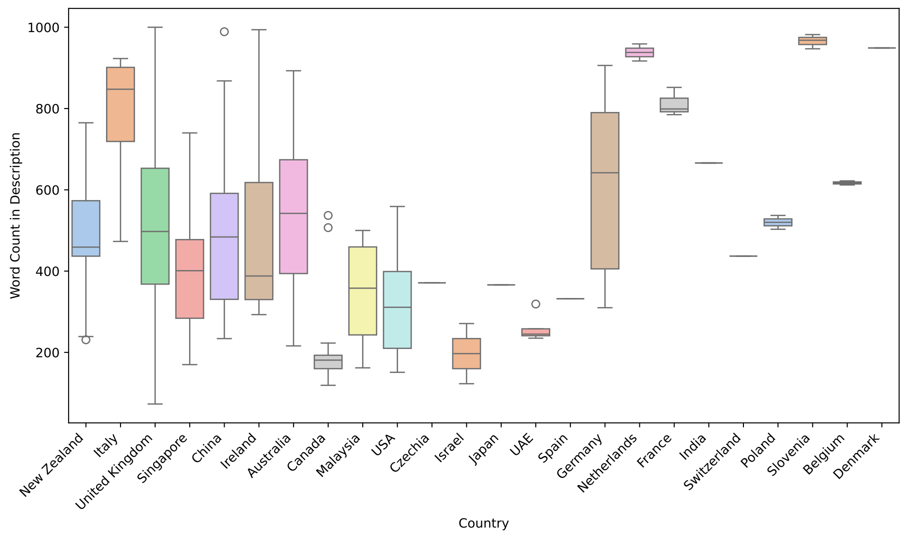


Figure 20: Posting Length by Country — FindAPhD

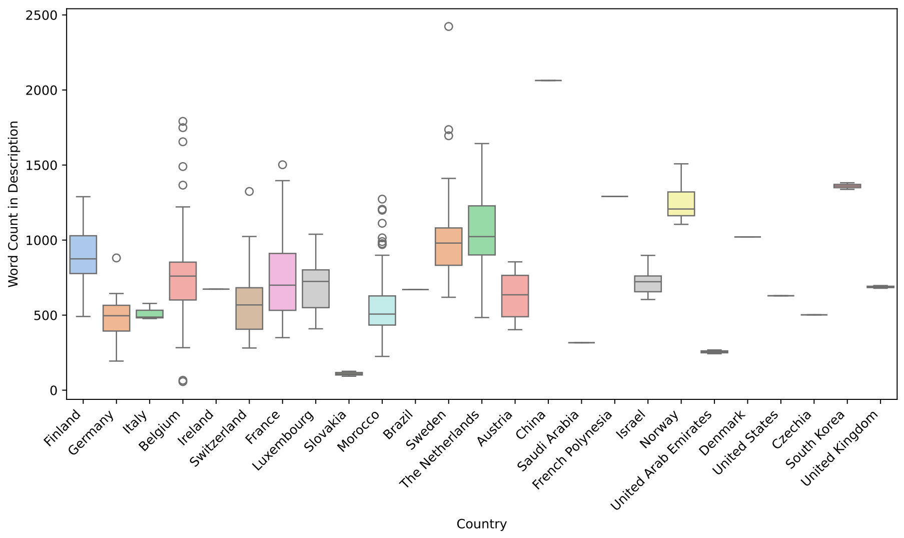


Figure 21: Posting Length by Country — AcademicPositions

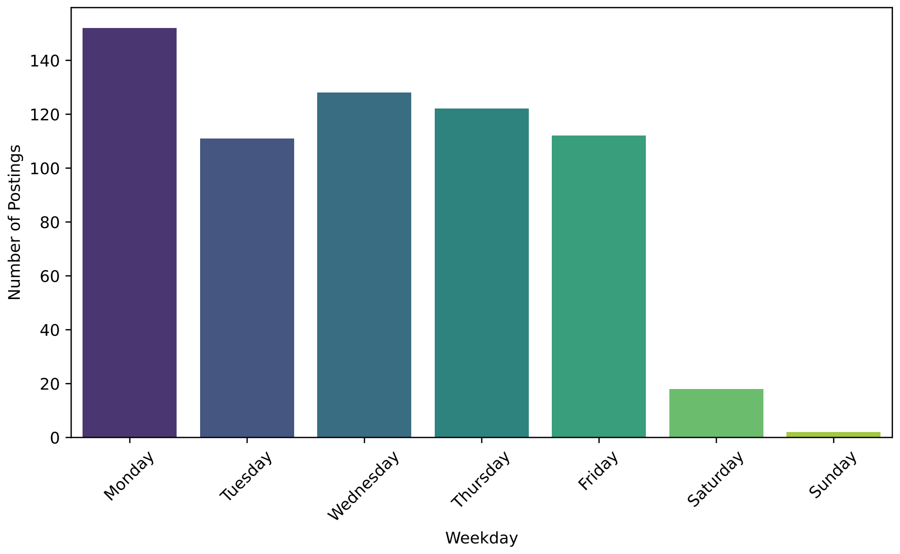


Figure 22: Postings by Weekday

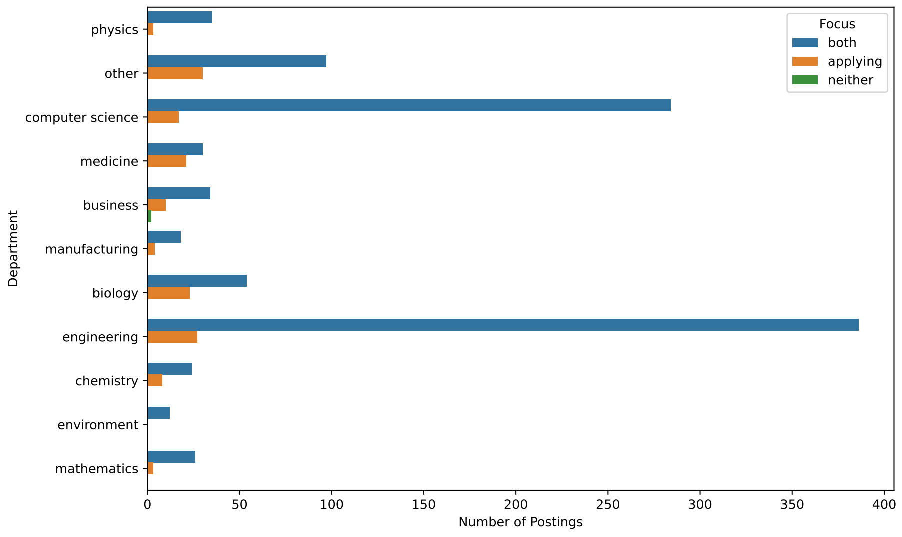
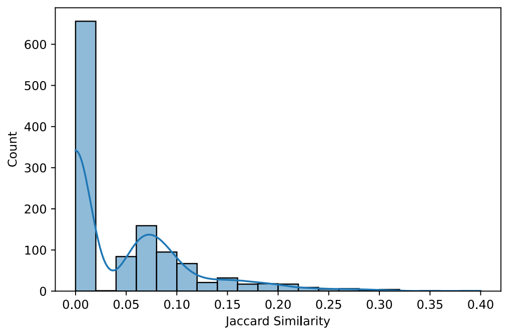
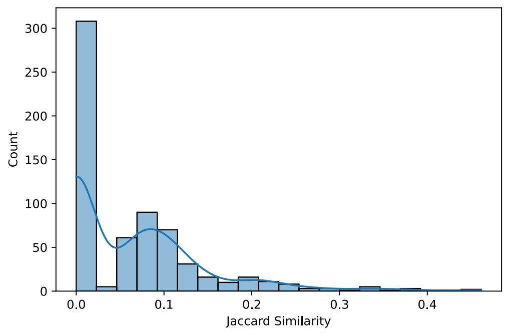


Figure 23: Focus by Department



(a) FindAPhD



(b) AcademicPositions

Figure 24: Jaccard Similarity Between Human and LLM Tags

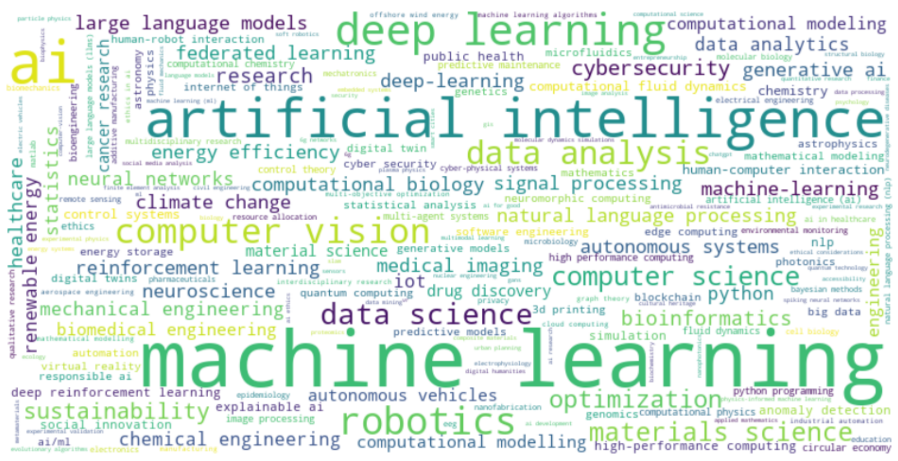


Figure 25: LLM Tags World Cloud — FindAPhD

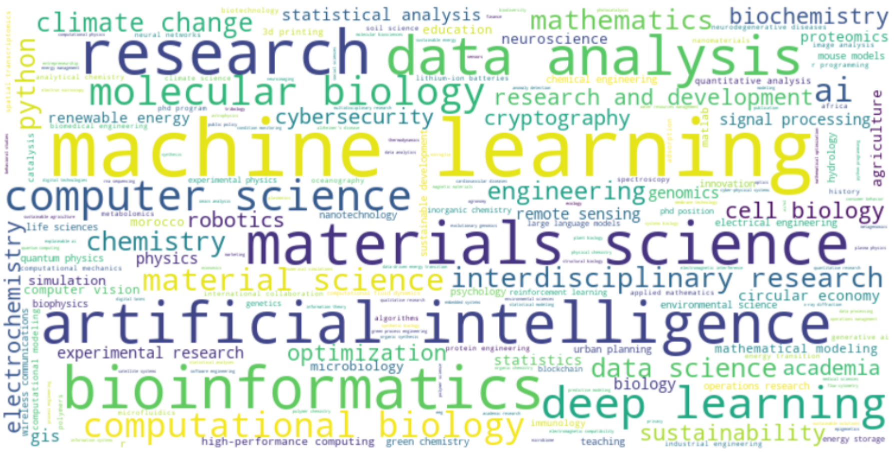


Figure 26: LLM Tags Word Cloud — AcademicPositions

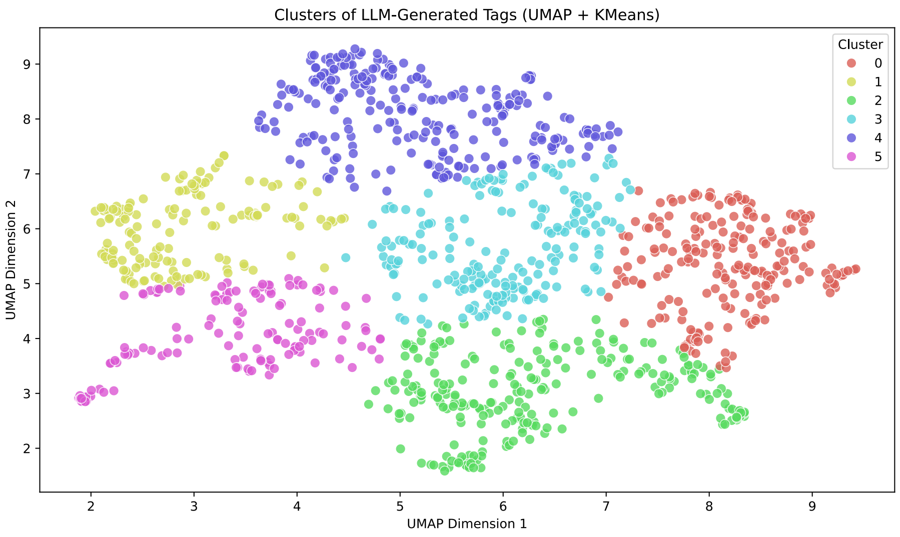


Figure 27: Clusters of LLM-Generated Tags — FindAPhD

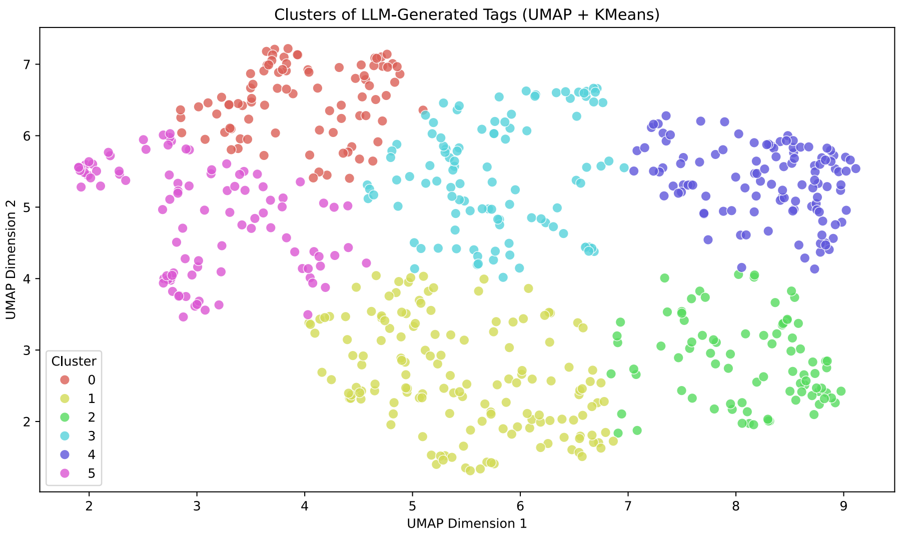


Figure 28: Clusters of LLM-Generated Tags — AcademicPositions