Одеський національний університет імені І. І. Мечникова
Факультет математики, фізики та інформаційних технологій
Кафедра Оптимального Керування та Економічної Кібернетики

Кваліфікаційна робота

на здобуття ступеня вищої освіти «бакалавр»

«Дослідження ринку праці за допомогою великих мовних моделей»

«Extracting job market insights using LLM»

Виконав: здобувач денної форми навчання
спеціальності 113 Прикладна Математика
Освітня програма «Прикладна математика»
Циналєвський Богдан Владиславович

Керівник: канд. фіз.-мат. наук, доц. Страхов Є. М.
Рецензент: Платонов В. В.

Рекомендовано до захисту:

Протокол засідання кафедри

№ ___ від _____ 2025 р.

Завідувач кафедри

_____ _____

Захищено на засіданні ЕК № _____

Протокол № ___ від _____ 2025 р.

Оцінка _____ / _____ / _____

Голова ЕК

_____ _____

Одеса — 2025 р.

Odesa I. I. Mechnikov National University
Faculty of Mathematics, Physics and Information Technology
Department of Optimal Control and Economic Cybernetics

Diploma thesis

bachelor

Extracting job market insights using LLM

Fulfilled by: full-time student
specialty 113 Applied Mathematics
Bohdan Tsynalievskyi

Supervisor:
Ph. D. in Phys. and Math. Strakhov E. M.
Reviewer: Platonov V. V.

Odesa — 2025

CONTENTS

Вступ 4

Introduction 6

1 Technology Breakdown 8
- 1.1 Large Language Models and Transformers 8
- 1.2 Retrieval-Augmented Generation (RAG) and Retrieval Techniques 9
- 1.3 RAG in our context 10
- 1.4 Vector Databases and ANN Indexing 11

2 Data Scraping and Exploratory Data Analysis 12
- 2.1 Data Collection 12
- 2.2 Exploratory Data Analysis 15
  - 2.2.1 Data Overview and Missing Data 15
  - 2.2.2 Job Ad Distribution by Category 17
  - 2.2.3 Salary Distribution 19
  - 2.2.4 Summary of Insights 21

3 Experimental Results 23
- 3.1 Performance Across Models 24
- 3.2 Experiment Iterations and Findings 25
- 3.3 Best Model Pipeline 26

Conclusion 29

Висновок 31

Bibliography 33

ВСТУП

Мета дослідження: Метою цієї роботи є розробка та оцінка підходу машинного навчання для прогнозування рівня заробітної плати за оголошеннями про вакансії в Інтернеті. Мотивація полягає в особливостях українського IT-ринку, де компанії часто не вказують розмір зарплати, що ускладнює пошук вакансій кандидатам. Завдання полягає в тому, щоб застосувати методи добування даних і сучасні технології штучного інтелекту — зокрема великі мовні моделі (LLM) та методи на основі пошуку інформації — для створення моделі, яка зможе оцінити зарплату поза описом і атрибутами вакансії. Це передбачає побудову корпусу оголошень з відомими зарплатними даними та навчання предиктивних моделей на цих прикладах.

Об'єкт дослідження: Об'єктом даного дослідження є сукупність IT-вакансій та їх характеристик, що впливають на рівень заробітної плати. Зокрема, ми працюємо з набором даних оголошень, зібраних із українського порталу DOU.ua за середину 2024 року, які охоплюють різні категорії (наприклад, програмна інженерія, наука про дані, маркетинг). Кожне оголошення містить текстовий опис (вимоги, обов'язки тощо) та структуровані поля (локація, категорія вакансії тощо). Аналіз цих даних дозволяє виявити, які ознаки (наприклад, необхідні навички, рівень позиції, місцезнаходження компанії) корелюють із вищими або нижчими зарплатами. Цей контекст визначає межі нашого дослідження: передбачення рівня заробітної плати для конкретного оголошення на основі його змісту та метаданих.

Методи дослідження: Для реалізації мети було застосовано поєднання методів інженерії даних та машинного навчання. Спершу проведено веб-скрапінг для збору сирих даних про вакансії, після чого з використанням великої мовної моделі (GPT-3.5) виконано постобробку й структурування тексту. Далі здійснено розвідувальний аналіз даних (EDA) для виявлення закономірностей та формування нових ознак (наприклад, витяг ключових слів із опису, кодування категорій вакансій). Ми протестували кілька підходів до моделювання, зокрема ансамблеві деревові моделі (Random Forest, CatBoost), нейронні мережі (багатошаровий перцептрон) та регресію з опорними векторами (SVR). Моделі навчалися та оцінювалися за часовим

розбиттям (останні оголошення використовувалися як відкладена вибірка) й порівнювалися за середньою абсолютною помилкою (MAE). Такий підхід дозволяє поєднати теоретичні надбання (зокрема останні досягнення NLP, такі як генерація з підсиленням за допомогою пошуку) з практичним моделюванням та перевіркою на реальних даних.

INTRODUCTION

Research Aim: This thesis aims to develop and evaluate a machine learning approach for predicting job salaries from online job postings. The motivation stems from the IT job market in Ukraine, where companies often do not disclose salaries, making it challenging for job seekers and recruiters to gauge fair compensation. The goal is to leverage data mining and modern AI techniques – including Large Language Models (LLMs) and retrieval-based methods – to create a model that can estimate a position's salary based on its description and attributes. This involves building a dataset of job listings with known salaries and training predictive models to generalize from those examples.

Object of Study: The object of this research is the collection of IT job advertisements and their characteristics that influence salary levels. In particular, we focus on a dataset of job postings scraped from a Ukrainian tech job board (DOU.ua) covering various job categories (e.g., Software Engineering, Data Science, Marketing) over mid-2024. Each posting includes textual descriptions (job requirements, responsibilities, etc.) and structured fields (location, job category, etc.). By analyzing this data, we seek to understand which features (such as required skills, job level, company location) correlate with higher or lower salaries. This context defines the scope of our study: predicting the salary for a given job posting based on its content and metadata.

Research Methods: To achieve the aim, we employed a combination of data engineering and machine learning methods. First, we performed web scraping to collect raw job postings, then used an LLM (GPT-3.5) to post-process and structure the textual data. Exploratory Data Analysis (EDA) was conducted to uncover patterns and inform feature engineering (e.g., extracting keywords from descriptions, encoding job categories). We then experimented with several modeling techniques, including ensemble tree-based models (Random Forest, CatBoost), neural networks (Multilayer Perceptron), and Support Vector Regression. Models were trained and evaluated using a time-based split (with the latest postings as a hold-out set) and compared using Mean Absolute Error (MAE) as the performance metric. The research approach balances theoretical insights

(drawing on recent advances in NLP like retrieval-augmented generation) with practical modeling and evaluation on real-world data.

CHAPTER 1

TECHNOLOGY BREAKDOWN

1.1 Large Language Models and Transformers

Modern Large Language Models are built on the Transformer architecture introduced by Vaswani et al. (2017). The Transformer employs a mechanism of self-attention that allows the model to weigh the importance of different words in a sequence when encoding text [1].

Unlike recurrent networks, Transformers process all tokens in parallel and use multi-head self-attention layers to capture long-range dependencies efficiently. Each encoder layer transforms its input by attending to all positions in the sequence, enabling it to learn contextual representations (for example, understanding that in "bank account" vs "river bank" the word "bank" has different meanings depending on context).

The result is a deep neural network architecture particularly well-suited for language understanding and generation tasks. Large Language Models such as GPT-3 and BERT are essentially very large Transformers (with hundreds of millions to billions of parameters) trained on massive text corpora.

These models undergo unsupervised pre-training (e.g., GPT predicts the next word, while BERT uses masked-word and next-sentence objectives) to learn a broad "knowledge" of language. After pre-training, they can be fine-tuned on specific tasks with relatively few additional parameters.

Notably, Devlin et al. (2018) showed that a pre-trained Transformer like BERT can be fine-tuned with just one additional output layer to achieve state-of-the-art results on many NLP tasks [2] without substantial task-specific architecture modifications. This pretrain-and-finetune paradigm has become a cornerstone of modern NLP.

Despite their power, a limitation of such LLMs is that their knowledge is static: they rely entirely on the data seen during pre-training. Moreover, LLMs have a tendency to hallucinate information—producing plausible but

ungrounded outputs—especially on knowledge-intensive queries. This motivates techniques that combine LLMs with external knowledge sources, leading to the concept of Retrieval-Augmented Generation.

1.2 Retrieval-Augmented Generation (RAG) and Retrieval Techniques

Retrieval-Augmented Generation (RAG) is a framework proposed by Lewis et al. (2020) to address the knowledge limitations of LLMs [3]. A RAG model augments a standard generative model with a retriever component that fetches relevant external documents to ground the generated output in factual information.

In practice, the retriever first finds the top-k pertinent documents from the knowledge source (e.g., Wikipedia or a domain-specific database). These retrieved texts are then provided as additional context to the generator model, which incorporates them to produce a final answer. By grounding generation in fresh, specific data, RAG greatly reduces hallucinations and allows the model to update its knowledge without retraining.

There are two main families of retrieval techniques: sparse and dense. Sparse retrieval relies on exact keyword overlap—BM25 being the most prominent example. BM25 is strong when query terms appear verbatim in relevant documents but struggles with synonyms or paraphrases.

Dense retrieval, in contrast, embeds queries and documents in a high-dimensional vector space so that semantically similar pairs have high cosine similarity. Dense Passage Retrieval (DPR) [5] is a representative method that uses two BERT-based encoders and nearest-neighbor search.

RAG models can employ either approach. Dense retrieval usually offers higher recall for semantic matches, while sparse methods provide efficiency and exact keyword precision; hybrid setups often combine them. In the original RAG paper, the retriever was DPR and the generator was BART. Variants such as RAG-Sequence vs RAG-Token differ in whether retrieval happens once per query or repeatedly during decoding.

Other notable retrieval-augmented models include REALM [6] and Fusion-in-Decoder (FiD) [7]. All share the core idea: marry the broad knowledge of LLMs with the precision of information retrieval—a concept we will leverage for job-market question answering.

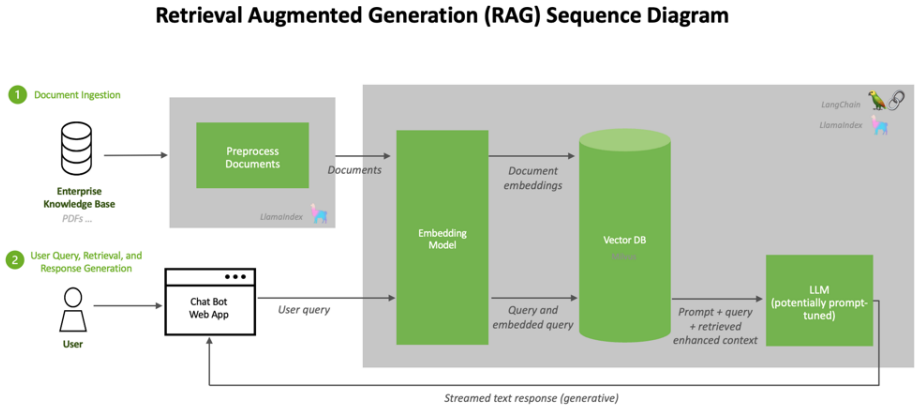


Figure 1.1. Example of a Retrieval-Augmented Generation pipeline [4]

1.3 RAG in our context

In this thesis, our QA system for job postings first retrieves the most relevant job descriptions or statistics from a vector database before generating an answer. For example, when asked "What are common required skills for data scientist positions?", the system fetches data-scientist listings and then prompts the LLM to summarise the skills that appear.

This design grounds answers in real job data rather than the model's general knowledge. RAG pipelines have already proven effective in customer-service chatbots and enterprise search, where trustworthiness and up-to-date information are critical.

Our RAG module thus contains two components: a retriever, which indexes all job postings (as embeddings) and returns the top-k matches for a query, and a *, which takes the user's question plus those retrieved snippets and produces a natural-language answer. Supplying a fixed set of top passages as part of the prompt—a simpler variant used in the original RAG paper—will be adopted here.

1.4 Vector Databases and ANN Indexing

Dense retrieval at scale requires vector databases that can perform fast similarity search over embeddings. A brute-force scan (exact search) is prohibitively slow for millions of vectors, so systems employ Approximate Nearest Neighbor (ANN) algorithms that trade a small loss in recall for large speed gains [8].

- **IVF (Clustering + Inverted File).** Vectors are clustered; at query time only the nearest clusters are searched. IVF is often combined with Product Quantization (PQ) for memory-efficient storage.
- **HNSW (Graph-based).** The Hierarchical Navigable Small World graph [9] builds a multi-layer graph of vectors and performs greedy search, achieving near-exact recall with logarithmic time complexity but higher memory overhead.
- **Other methods.** Tree-based approaches (e.g., Annoy's random-projection trees) and quantization hybrids (e.g., ScaNN) offer different trade-offs.

Modern libraries such as Faiss, ScaNN, Annoy and Milvus implement several of these algorithms, often with GPU acceleration, making real-time semantic search feasible—even for retrieving similar job postings or embeddings in our project. Although we did not implement a full RAG pipeline in this work, vector-based retrieval is a key component of the proposed future extension.

CHAPTER 2

DATA SCRAPING AND EXPLORATORY DATA ANALYSIS

2.1 Data Collection

The dataset for this study was built by scraping job postings from the DOU.ua website – a popular Ukrainian IT jobs portal. We targeted all major job categories on the site (57 distinct categories, ranging from programming languages like Python, Java, C++ to roles like QA, DevOps, Designer, Marketing, etc.).

For each category, the scraping script navigated to the category's listing page and collected the summary of job postings (job title, company, location snippet, and posting URL). Each job's detailed page was then fetched to retrieve the full description and any structured info available (like date posted).

We implemented the scraper using Python (with `requests` and `BeautifulSoup`) and made it robust against common challenges: for example, adding a delay between requests (3 seconds per category loop) to avoid overloading the server or triggering anti-scraping throttling. The scraper also maintained a log and a local CSV to avoid duplicates – if a job URL was already seen in previous runs, it was skipped. This allowed incremental updates to the dataset.

Overall, the raw scrape resulted in a few thousand unique job postings collected over mid-2024. We observed that not all postings include salary information; in fact, a majority of companies do not list a salary on DOU. These missing target values posed a limitation – ultimately, our modeling could only use the subset of postings where salary was specified (approximately 3,134 postings, which is about 12% of the scraped data).

LLM-Assisted Data Processing: After scraping, each job post was essentially an unstructured HTML blob (with a job description, requirements, maybe responsibilities, etc. in prose form). To systematically extract relevant fields, we utilized an LLM (GPT-3.5) via OpenAI's API to parse and structure

the job information.

We engineered a prompt instructing the model to read the job description text and output a structured summary with specific fields: Role, Job Level, Project Description, Responsibilities, Requirements, Additional Points, etc., as well as the Category and Salary if mentioned in text. We explicitly constrained certain fields – for example, Job Level had to be one of a fixed set {trainee, junior, middle, senior, lead, manager}, and Category had to be one from the predefined list of 57 categories. If the description did not state a level or category, the model was instructed to output "Not specified." The prompt ensured the model understood the output format. An excerpt is shown below (translated for clarity):

```
Date: ...
Location: ...
Role: ...
Job Level: [one of: trainee, junior, middle, senior, lead, man
Category: [one of: Python, QA, Design, ... etc.]
Salary: [state the salary if mentioned in the description, oth
Text of the job: "  <full job description text>  "
```

The model's response was then parsed line-by-line to populate a structured record for each job. This approach saved us from writing dozens of ad-hoc text-parsing rules and was quite effective in capturing information such as the "Job Level" (e.g., the model infers junior/middle/senior if the text hints at required experience) and any salary figures buried in the description.

In cases where the model produced an unrecognized job level (say, it output "expert," which is not in our list), we flagged it and treated it as missing. Indeed, about 4.1% of postings ended up with Job Level empty after validation—these were cases where the level was unclear and the LLM's guess was invalid or blank.

The Category field was almost always filled, often matching the category we scraped from (we trust the site's categorization, but using the LLM provided a double-check and normalized naming). The Salary field in the structured output was usually "Not specified," except for those 3 134 posts where a salary actually appeared in the description (the LLM would extract it exactly, e.g., "$2000–3000"). We later parsed those salary strings into numeric values.

Using the LLM in this way significantly streamlined data cleaning. However, we took care to limit the prompt size—only the first 1 500 characters of the job description were sent to avoid hitting token limits. In practice, this usually covered the entire relevant text (very few postings exceeded this length). We also acknowledge a subtle risk: the LLM could hallucinate plausible content that wasn't actually present. To mitigate this, the prompt emphasized "extract," and the model's structure mostly prevents fabrication (it tends to leave fields blank or copy text from input rather than invent details). We spot-checked a number of outputs to ensure fidelity.

In a few instances, the model's classification of Category or Job Level might have been imperfect (e.g., classifying a role as "Developer" under Category when the site listed it under a specific language); we resolved these by trusting the site category if ambiguity arose.

After LLM processing, we obtained a structured dataset with columns: Date, Location, Role, Job Level, Category, Salary, and several descriptive text fields (Project description, Responsibilities, etc.).

Additional cleaning steps included normalizing city names (mapping synonyms like "Kyiv" vs "Kiev" and handling "Remote" as a location) and splitting any multi-city listings (some jobs were open to multiple cities). We derived new features such as a boolean `is_remote` flag, the number of cities mentioned, and one-hot encoding for each city (e.g., a job could have both `city_Kyiv`=1 and `city_Lviv`=1 if it allowed multiple bases).

We also processed the Salary field: for ranges like "$2000–3000," we stored $min = 2\,000$, $max = 3\,000$ and computed the mean (2 500) as the target value. All salaries were converted to a common currency (USD) since the site often uses USD for IT jobs; a few listed in local currency were converted at the current rate for consistency.

Finally, we engineered text features: we created a `combined_text` feature by concatenating the Role title, project description, requirements, etc., to use for any NLP-based modeling. Instead of using the raw combined text directly, we extracted key technical skills as binary features. We compiled a list of over 50 keywords corresponding to programming languages, frameworks, and tools (e.g., Python, Java, JavaScript, AWS, Docker, React, etc.). Each job's combined text

was scanned for these terms (case-insensitive), and a feature like `tech_python` or `tech_docker` was set to 1 if present.

This resulted in additional columns in the dataset (as seen in our final feature set with many `tech_*` columns). These indicators aim to capture the stack or skillset required, which presumably influences salary—for instance, postings requiring hot skills like AWS or Kubernetes might pay more on average.

2.2 Exploratory Data Analysis

2.2.1 Data Overview and Missing Data

The dataset comprises approximately 3,300 unique job advertisements collected from a Ukrainian IT job portal. Each job ad entry includes features such as the posting date, job location, role title, required experience level, textual descriptions (project details, responsibilities, requirements, additional points), a job category, the job posting URL, and the stated salary (if provided). We conducted an exploratory data analysis to understand the structure and quality of this data before modeling.

We first examine the distribution of job advertisements across different locations and experience (job level) categories.

Figure 2.1(a) shows the count of job ads by location. It is immediately evident that Remote positions constitute the vast majority of postings. Roughly three-quarters of the ads are for remote jobs, far outnumbering any single physical location. The next most common location is Kyiv, the capital, which accounts for the largest share of city-specific jobs (on the order of 15–20% of ads). Beyond Kyiv, no other individual city contributes more than a few percent of the total postings. For instance, Lviv (the largest city in western Ukraine) is a distant third with only a small fraction of the ads, and other regional hubs like Odesa and Dnipro have even fewer postings. Many smaller cities (e.g., Poltava, Cherkasy, Chernivtsi, etc.) appear in the dataset but each with negligible counts (often only a handful of ads, if any).

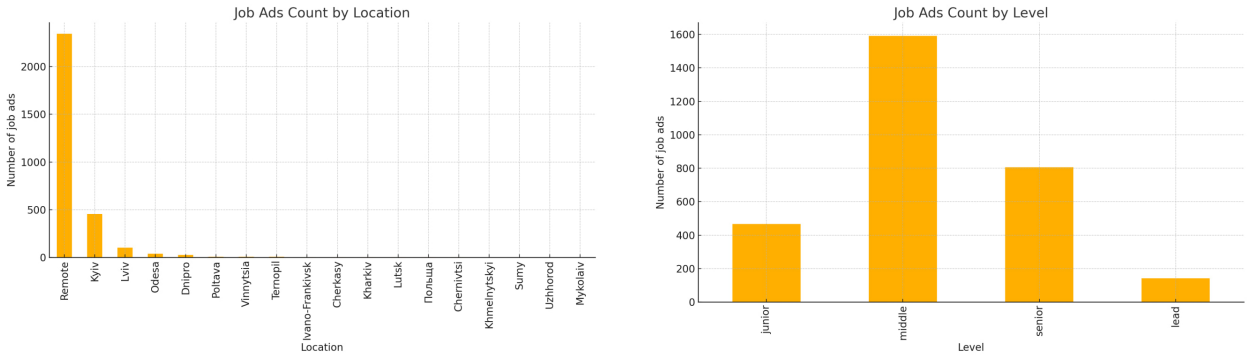
This highly skewed geographic distribution likely reflects both the preva-

lence of remote work (especially amid the recent shift to remote arrangements and the war-related displacement in Ukraine) and the concentration of the tech industry in major cities like Kyiv. In summary, most tech job opportunities in this dataset are not tied to a specific locale, and among those that are, Kyiv dominates while other cities play a minor role.

In the same vein, Figure 2.1(b) illustrates the distribution of job ads by required experience level. We observe a clear imbalance in favor of mid-level positions: Middle-level roles constitute the majority of postings (approximately half of all ads). Senior positions form the second-largest group (roughly a quarter of ads), followed by Junior roles (around 15%). Lead roles (the most senior positions, e.g. team leads or department heads) are comparatively few, comprising only about 5–10% of the postings.

This ordering — Middle > Senior > Junior > Lead in frequency — is intuitive for a job market, as organizations often have more openings for mid-level professionals who have some experience but are not yet at leadership level. The smaller share of junior roles might be due to companies seeking experienced candidates or possibly fewer entry-level openings during the period of data collection. Similarly, the limited number of lead positions aligns with expectations that high-level vacancies are fewer and tend to be filled from within or opened infrequently.

Overall, the experience level distribution highlights that demand peaks at the mid-career level, with progressively fewer openings at the junior and top-leadership ends.



(a) Job ads by location.

(b) Job ads by experience level.

Figure 2.1. Distributions of job advertisements across locations (a) and experience levels (b). Remote jobs constitute the majority of postings, followed by jobs based in Kyiv, whereas all other cities contribute only minor shares. By experience level, Middle positions are most common, outnumbering Senior and Junior roles, with Lead positions being the least common.

2.2.2 Job Ad Distribution by Category

Another important aspect of the data is the distribution of job ads across different job categories or specializations. Figure 2.2(a) presents the top 20 job categories by number of ads in the dataset. These categories represent the functional roles or domains of the jobs (such as specific technical specializations or job functions).

We observe that the distribution by category is moderately skewed: a few categories comprise a large share of the postings, while many others have relatively small counts. The single most populous category is **Sales**, with around 200 job ads, indicating a significant demand for sales-related roles (likely technical sales or IT product sales) in the tech sector.

The next most common category is labeled **Other** (nearly 200 ads as well), which serves as a catch-all for postings that do not fit into a well-defined category. The prominence of an "Other" category suggests that a considerable number of job ads did not belong to the standard set of categories, underscoring a diversity of roles or perhaps limitations in the categorization scheme.

Following Sales and Other, prominent specific tech categories include **Front End** development and **PHP** development, each with roughly 150–180

ads. **AI/ML** (Artificial Intelligence/Machine Learning) roles also appear in the top five, reflecting the high demand for AI/ML specialists (around 150 ads).

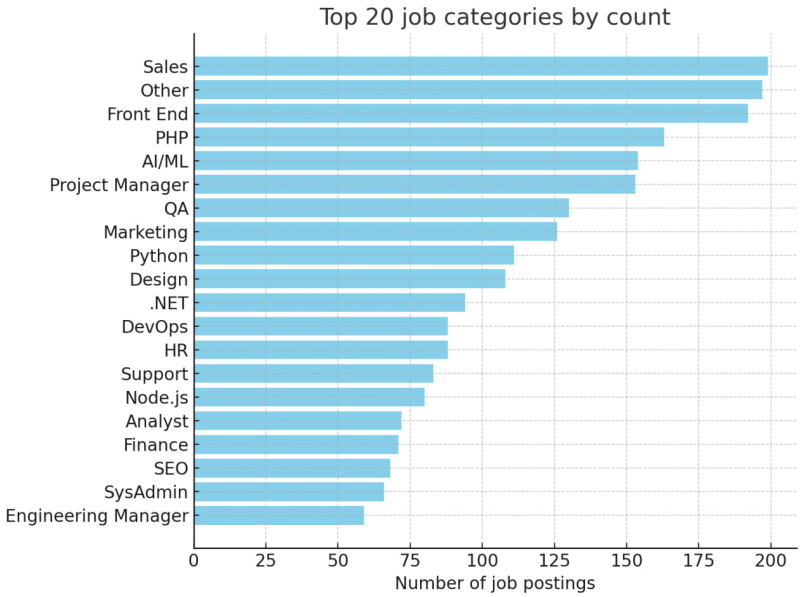
Beyond these top categories, the counts drop off for the 6th to 20th ranked categories, which include various specialized roles (such as backend languages, QA, DevOps, Project Management, etc.). Each of those tends to have on the order of tens to low hundreds of ads. The long tail of the distribution implies that while there are many different role categories represented (dozens in total), most of them have relatively few postings compared to the top few.

To emphasize the dominance of the leading categories, Figure 2.2(b) zooms in on the top five categories alone. These top five (Sales, Other, Front End, PHP, AI/ML) account for a substantial portion of the dataset. In fact, combined they make up roughly a quarter to a third of all job ads.

This concentration indicates that the job market (at least as captured by our data source) is somewhat dominated by a few broad role types. For example, the high ranking of Sales positions suggests that tech companies in Ukraine are hiring not just technical staff but also a significant number of sales and business-development professionals to drive product outreach. The inclusion of Front End and PHP in the top tier reflects the continued demand for web-development skills, whereas the strong showing of AI/ML roles highlights the growing importance of data science and machine-learning expertise.

Meanwhile, the large "Other" category in the top five reinforces that many postings did not cleanly fit into the predefined categories – these could be niche roles or multi-disciplinary positions. Overall, the category-distribution analysis shows a bimodal pattern: a handful of categories dominate the postings, but there is also a wide variety of less-common roles represented.

This insight is useful for feature engineering in our prediction task: we may need to account for very sparse categories or consider grouping infrequent categories (although the presence of the "Other" category already groups many of the rare ones together). It also indicates that any salary-prediction model should be aware of the job category, as certain categories (e.g., AI/ML vs. Sales) likely have different salary ranges.



(a) Top 20 job categories by number of ads.

Figure 2.2. Distribution of job advertisements across categories. Panel (a) shows the frequencies of the top 20 job categories. A few categories (notably Sales, Other, Front End, PHP, AI/ML) have the highest counts, while most other categories have substantially fewer postings.

Salary Distribution by Experience Level and Location

(a) Salary by experience level.

(b) Salary by job location.

Figure 2.3. Salary distributions (monthly, in USD) broken down by experience level (a) and by job location (b).

2.2.3 Salary Distribution

Finally, we consider the overall distribution of salaries in the dataset, disregarding the categorical groupings. Figure 2.4 displays a histogram of the mean salary values across all job ads excluding extreme outliers. (For ads that

provided a salary range, the mean of the range was used, and for this analysis we filtered out a few exceptionally high salaries above the typical range to focus on the bulk of the data.)

The histogram reveals that the salary data is right-skewed. Most job ads cluster in the lower salary ranges: there is a high concentration of salaries roughly between $500 and $2,000 USD. The tallest bars (the modal range) occur around the $800–$1,200 mark, indicating that this is a common monthly salary interval for many postings. Beyond about $2,000, the frequencies taper off, illustrating that higher salaries are progressively less common.

There is a long tail extending toward the right (higher salary values up to around $6,000 in the plotted range), showing that while rare, there are positions offering several times the median salary. The exclusion of outliers (e.g., salaries above roughly $6k–$8k) in this figure makes the central pattern clearer—without those extreme values, the bulk of the data is easier to discern. Even without the top outliers, the range of salaries remains quite broad (spanning from near $0 or a few hundred dollars up to around $6,500 in the figure).

The skewness of the distribution implies that the mean salary is higher than the median salary. In fact, the median salary lies near the lower end of the high-frequency range (likely on the order of $1,200–$1,500), whereas the mean is pulled upward by the relatively few high-paying positions. The overall spread (standard deviation) is large due to this long tail of high salaries.

We also note a slight secondary bump in the histogram around $2,500–$3,000, which could hint at a subset of more senior or specialized roles forming a higher-paying cluster distinct from the main group around $1,000. In practical terms, this distribution suggests that most tech job openings in Ukraine offer modest salaries, but there is a non-negligible segment of jobs (typically senior, lead, or internationally-oriented roles) that offer significantly higher pay.

This variability will pose a challenge for salary prediction, as the model must handle a wide range of possible outcomes. It also reinforces the need to manage outliers carefully: extremely high salaries could unduly influence certain modeling techniques, so one might consider capping or separately modeling the top end of the distribution if appropriate. Overall, the salary histogram provides a high-level view that complements the earlier boxplot analysis, confirming the

positive skew and broad range of salaries in the dataset.

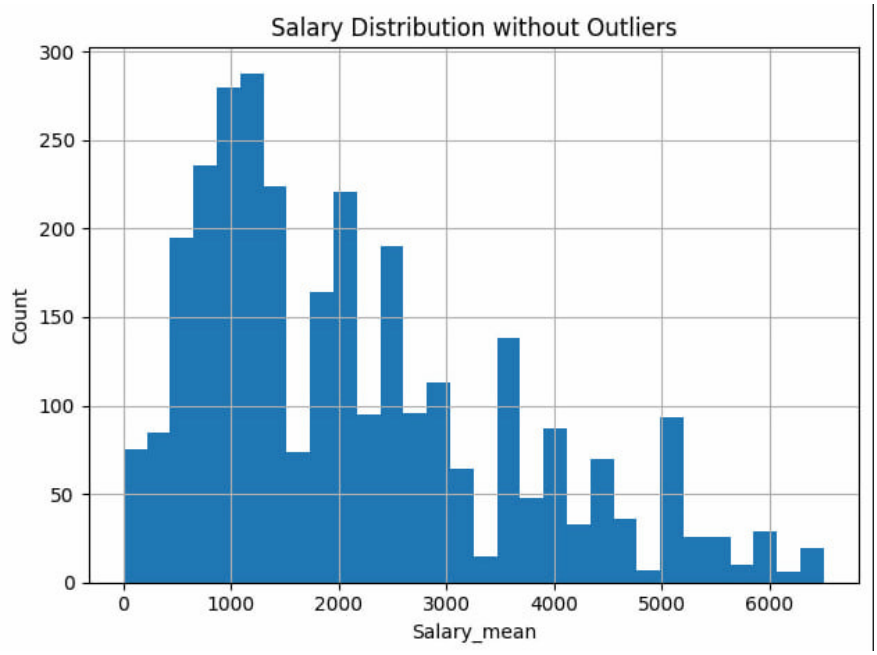


Figure 2.4. Histogram of the distribution of job advertisement salaries (monthly USD), excluding a few extreme outlier values for clarity. The distribution is right-skewed, with the majority of salaries falling between a few hundred and a few thousand dollars. A peak is visible around $1k, and the frequency gradually declines toward the higher end of the range. Even after removing outliers above the upper $6k range, a long tail of relatively high salaries remains evident.

2.2.4 Summary of Insights

The exploratory data analysis reveals several key characteristics of the Ukrainian tech job ads data. The dataset is dominated by remote jobs and Kyiv-based jobs, indicating that location (and the flexibility of remote work) is a major factor in how the job market is structured. The demand by experience level is heavily skewed towards mid-level professionals, with far fewer junior and lead positions advertised. We also found that job ads span a wide array of categories, though a handful of categories (Sales, certain developer roles, etc.) account for a disproportionate share of postings. Crucially for the salary prediction task, salaries in the dataset exhibit a high variance and skew – most jobs offer salaries in the low-to-mid range, but there are notable high-end outliers. The median salaries differ markedly by experience level and location, affirming that these features should be strong predictors in any modeling effort. Additionally,

the prevalence of missing salary data (around 65%) establishes a clear need for predictive modeling to fill in those gaps, while the patterns observed (e.g., higher medians for seniors, remote roles, certain categories) provide intuitive guidance for feature relevance. These EDA insights set the stage for the modeling work to follow, by highlighting which factors are likely important and what challenges (such as skewed distributions and missing values) the model must be designed to handle.

CHAPTER 3

EXPERIMENTAL RESULTS

The core prediction task of this project was to predict the monthly salary (in USD) for a given job posting, based on the information available at the time of posting. We treated this as a regression problem. Our target variable was the numeric salary (post-processed to a single representative value in USD per month). We experimented with a wide range of regression models and techniques, iterating to improve performance.

Models and Techniques Tried: We initially cast a wide net, trying everything from simple linear models to advanced ensemble methods. The models we evaluated included:

- Linear Models: Basic regression algorithms like Ordinary Least Squares Linear Regression, Ridge, and ElasticNet (which add regularization to linear models).
- Tree-Based Ensembles: Several powerful nonlinear models such as Random Forest Regressor, XGBoost, LightGBM, and CatBoost. These models can capture complex interactions and often perform well with minimal tuning, especially CatBoost which is robust with categorical features.
- Neural Network: A simple feed-forward PyTorch neural network was built to see if a multi-layer perceptron could capture patterns in the data.
- Ensembles: We also attempted ensemble strategies like stacking (combining multiple model predictions via a meta-model) and averaging ensembles (averaging predictions from several models) to see if a combined approach would yield better accuracy.

Each of these models was trained on the dataset with various feature representations. Importantly, we tried both using the raw textual features (via embeddings or vectorization) and dropping them to see impact. We also split the data into training and testing sets in a time-aware manner – using the latest 20% of postings by date as a hold-out test set, so that the model is always evaluated on "future" data relative to its training data (simulating how it would predict for new jobs coming in later).

3.1 Performance Across Models

The prediction performance varied widely across these experiments. We used Mean Absolute Error (MAE) in USD as the primary evaluation metric (lower is better). Simpler models like basic linear regression, without careful feature processing, struggled – some yields were on the order of MAE approximately $3000 (meaning predictions were off by $3000 on average, which is very high given many salaries themselves were in the $2000–4000 range). This was essentially no better than a naive guess in those cases. On the other hand, some more sophisticated models got the error down to around $700–800 with default settings. We noticed that models like Random Forest and XGBoost started off around the mid-range (MAEs in the $1000+ range) and could be improved slightly with tuning. CatBoost, in particular, showed promising results out-of-the-box (sub-$1000 MAE) given its ability to natively handle categorical variables and its robustness.

Table 3.1. Salary-prediction performance across models (lower MAE is better)

Model	**Setting**	**MAE (USD)**
Linear Regression	basic features	$\approx 3\,000$
Random Forest	default	$\approx 2\,000$
Random Forest	tuned	$\approx 1\,500$
XGBoost	default	$\approx 2\,000$
XGBoost	tuned	$\approx 1\,600$
CatBoost	default	≈ 850
CatBoost	tuned	≈ 750
CatBoost	**Best Pipeline**	**144.6**

3.2 Experiment Iterations and Findings

During experimentation, we attempted several approaches to improve performance:

- Using the full dataset vs. filtered data: Initially we trained on all data including entries with extremely high salaries or possibly noisy entries. We found that removing or capping outliers (e.g., dropping the top 1% of salaries) was sometimes helpful, as it reduced the distortion of the error metric and made models focus on the main patterns. In some runs, trimming these outliers improved MAE on the rest of the data by a notable margin.
- Feature normalization and transformation: We experimented with scaling features (like using Min-Max scaling or Standardization on numeric features such as years of experience if that had been extracted, etc.). We also tried different representations for the target variable – one successful strategy was to model the log of the salary (since salary was skewed). Taking log1p(salary) as the target made the distribution more normal and many models then performed better (this needs an inverse transform on predictions, and we evaluated error back in USD).
- Hyperparameter tuning: We manually tried adjusting key hyperparameters for the tree-based models – e.g., the number of trees, tree depth, learning rate for boosting models, and regularization parameters. For instance, reducing the learning rate in XGBoost and increasing the number of estimators helped a bit, and for CatBoost we tried various tree depths. However, manual tuning was hit-and-miss; some models improved, others overfit or showed only minor gains. A systematic grid search or Bayesian optimization was considered but given the computational time we did limited manual tuning due to resource constraints.
- Text feature processing: Early on, we tried incorporating the textual job description fields by using pre-trained sentence embeddings (turning each job's description into a vector of numbers) and concatenating these with other features. This led to very high dimensional data and some models (like linear ones) overfit dramatically, while others didn't improve much. We realized that a more structured approach to text was needed (either

reduce dimensions or select important phrases). Later, we moved to using TF-IDF vectorization on the text (to quantify important words) followed by SVD (Singular Value Decomposition) to reduce this to a smaller set of latent text features. This approach gave us a handful of numeric features capturing the gist of the job description, which proved far more effective than using hundreds of raw embedding dimensions.

Through these iterations, we identified that the most consistent and impactful improvements came from better preprocessing and feature engineering, rather than from blindly trying different model types. In other words, how we prepared the data had a larger effect on accuracy than which algorithm we used (provided the algorithm was reasonably powerful). This led us to design a final modeling pipeline incorporating all the best practices we learned.

3.3 Best Model Pipeline

The best result was achieved with a carefully constructed pipeline using **CatBoostRegressor** as the final estimator. Key elements of this winning approach were:

- Target Transformation: We applied a log-transform to the salary target (log1p) during training. The CatBoost model then predicted log-salary, and we exponentiated its predictions to get back to USD. This stabilized the training and gave more weight to getting the mid-range salaries right, rather than being overly influenced by the few very high salaries.
- Feature Processing Pipeline: We used a ColumnTransformer to handle different feature types in parallel. For textual features (like the job's Responsibilities/Requirements text, or the Job Title), we applied a TF-IDF vectorizer and then truncated it using SVD to around 50 components, creating compact text features. For structured categorical features (such as Job Level, Category, Location), we took advantage of CatBoost's ability to handle categorical data natively by encoding them as category indices (or in some cases using one-hot encoding via the transformer for comparison). For any numeric features , we scaled them appropriately. This unified pipeline ensured that all preprocessing (text vectorization, dimensionality

reduction, encoding) was done within cross-validation folds and on the fly, preventing any data leakage.

- CatBoost Hyperparameters: We found an optimal setting for CatBoost with a tree depth of 8, a modest L2 regularization term of 3 to prevent overfitting, and used early stopping during training (monitoring a validation set) to stop when improvement plateaued. CatBoost was a good choice because it can utilize categorical features through techniques like target encoding internally and is robust with heterogeneous data. It also tends to be less sensitive to scaling and handle missing values well, which simplified our pipeline.
- Time-Aware Split for Validation: As mentioned, we reserved the latest 20% of the data (by date) as a hold-out test. All model tuning and selection was done via cross-validation on the earlier portion of data, making sure not to leak future information backward. The final chosen pipeline was then retrained on the full training set (80% of data) and evaluated on the hold-out. This mimics a realistic scenario where we train on past data to predict future salaries.
- Performance of Final Model: The CatBoost pipeline delivered a Mean Absolute Error of approximately $144.60 USD on the hold-out test set. This was a dramatic improvement compared to our initial models. An MAE of $145 means that, on average, our salary predictions were off by only $145, which is a very small error relative to typical salaries in the data (often 5–10% of the actual salary value). In practical terms, the model can predict, say, a $2000 salary and be within ±$150 of the true value, which is quite precise for this domain.

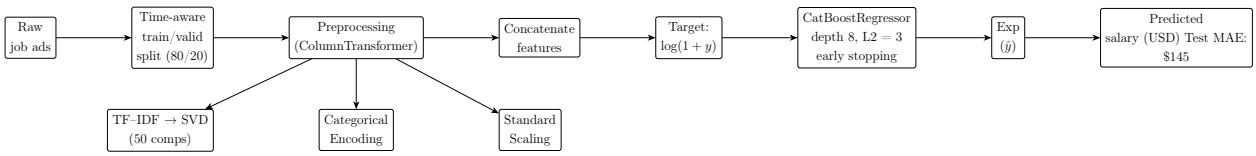


Figure 3.1. Best pipeline schema

To summarize the key insights from our experiments: CatBoost-based pipeline, with proper handling of categorical and text data, significantly outperformed models that either ignored the text or handled it poorly. By feeding the model both the structured information (levels, categories, etc.) and distilled textual

information (TF-IDF/SVD features), we enabled it to learn a more complete picture. CatBoost's built-in handling of categorical variables (like Job Level or Location) was particularly advantageous, as it preserved ordinal relationships and high-cardinality categories without blowing up the feature space.

The distribution of salaries was highly skewed, and using the logarithm of salary for training improved model stability. Predictions in log-space reduced the influence of extreme high values and resulted in better overall MAE when converted back. This technique acted almost like an automatic outlier moderation and gave the model a chance to optimize relative errors more evenly across the range of salaries.

We discovered that removing or capping outlier salaries (very rare extremely high values) and treating text features separately from tabular features were critical steps. If we did not remove outliers, some models would chase those points and be less accurate on the majority. Similarly, mixing raw text-derived features with numeric ones without scaling or reduction caused issues; explicitly separating and reducing text features (through SVD) prevented the curse of dimensionality from harming the model.

Most other models we tried either overfit (for example, the neural network started memorizing the training data given its flexibility, and ensembles like Random Forest would sometimes fit noisy patterns) or underperformed (for instance, linear models couldn't capture nonlinear interactions). Even sophisticated models like XGBoost did not surpass CatBoost in our case, likely due to CatBoost's better handling of our mix of features and the careful tuning and pipeline we constructed around it. Ensembling multiple models did not yield significant gains once we had the strong CatBoost pipeline; a simple approach with one well-tuned model and good preprocessing outshined complex ensembles on this problem.

CONCLUSION

- **Project Accomplishments**: In this project, we collected and processed a comprehensive dataset of Ukrainian IT job postings from DOU. We applied an LLM to transform unstructured job descriptions into structured data (extracting fields like seniority level and key requirements), performed exploratory data analysis to understand the job market, and built a series of machine learning models to predict salaries. The entire pipeline included data scraping, cleaning, enrichment with AI, and rigorous model training/validation. We also developed an interactive component using a vector database and LLM (via Retrieval-Augmented Generation) to enable question-answering on the dataset, demonstrating a novel way to query and interact with the data beyond static analysis.
- **Key Findings and Results**: Our EDA revealed several important trends in the Ukrainian IT job market, such as a heavy concentration of openings for mid-to-senior level roles and significant variation in salaries across different cities and specialties. We also saw that salaries for IT roles, while varied, generally increase with experience level—a fact quantifiable through our analysis. On the modeling front, our best predictive model—a CatBoost regression pipeline with tailored preprocessing—achieved a Mean Absolute Error of around $144.60 USD on the test set. This high level of accuracy indicates that the model can serve as a useful tool for estimating market salaries. It also underscores the effectiveness of combining textual and structured features in salary prediction: knowing the job level, category, and some description of the role allows for very close salary estimates.
- **Future Work**: There are several avenues to further improve and extend this work. Firstly, more data and labels could enhance the model—for example, incorporating additional features like company size or specific skill requirements (if available) could provide more signal. If we could obtain a larger dataset or continuously update it, the model could stay current with market trends. Secondly, exploring larger or more complex models might yield improvements: for instance, a specialized deep learning model that utilizes transformer-based text embeddings (like BERT or a domain-specific language model) might capture

nuances in job descriptions better than TF-IDF. Related to this, using better text embeddings or language model features could improve how we represent the job requirements and responsibilities in the prediction model. Thirdly, implementing a robust cross-validation scheme (such as a time-series cross-validation that respects temporal order) would ensure the model's performance is consistent and not just a one-off on a particular split; this would make the results more reliable. Finally, on the interactive side, we can enhance the LLM+RAG system by enriching the knowledge base (for instance, adding historical statistics or summary data to the vector store) and by fine-tuning the prompts or using a more advanced LLM. This would allow end-users (recruiters, job seekers, analysts) to ask even more complex questions and get accurate, data-backed answers.

ВИСНОВОК

- **Результати роботи**: У межах цього проєкту ми зібрали та обробили вичерпний набір даних українських IT-вакансій із DOU. Ми застосували велику мовну модель для перетворення неструктурованих описів вакансій у структуровані дані (витягуючи такі поля, як рівень seniority та ключові вимоги), провели розвідувальний аналіз даних для вивчення ринку праці та побудували низку моделей машинного навчання для прогнозування зарплат. Уся конвеєрна обробка включала веб-скрапінг, очищення даних, обробку за допомогою LLM та ретельне навчання й валідацію моделей. Ми також розробили інтерактивний компонент із використанням векторної бази даних і великої мовної моделі (за допомогою Retrieval-Augmented Generation), що дозволяє задавати питання щодо набору даних, демонструючи інший підхід до взаємодії з даними.
- **Основні висновки та результати**: Розвідувальний аналіз даних виявив кілька важливих тенденцій на українському IT-ринку праці: значну кількість вакансій на рівні mid-to-senior, а також істотну варіативність зарплат у різних містах і спеціалізаціях. Ми також побачили, що зарплати в IT, хоч і різняться, загалом зростають із рівнем досвіду — факт, який вдалося кількісно підтвердити. Що стосується моделювання, наша найкраща модель — конвеєр на базі `CatBoostRegressor` із лог-трансформацією цільової змінної, TF-IDF + SVD для текстових ознак та вбудованою обробкою категоріальних полів — показала середню абсолютну помилку близько $144,60 USD на тестовій вибірці. Така точність свідчить про те, що модель може слугувати корисним інструментом для оцінки ринкових зарплат і підкреслює ефективність поєднання текстових і структурованих ознак у прогнозуванні: знання рівня позиції, категорії та часткового опису вакансії дозволяє отримувати дуже близькі оцінки до істинних .
- **Перспективи подальшої роботи**: Існує кілька напрямів для подальшого вдосконалення та розширення цієї роботи. По-перше, більше даних і додаткові мітки можуть покращити модель — наприклад, включення таких ознак, як розмір компанії чи конкретні вимоги до навичок (за наявності), може надати додатковий сигнал. Отримання більшого набору даних або

його безперервне оновлення дозволило б моделі залишатися актуальною щодо ринкових тенденцій. По-друге, дослідження більших або складніших моделей може дати покращення: наприклад, спеціалізована глибока модель, що використовує трансформерні текстові ембеддинги (наприклад, BERT або домен-специфічну мовну модель), може краще захоплювати нюанси опису вакансій, ніж TF-IDF. Використання покращених текстових ембеддингів або можливостей сучасних мовних моделей може підвищити якість представлення вимог і обов'язків у моделі прогнозування. По-третє, впровадження надійної схеми крос-валідації (наприклад, часової крос-валідації з урахуванням хронологічного порядку) забезпечить стабільність продуктивності моделі та уникне випадкових результатів на окремій підмножині; це зробить результати більш надійними. Нарешті, з точки зору інтерактивності, можна покращити систему LLM+RAG, розширивши базу знань (наприклад, додавши історичну статистику чи підсумкові дані у векторну базу) та відшліфувавши підказки або використавши більш просунуту велику мовну модель. Це дозволить кінцевим користувачам (рекрутерам, шукачам роботи, аналітикам) ставити ще складніші запитання та отримувати точні, обґрунтовані відповіді.

BIBLIOGRAPHY

1. Transformer: A Novel Neural Network Architecture for Language Understanding August 31, 2017 Posted by Jakob Uszkoreit, Software Engineer, Natural Language Understanding https://research.google/blog/transformer-a-novel-neural-network-architecture-for-language-understanding/: :text=Neural
2. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova 11 Oct 2018 https://arxiv.org/abs/1810.04805: :text=stands
3. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, Douwe Kiela 22 May 2020 https://arxiv.org/abs/2005.11401
4. Understanding RAG: Beyond Basic AI Implementation https://www.visioneerit.com/blog/understanding-rag-beyond-basic-ai-implementation
5. Dense Passage Retrieval for Open-Domain Question Answering Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, Wen-tau Yih 10 Apr 2020 https://arxiv.org/abs/2004.04906
6. REALM: Retrieval-Augmented Language Model Pre-Training Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, Ming-Wei Chang 10 Feb 2020 https://arxiv.org/abs/2002.08909
7. Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering Gautier Izacard, Edouard Grave April 2021 https://aclanthology.org/2021.eacl-main.74/
8. Faiss: A library for efficient similarity search By Hervé Jegou, Matthijs Douze, Jeff Johnson MARCH 29, 2017 https://engineering.fb.com/2017/03/29/data-infrastructure/faiss-a-library-for-efficient-similarity-search/: :text=above
9. Efficient and robust approximate nearest neighbor search using Hierarchical

Navigable Small World graphs Yu. A. Malkov, D. A. Yashunin 30 Mar 2016 https://arxiv.org/abs/1603.09320