Salary prediction based on job description in the IT sector

Dmitrii Shiriaev, Eugene Romanov, George Besedin

December 2024

Abstract

Setting an optimal salary for a job opening is a complex pipeline which requires careful market analysis and understanding the nature of the job to be conducted. Here, we addressed the problem of predicting the salary within the IT sector based on job descriptions and related features by collecting and preprocessing the data from the job posting platforms, setting up baselines and experimenting to obtain the best results. We tried a diverse panels of approaches, including CatBoost, RNN/CNN- and Transformer-based methods. The best result with R^2 score of 0.770 was achieved by the custom setup incorporating CatBoost predictions with the predictions of the transformer model whose learning is enhanced by using Huber loss. Our project code is available on GitHub: `https://github.com/dm-shr/tech-salary-prediction`.

1 Introduction

The digital transformation has fundamentally reshaped the modern job market, particularly in the information technology (IT) sector. Despite technological advances, significant challenges persist in the job search process, with salary transparency emerging as a critical issue. Many companies withhold salary information for competitive or privacy reasons, creating an information asymmetry between employers and job seekers. While some emerging job platforms mandate salary disclosure, large recruitment websites, which host the majority of vacancies, still allow optional salary disclosure.

Recent academic literature demonstrates growing interest in addressing this transparency gap through data-driven solutions. Researchers have increasingly applied machine learning techniques and neural networks to predict salary ranges based on job descriptions and related parameters. Here, we contribute to this evolving field by developing and evaluating various salary prediction approaches specifically for the IT sector in Russia.

The primary objective of this research is to develop accurate salary prediction models based on job descriptions and related characteristics in the IT sector. To achieve this goal, the following objectives have been established:

- Review the related work and summarize available approaches;
- Define the problem and construct an appropriate dataset;
- Develop predictive models and compare them with existing baselines.

The study utilizes job vacancy data collected from HeadHunter (`https://hh.ru/`) and Getmatch (`https://getmatch.ru/`), leading IT job platforms in Russia, spanning November to December 2024 and focusing exclusively on the IT sector. All implementations were conducted using Python programming language, leveraging its robust ecosystem of data science and machine learning libraries. The project has four major parts:

1. a review of existing approaches in salary prediction field;
2. dataset collection, preprocessing, and analysis of key patterns;
3. setting up baselines and implementing experiments;
4. comparative analysis of models performance.

Code for the deployed model is publicly available on GitHub: `https://github.com/dm-shr/tech-salary-prediction`.

Our team consist of:

- **Dmitrii Shiriaev**:
  - Backend / Frontend development
  - Deployment
  - Data / ML / inference pipeline
  - Transformer experiments
  - CatBoost experiments
- **Eugene Romanov**:
  - exploratory data analysis (EDA)
- **George Besedin**:
  - Bi-GRU-CNN baseline

All authors contributed in writing the report manuscript.

2 Related Work

The field of salary prediction using machine learning has evolved significantly in recent years, with approaches ranging from traditional statistical methods to advanced deep learning techniques. This section reviews key contributions across different methodological approaches and application domains.

2.1 Transformer-Based Approaches

Recent work has demonstrated the effectiveness of transformer architectures for numerical prediction tasks. Notably, Le et al. [Le and Dupont, 2022] explored the application of BERT models for regression tasks, comparing multiclass classification versus direct numerical prediction approaches. Their work on sentiment score prediction demonstrated that fine-tuning BERT with a regression head can achieve comparable results to classification-based approaches while providing more granular predictions.

Building on this foundation, Shilo and Horev [Shilo, 2022] applied transformer models specifically to salary prediction, showing significant improvements over traditional TF-IDF based approaches. Their implementation demonstrated that contextual embeddings from transformers can effectively capture subtle nuances in job descriptions that correlate with salary levels.

However, some research suggests potential limitations of transformer architectures for regression tasks. A comprehensive study by Zhang et al. [Zhang et al., 2024] found that "Transformers are bad at approximating smooth functions" but excel at piecewise constant approximations. This finding has important implications for salary prediction, suggesting that hybrid approaches might be more effective for capturing both discrete and continuous aspects of salary determination.

2.2 Other Neural Network-Based Approaches

Within the subset of non-Transformer neural network-based methods to predict salaries based on the job description, RNN and CNN architectures are the most widely used ones. Bonnie et al. [Ma, 2020] explored the use of combined LSTM- and CNN-based models and demonstrated accuracy of 0.515 on the task of predicting the correct salary bin class. A somewhat similar setup was implemented by [Sergeeva, 2024], using Bi-GRU-CNN (Bidirectional - Gated Recurrent Unit - Convolutional Neural Network) architecture for the regression task, achieving state-of-the-art results with an R^2 score of 0.64.

2.3 Traditional Machine Learning Approaches

Several studies have explored traditional machine learning approaches for salary prediction. Bonnie et al. [Ma, 2020] evaluated multiple classical models including Random Forests and SGD classifiers with various feature engineering approaches, finding that TF-IDF with Random Forest achieved the highest accuracy (0.71) for salary range prediction. Similarly, Krishujeniya [Krishujeniya, 2023] demonstrated success with Random Forest Regressors, achieving an R^2 score of 0.9385 on a comprehensive dataset of data professional salaries.
Another popular choice for the salary prediction tasks is the family of frameworks based on gradient boosting on decision trees. For instance, a work by Sergeeva [Sergeeva, 2024] explores the use of CatBoost, gradient boosting framework specialized on dealing with the highly categorical features, for salary prediction within the IT sector, obtaining an R^2 score of 0.62.

2.4 Hybrid and Domain-Specific Models

Recent work has begun exploring hybrid approaches that combine different methodological strengths. Mumminen [Mumminen, 2023] demonstrated that clustering the data before applying regression models could significantly improve prediction accuracy. This approach acknowledges the inherent segmentation in job markets and salary structures.

2.5 Time Series and Sequential Aspects

While most approaches treat salary prediction as a static problem, some researchers have explored temporal aspects. Li et al. [Li et al., 2020] introduced novel transformer architectures for time series forecasting that could potentially be adapted for salary trend prediction, achieving superior results compared to traditional time series models.

This review of the literature suggests that while transformer-based approaches show promise, the most effective solutions may require combining multiple techniques to address the complex nature of salary prediction in the IT sector. Our work builds upon these findings while addressing the specific challenges of the Russian IT job market.

3 Model Description

In our experiments, we employed the following stack to solve the salary prediction problem based on job descriptions in the IT sector:

- CatBoost as a baseline gradient boosting model;
- ruBERT-tiny-turbo as a baseline transformer model;
- Bi-GRU-CNN architecture as a baseline RNN/CNN model;
- Enhanced transformer-based approaches with various modifications. For that, ruBERT and multilingual-e5 architectures were used.

3.1 CatBoost

CatBoost is an advanced implementation of a decision tree-based gradient boosting approach developed by Yandex. The model effectively handles both numerical and categorical features, making it particularly suitable for processing structured job description data. CatBoost's key advantages include its ability to automatically handle categorical features and its robust handling of text data through built-in tokenization and embedding mechanisms.

CatBoost builds upon standard gradient boosting by using an ordered boosting scheme to prevent overfitting. Instead of calculating gradients using a single model for all examples, it maintains separate models based on random permutations of the training data. For categorical features, CatBoost uses an ordered

target encoding approach - it calculates statistics using only the data points that come before the current example in a random permutation, which reduces information leakage during training.

The framework's performance benefits from its use of symmetric trees, where the same splitting criteria are used across each tree level. This design choice speeds up both training and prediction while maintaining model stability. CatBoost also includes built-in text processing capabilities through tokenization and embedding mechanisms, making it particularly effective for tasks involving both structured and text data.

3.2 ruBERT

As our baseline transformer model, we employed a variation of ruBERT (sergeyzh/rubert-tiny-turbo) pre-trained on Russian and English texts. Its small size of 29M parameters and reduced inference time makes the model an attractive platform for the experiments when GPU resources are limited. This model was fine-tuned for our regression task using Mean Squared Error (MSE) loss. The architecture maintains the core BERT principles while being optimized for computational efficiency. We utilized this model to extract meaningful features from job descriptions, considering the specific nature of IT-sector terminology in Russian language contexts.

3.3 Bi-GRU-CNN Architecture

The Bi-GRU-CNN architecture follows the implementation proposed by [Sergeeva, 2024], combining recurrent and convolutional neural networks for processing job descriptions. The architecture consists of three main components:

1. **Text Embedding**: Converts input text into dense vector representations, capturing semantic relationships between words using pre-trained embeddings.

2. **Sequential Processing**: A bidirectional GRU layer processes the text in both directions, capturing the context from both past and future tokens in the sequence.

3. **Feature Extraction**: A convolutional layer with pooling extracts and combines local patterns from the processed sequence for final prediction.

Figure 1: Architecture of the Bi-GRU-CNN model for salary prediction, based on [Sergeeva, 2024].

The model includes regularization techniques and a comprehensive text preprocessing pipeline to handle Russian language input.

3.4 multilingual-e5

Having chosen the best performing setups with *rubert-tiny*, we additionally evaluated them using *intfloat/multilingual-e5-small* transformer architecture with 118M parameters. This model, pre-trained on a large multilingual text pair dataset [Wang et al., 2024], originates from E5 model (Embeddings from bidirectional encoder representations). The latter is based on the shared biencoder architecture and was trained to differentiate between relevant or irrelevant/negative sentence pairs using their pooled representations [Wang et al., 2022], which, in our opinion, makes it a solid choice for a task of regression over the sentence embeddings.

3.5 Advanced Transformer Experiments

Building upon the baseline transformer model, we conducted several experiments with modified architectures and training approaches, including:

- Experimentation with different loss functions beyond MSE;
- Exploration of domain adaptation techniques beyond classical fine-tuning;
- Introduction of an extra cross-attention block between textual features.

We cover these and other approaches in more detail in the Experiments section.

4 Dataset

4.1 Data information

This analysis explores the landscape of IT and tech-related job vacancies in Russia, using data collected from two major job platforms: HeadHunter (hh.ru) and GetMatch. HeadHunter is Russia's largest job search platform, while GetMatch specializes in IT recruitment and tech positions. The data was collected

through custom API parsing scripts located in the `data-collection` folder of the GitHub repository, utilizing official HeadHunter API for HeadHunter data and web scraping for GetMatch data. The dataset covers the period of October-November 2024 and contains detailed information about various tech positions, including salary ranges, required skills, locations, and company details. Our parsing mechanism ensured the collection of both structured data (like salary ranges, experience requirements) and unstructured data (such as full job descriptions and required qualifications), providing a comprehensive snapshot of the Russian IT job market during this period. To obtain a copy of the dataset, follow the manual on the main page of the repository.

- **Data Sources:** HeadHunter (hh.ru) and GetMatch job platforms
- **Time Period:** December 2024
- **Geographic Coverage:** Various cities across Russia
- **Main Features:**
  - **Title:** The job title or position being advertised.
  - **Location:** The geographic location of the job, including city or region.
  - **Company:** Information about the hiring company or employer.
  - **Skills:** Specific skills or technologies required for the position.
  - **Description:** A detailed description of job responsibilities and requirements.
  - **Salary_from:** The lower bound of the salary range offered (target).
  - **Salary_to:** The upper bound of the salary range offered.
  - **Currency:** The currency in which the salary is specified (e.g., RUB, USD).
  - **Experience_from:** Minimum years of work experience required.
  - **Experience_to:** Maximum years of work experience considered.

4.2 Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is a crucial step in understanding and summarizing the key characteristics of a dataset. It helps to uncover hidden patterns, detect anomalies, and derive meaningful insights that can guide further analysis or modeling. In this study, EDA is applied to the dataset of IT job vacancies in Russia to analyze job titles, salary ranges, required skills, experience levels, and other key features. The goal is to identify trends and relationships that can provide a deeper understanding of the IT job market landscape.

The insights from this analysis can be valuable for:

- **Job seekers:** Understanding market demands and salary expectations.

- **Companies:** Building competitive compensation packages.
- **HR professionals:** Developing effective talent acquisition strategies.
- **Educational institutions:** Adapting their curricula to align with market needs.
- **Economic researchers:** Studying trends in the tech sector in Russia.

Let's dive into the data and uncover the patterns and trends in the Russian IT job market.

To begin our analysis, we first explore the basic information in the dataset to understand its structure and quality. This preliminary examination provides insights into the completeness, reliability, and representativeness of the data.

- The dataset provides a comprehensive snapshot of the Russian IT job market, containing 33,530 job postings.
- The majority of the data comes from HeadHunter (96%) with additional contributions from GetMatch (4%).
- The data quality varies across different fields, revealing patterns of missing values and uneven data distribution.

Data Quality Patterns:

- **Core information:** Well-maintained data for *job titles*, *companies*, and *locations*.
- **Grading systems:** Significant gaps, with 95.5% of data missing.
- **Skills information:** 39% of entries lack detailed skills descriptions.
- **Salary information:** Notable gaps, as 48% of postings miss maximum salary data.
- **Experience requirements:** Reasonable coverage, with only 4.8% of entries missing.

Now let's conduct a general analysis of each feature separately.

Title. The dataset includes a total of 33,538 job postings with 15,569 unique job titles. The average length of a job title is 33.30 characters. The most frequently appearing job titles are System Administrator, Technical Support Specialist, and 1C Developer. These titles highlight the significant demand for roles related to IT infrastructure, technical support, and enterprise systems within the Russian IT job market.

Figure 2: Most popular words in job titles

Title	Count
Системный администратор	1598
Специалист технической поддержки	886
Программист 1С	869
Графический дизайнер	377
Инженер-программист	247
Системный аналитик	204
Специалист технической поддержки (без продаж)	203
Инженер технической поддержки	185
Руководитель проектов	184
Дизайнер	175

Table 1: Top 10 most popular job titles

Location. The majority of job postings are concentrated in Russia's largest cities. Moscow leads with 8,673 vacancies, followed by St. Petersburg with 3,059 vacancies. Other notable IT hubs include cities like Yekaterinburg, Kazan, Novosibirsk, and Krasnodar. However, a substantial portion of vacancies, totaling 16,156, are spread across other locations that do not fall into the top 10 cities. This suggests a diverse geographical distribution of IT job opportunities across Russia, indicating a growing demand for tech professionals even outside major metropolitan areas.

Location	Number of vacancies
Москва	8673
Санкт-Петербург	3059
Екатеринбург	1052
Казань	821
Новосибирск	814
Краснодар	779
Нижний Новгород	675
Ростов-на-Дону	537
Воронеж	492
Челябинск	472
Other	15870

Table 2: Number of vacancies by location

Company. The distribution of vacancies across companies reveals that a majority of postings belong to the "Other" category, which represents a broad range of smaller companies. However, among the largest contributors, the top company is Yandex, underscoring the dominance and influence of big tech companies in Russia. This observation reflects the growing role of major technology corporations in driving the IT job market.

Company	Number of vacancies
Яндекс Крауд	2255
INSOFT	489
Алабуга, ОЭЗ ППТ	352
Ростелеком	317
Первый Бит	221
Т-Банк	206
Ozon	204
Ростелеком Контакт-центр	166
МАГНИТ, Розничная сеть	136
Тензор	127
Other	29054

Table 3: Number of vacancies by company

Skills. The dataset indicates a diverse range of technical skills demanded across job postings. On average, postings require between 5–10 unique skills. The most frequently sought-after technical skills include SQL, Linux, Python, and Git. These skills reflect the importance of data management, operating systems, programming, and version control in the IT industry, aligning with global trends in technical requirements.

Figure 3: Most popular skills

Skill	Number of Vacancies
Грамотная речь	1976
SQL	1973
Пользователь ПК	1682
Linux	1511
Работа в команде	1481
Техническая поддержка	1421
1С программирование	1344
Git	1288
Настройка ПК	1234
Python	1167

Table 4: Top 10 most popular skills

Experience. The distribution of experience requirements shows a clear trend. Both minimum and maximum experience requirements peak at the "3+ years" range, making it the most common baseline for IT roles. This suggests that specialists at the middle+ level are currently in high demand, reflecting the need for experienced professionals who can contribute effectively to technical teams and projects.

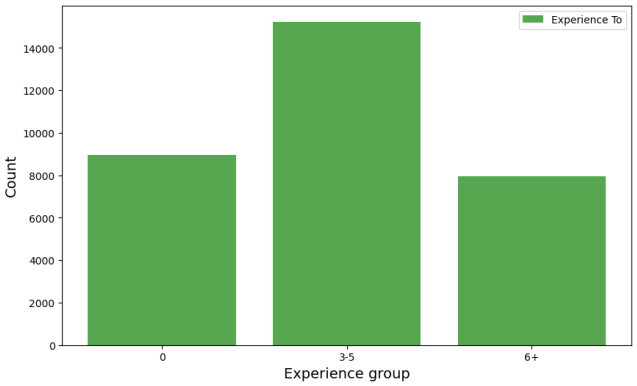

Figure 4: Distribution of maximum experience by groups

Description. The average job description in the dataset contains approximately 1824 characters. Most job descriptions include standard information such as registration processes, work schedules, and insurance benefits. While these details are essential, they highlight the relatively generic nature of many job postings, focusing on organizational and employment conditions rather than specific role-related expectations .

Figure 5: Most popular words in job descriptions

Now let's analyze the salary indicator, which will be our target.

Overall Distribution. The analysis of salary distributions reveals that the highest concentration of salaries falls within the 50-100 thousand range. There are also noticeable outliers in the higher salary ranges. This distribution suggests a mature market with well-established salary bands and structures.

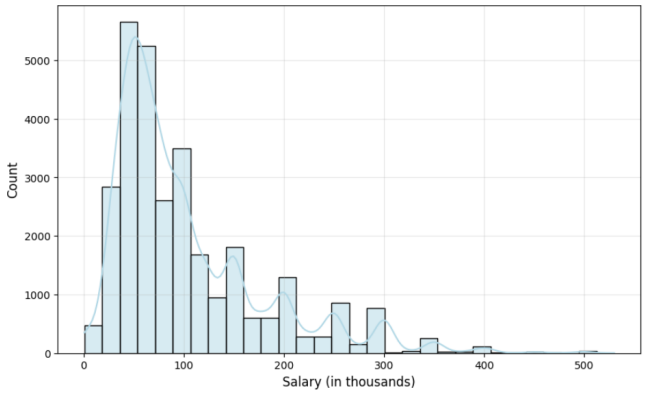

Figure 6: Salary Distribution (`salary_from`, 1st-99th percentile)

Top Job Categories. The analysis covers 10 primary job categories with distinct salary patterns. The top three positions by average salary are as follows:

- **System Analyst:** 161,045
- **1C Programmer:** 154,825
- **Project Manager:** 136,826

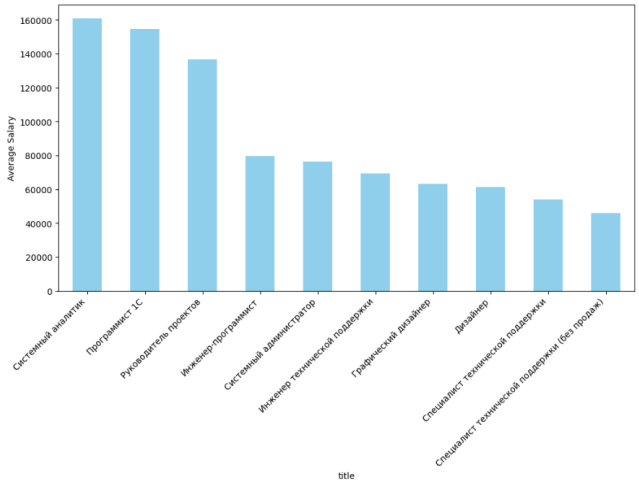

Figure 7: Average salary for Top 10 title categories

At the lower end of the salary spectrum, Technical Support roles (non-sales) average around 46,128. These findings highlight a clear salary hierarchy that reflects the technical complexity and responsibility levels associated with each

role. Additionally, there is a notable salary gap of approximately 115,000 between the highest and lowest-paid categories.

Experience Level Correlation. The analysis also explores the relationship between experience level and salary. The median salary shows a steady increase with higher experience levels. However, for each experience level, there are significant **outliers** with exceptionally high salaries, indicating that some roles or industries reward experience disproportionately.

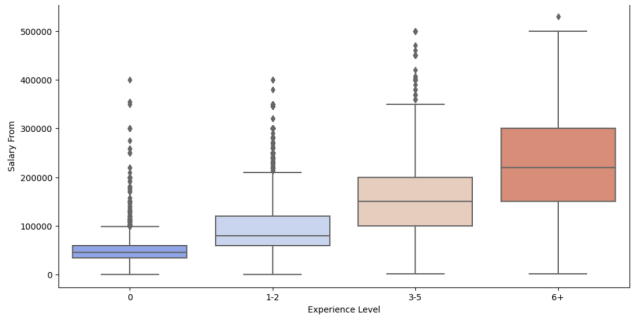

Figure 8: Salary distribution by experience level

After the performed EDA, we prepared the base features to be used by the baseline regression models.

4.3 Basic features and preprocessing

For the target to be predicted, we chose lower salary bound (`salary_from`), scaled into thousands RURs and then log-transformed to allow more normal-like distribution.

Below is the list of the features used by the baseline models.

4.3.1 Transformer and Bi-GRU-CNN baseline

1. Job description (`description_no_numbers`): full text of a job posting with numbers from 10,000 to 100,000,000 replaced by a special [NUMBER] mask to exclude potential data leakage. In our experiments, we also refer to it as Feature 1;

2. Concatenated title, company, location, skills, source (headhunter vs getmatch) in a string (See Table 5). We also refer to it as Feature 2;

For Bi-GRU-CNN, we also used the following text preprocessing pipeline:

1. All text converted to lowercase;

2. Tokenization and lemmatization using Mystem (Russian language morphological analyzer);

3. Removal of special characters and extra whitespace;
4. Vocabulary building with minimum frequency threshold (min_freq=2);
5. Sequence padding to fixed length;
6. Word to index conversion using word2idx mapping;
7. Embedding initialization using pre-trained Russian word vectors.

4.3.2 CatBoost baseline

1. `description_with_skills`: same as "Job description", but skills are concatenated to it;
2. `title`: name of the job position;
3. `location`: location of the job;
4. `company`: company posted;
5. `source`: "headhunter" or "getmatch";
6. experience_from: lower bound of the experience in years;
7. `experience_to`: upper bound of the experience in years, set to 10 when not provided;
8. `job_description_size`: length of the Job description feature.

```
Позиция: {position}
Компания: {company}
Место: {location}
Навыки: {skills}
Источник: {source}
```

Table 5: Scheme of the textual Feature 2.

5 Experiments

In our attempt to achieve the top performance of the regression model, we decomposed the problem into several different domains:

1. **Two textual features**. Since we have two types of the text distinct by their nature, feature 1 (job description) and feature 2 (concatenated title, company, location, skills, source), and their combined length exceeds maximal length of an average transformer model (e.g. 512 tokens for most BERT-like architectures), we decided to process them separately.

For that, we tried three approaches: i) single transformer model trained on two features with downstream concatenating the representations of each text; ii) two transformer heads used, each trained on a separate feature, which the same downstream concatenation; iii) single transformer used, but instead of a simple concatenation of the pooled representations, we run an additional cross-attention layer on the top of transformer outputs to allow the better information transfer between the two textual features. In all these scenarios, [CLS] tokens embeddings or average pooling from each processed feature will be employed in a downstream regression.

2. **Skewed target distribution**. As we showed in the EDA section, there is a subset of the job postings with the bottom salary (`salary_from` range way above its median value. Even though deep learning- and gradient boosting-based approaches are far less sensitive to the outliers compared to the more basic models like linear regression, having more "normal-like" distribution of the target may prove beneficial for the overall prediction quality. To account for the potential negative impact of the outliers on the model performance, we decided to (in addition to predicting log-transformed salary target) use Huber loss, which gives less preference towards the outliers, instead of the mean squared error loss (MSE) when training the regression model.

3. **Heterogeneous nature of the features.** The data available for predicting the salary of the job positions can be essentially split into the three major groups: i) numerical features, e.g. years of overall experience of a candidate; ii) categorical features, such as the presence of a particular keyword; and iii) semantic properties of the text, including its tone, sentiment, etc. The use of a gradient boosting-based methods may be the best in explaining the variance of the target associated with numerical and categorical features but would overlook the semantic patterns in the text. On the contrary, the use of transformer-based approach focused on the natural language processing can achieve the highest results on textual-mostly features but may not capture well the relationships between the numbers and the keywords.

   As an experiment, we decided to assemble the predictions of the two different model types (Catboost as a gradient boosting-based and transformer-based architecture) into one to see whether the percentage of the observed variance we can explain increases in combination compared to the relying on each of the models' predictions separately. Most of the experiments will be performed on a rather small transformer model due to the compute limitations, but to see the effect of blending with the Catboost model predictions, we will test the quality on the bigger transformer model with performance comparable to the one in the Catboost approach.

4. **Specific knowledge domain**. As properties of the texts from the job postings differ strongly from the general language corpus (the use of a technical jargon, job market-related phrases), we tried to increase the ability

of a pre-trained transformer model architecture to capture the patterns in the textual data. On the top of the default transformer finetuning by regression on the [CLS] token embedding, we explored two more options. First, pre-tuning the model with the TSDAE (Transformer-based Sequential Denoising Auto-Encoder for Unsupervised Sentence Embeddings) technique with the subsequent regression. In TSDAE, the model learns to match the [CLS] token embedding (or any other compact sentence representation) of the noised sentence to the one of the original text. Additionally, leveraging the masked language modeling (MLM) ability of a BERT architecture, where we merged the job description with a small prompt containing the [MASK] token to predict the salary. The [MASK] token embedding was then concatenated with the embeddings of the [CLS] tokens.

5.1 Huber Loss

Huber loss is a combination of the squared loss and absolute loss functions:

$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \leq \delta, \\ \delta(|a| - \frac{1}{2}\delta) & \text{for } |a| > \delta, \end{cases} \tag{1}$$

Within a certain boundary defined by δ, the loss behaves as a squared loss. However, outside the boundary, it effectively acts as an absolute loss, becoming less sensitive to the outliers [Huber, 1964].

5.2 Cross-attention for better "communication" between two textual features representations

In the default setup, the information flow between textual feature 1 (job description) and 2 (concatenated features of the posting, such as title, company, source, etc.) is performed via concatenating the compact representation of each feature (e.g. [CLS] token embedding) with the subsequent regression on the combined embedding. To explore more ways of communication between the features, we employed the cross-attention approach, as described in [Vaswani et al., 2017].

For each sample, we used embeddings of the feature 1 as queries Q, and embeddings of the feature 2 as keys K and values V. The attention weights are computed as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2}$$

where d_k is the dimension of the key vectors. For multi-head attention with $h = 8$ heads:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \tag{3}$$

where W^O is the output projection matrix that projects the concatenated attention heads back to the original embedding dimension.

Each head is computed as:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{4}$$

Where each respective projection matrix is used for queries, keys, and vectors.

After the attention block, [CLS] token embedding (or average pool representation), after residual connection and layer normalization, was concatenated with [CLS] token embedding (or average pool representation) from the second feature for the downstream regression. We used a padding mask for the keys to prevent padding tokens embeddings to participate in the block.

5.3 TSDAE (Transformer-based Sequential Denoising Auto-Encoder for Unsupervised Sentence Embeddings)

First suggested in 2021 by Wang et al [Wang et al., 2021], TSDAE is an unsupervised training approach where in encoder-decoder transformer setup, the model learns to reconstruct the representation (e.g., [CLS] token embedding) of a noised sentence (with tokens swapped or deleted) to make it as close as possible to the embedding of the original sentence. As the decoder has only access to the pooled information of the sentence ([CLS] token embedding in this case), this additionally forces the encoder to provide the compact representation of the text.

TSDAE may be used as either a pre-training task (instead of masked language modeling) or as a fine-tuning approach to adapt the model to a particular domain. We hypothesized that this concept may be a part of the fine-tuning pipeline where the model is first trained to reconstruct the noised embeddings of the small portion of the training data, and then further tuned in a "canonical" way by performing regression over the [CLS] token embeddings using the rest of the training samples.

As job descriptions in our dataset are rather large, with the majority of the samples being close to 512 tokens in length (maximal sequence length for the most of BERT-based models), we reformulated the TSDAE pipeline the following way. Instead of learning to reconstruct the [CLS] token embedding of the long piece of a noised text (whose semantic properties may be less pronounced than for a shorter fragment), we picked a subset of the sentences 10-60 words long (around 1000 examples) from 1% of the training data and run the model through TSDAE pipeline on the short sentences, using the parameters that achieved the best performance in the original publication (deletion as an only noise source; deletion ratio of 0.6; [CLS] token embedding as a sentence representation). Another BERT-based model was pre-tuned on the full-length noised feature 2 texts (combined title, company, location, etc.) using the same 1% training data. The resulting two models were then trained as usual on the regression task, each on the respective feature.

5.4 [MASK] token embedding as an additional source of the text representation

To further explore capabilities of the transformer architecture to be used in the regression task over the textual features, we decided to leverage masked language modeling ability of the BERT-based models in the salary prediction task. Here, as shown in the Table 6, we concatenate the job description text with the short prompt that contains a [MASK] token, whose embedding we are then going to concatenate with the [CLS] token representation for their downstream use in the regression.

[CLS] Далее указано описание вакансии. Судя по описанию, зарплата на этой позиции составляет [MASK]. [SEP] <JOB DESCRIPTION>

Table 6: Scheme of the modified job description.

5.5 Metrics

We are going to use two metrics as references in our experiments, R^2 score and mean absolute error.

5.5.1 R^2 score

A primary metric used in the study, R^2, also known as the coefficient of determination or R^2 score, represents the proportion of the variance in the target variable which can be explained by the given model. It is denoted as:

$$R^2 = 1 - \frac{\mathrm{SS}_{\mathrm{res}}}{\mathrm{SS}_{\mathrm{tot}}} \tag{5}$$

where

$$\mathrm{SS}_{\mathrm{res}} = \sum_{i=1}^{n} (y_i - f_i)^2 \tag{6}$$

and

$$\mathrm{SS}_{\mathrm{tot}} = \sum_{i=1}^{n} (y_i - \bar{y})^2 \tag{7}$$

Here, y_i is the target value for the i-th sample, f_i is the value given by the model for the i-th sample, and $\bar{y}$ is the mean value of the target. A baseline regressor which predicts by giving a constant output of the average value has $R^2 = 0$, and in the ideal case R^2 equals 1 (all the variance can be explained by the model). As the predictions can be arbitrarily wrong, the R^2 has no bottom boundary, hence its possible values are $(-\infty, 1]$.

5.5.2 Mean Absolute Error (MAE)

Another indicator of the regressor quality used here is mean absolute error, or MAE, which is an average of the absolute values of residuals, defined as:

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - f_i| \tag{8}$$

where y_i and f_i are the target value and the value given by the model for the i-th sample, respectively.

Less sensitive to outliers compared to a family of squared error-based metrics (e.g., mean squared error (MSE) and its rooted variation RMSE), MAE, in our opinion, is a proper reference for model performance. Its possible values are $[0, +\infty)$, and the only shortcoming is that it is less interpretable compared to the R^2 score, especially when predicting a log-transformed target.

5.6 Experiment Setup

With a fixed random seed, the dataset was divided into the training (80%) and validation (20%) splits. The models were trained on the regression task with the same random seed. This pipeline was repeated for three random seeds, the metrics were averaged, and 95% confidence intervals were calculated.
We used the following parameters for the training.

5.6.1 sergeyzh/rubert-tiny-turbo model

- Batch Size: 32
- Maximal Sequence Length: 1024
- Optimizer: AdamW, all parameters but learning rate (5×10^{-6}) set to default
- Pooling: via [CLS] token embedding
- MLP regression head: one linear layer to reduce embedding dimension to 128, then ReLU as a non-linearity, and final linear layer to reduce dimension to 1.
- Number of Epochs: 10

5.6.2 intfloat/multilingual-e5-small model

- Feature preprocessing: each feature was concatenated with a short string "query: " before training, as recommended by the model authors
- Batch Size: 32
- Maximal Sequence Length: 512

- Optimizer: AdamW, all parameters but learning rate (2×10^{-5}) set to default
- Pooling: Average pooling
- MLP regression head: one linear layer to reduce embedding dimension to 128, then ReLU as a non-linearity, and final linear layer to reduce dimension to 1.
- Number of Epochs: 10

5.6.3 Huber loss

For the experiments with **Huber loss**, we used default δ value of 1.

5.6.4 Multi-head attention block

For the experiments with adding **multi-head attention block**, we used the following parameters:

- Size: 8 attention heads (`num_heads`)
- The output dimension: the hidden size of the transformer model
- Dropout Rate: 0
- Pooling: [CLS] pooling for *rubert* and average pooling for *e5-small* model
- Residual connection with the downstream layer norm
- Maximal length for the feature 1 (job descriptions): 1024 or 512 tokens depending on the model type
- Maximal length for the feature 2 (concatenated job features): 256 tokens
- `key_padding_mask` was supplied to mask the padding tokens embeddings in the keys array.

5.6.5 TSDAE pre-tuning

For **TSDAE pre-tuning** of the feature 1-focused model, 1% of the training dataset was additionally used for the task (extract all the unique 10-60 word sentences from job descriptions and train the model on them). For feature 2-related tuning, we used all the full-length sentences from the same 1% of the training data.
The parameters:

- Noise Source: Deletion only
- Deletion Ratio: 0.6
- Pooling: [CLS] token embedding

- Tied Encoder-Decoder: True
- Batch Size: 2
- Learning Rate: 3×10^{-5}
- Weight Decay: 0
- Number of Epochs: 1

5.7 Baselines

To compare the results of the experiments, we chose three strong baseline solutions which are most often used for similar tasks, based on the reviewed literature in the related fields:

- gradient boosting with decision trees;
- combined RNN/CNN setup;
- transformer architecture.

Additionally, we also introduced a rather technical weak baseline where the prediction is always an average target value. Such a baseline will be useful when analyzing the performance in terms of MAE, whose values have no reference points.

5.7.1 Gradient boosting-based approach

It could be that most of the useful information which can be used for salary prediction is present in the numerical or categorical form. Such hypothesis found its reflection in the work of [Sergeeva, 2024], where the use of gradient boosting-based approach saw the closest score to the state-of-the-art solution in the salary prediction task (R^2 score of 0.62 vs 0.64). Based on that, we leveraged the power of CatBoost, decision tree-based framework suitable for working with datasets heavy on categorical features.

We tried to make this baseline as strong as possible, by incorporating extra features (such as number of years of experience, job posting source), adding custom text preprocessing pipelines (tokenization, dictionary type and size), and hyperparameter tuning. The full development of the CatBoost model can be found in the repository for this study.

5.7.2 Bidirectional GRU-CNN approach

The importance of textual information in job postings for salary prediction motivated the exploration of a hybrid neural network architecture combining bidirectional GRU (Gated Recurrent Unit) with CNN (Convolutional Neural Network). This approach, implemented as one of our baselines, focuses on capturing both sequential patterns and local features in job descriptions through its dual-architecture design.

The model processes two main text features: the job description (with numbers removed for better generalization) and a structured template combining job title, company name, location, required skills, and vacancy source. The architecture employs a 256-dimensional embedding layer initialized with pre-trained Russian word vectors, followed by a bidirectional GRU layer with 128 hidden units. The GRU output is then processed by a convolutional layer with 64 filters and kernel size 3, followed by max pooling and dropout layers for regularization.

Training was conducted using the Adam optimizer with a learning rate of 10^{-3} and L1 loss function, achieving moderate performance levels. To ensure robust evaluation, the model was initialized, trained and evaluated on 80/20 splits with multiple random seeds, and results were reported with 95% confidence intervals. The implementation details can also be found in the GitHub repository for this project.

5.7.3 Transformer-based approach

Despite its relatively weak ability to handle numerical/categorical features, transformer architecture by itself may be useful in capturing the semantic aspects of the text. Here, we used the BERT-based model *sergeyzh/rubert-tiny-turbo* which is a decent baseline due to its performance on ruMTEB benchmark and small size of 29M parameters, making it an attractive platform for the further experiments with a scarce GPU resources. Single BERT model was trained on both textual features using MSE loss.

6 Results

The results are presented in Table 7. When testing for the statistical significance in the results, confidence intervals comparison was used.

Experiment	R^2 score	MAE
Baselines		
By average	0.000 ± 0.000	0.513 ± 0.002
Bi-GRU-CNN	0.652 ± 0.012	0.288 ± 0.007
CatBoost	0.734 ± 0.005	0.248 ± 0.004
rubert-tiny-turbo (29M)	0.645 ± 0.027	0.289 ± 0.012
Modifications		
Double rubert-tiny-turbo	0.643 ± 0.024	0.291 ± 0.013
+ Huber loss + TSDAE	0.657 ± 0.056	0.285 ± 0.024
rubert-tiny-turbo + Huber loss	0.655 ± 0.035	0.286 ± 0.016
+ extra [MASK] pooling	0.599 ± 0.034	0.313 ± 0.015
+ cross-attention	0.671 ± 0.027	0.279 ± 0.014
multilingual-e5-small (118M) + Huber loss	0.723 ± 0.024	0.254 ± 0.013
+ cross-attention	0.729 ± 0.017	0.251 ± 0.009
+ CatBoost	**0.770 ± 0.001**	**0.229 ± 0.003**
+ cross-attention + CatBoost	0.769 ± 0.014	0.229 ± 0.01

Table 7: Performance of the models tested in the study. Metrics are reported as a mean value ± 95% confidence intervals across three random seeds. Overall state-of-the-art results are placed in **bold**, while the best results for a solo transformer model are underlined.

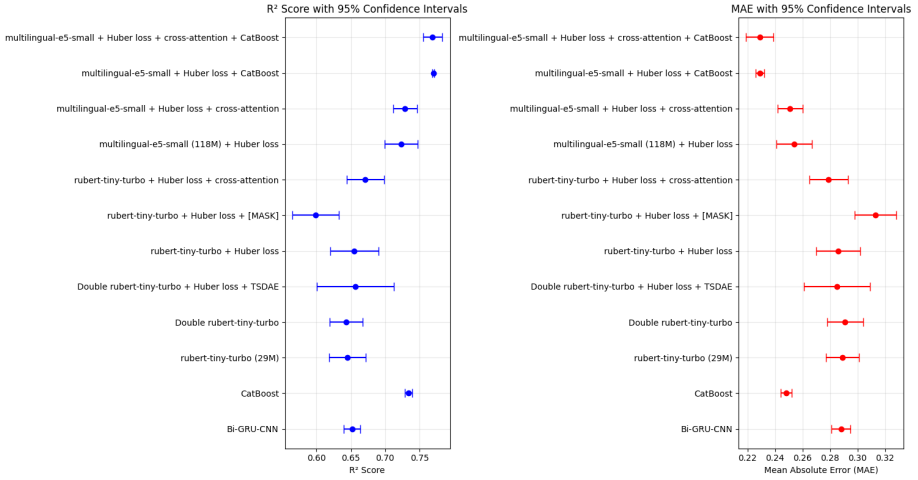

Figure 9: Comparison of model performances with confidence intervals. Left: R^2 Score showing explained variance of predictions. Right: Mean Absolute Error (MAE) showing average prediction deviation.

First, we cover the experiments whose performance saw improvement over the baseline approaches.

6.1 Impact of Huber loss

Introduction of Huber loss instead of MSE led to an increase in the mean R^2 score to 0.655 (+0.01 over baseline), aligning with the dataset's target outliers and indicating that a loss function less sensitive to large errors can be beneficial. However, this improvement was not statistically significant at the 95% confidence level, likely due to high variance in the transformer's predictions.

Compared to the CatBoost baseline, the performance of this transformer setup was still weaker. It could be that either i) the ability of the small BERT model to capture the semantic (and perhaps categorical) patterns does not exceed the CatBoost power to handle numerical/categorical data or ii) the data itself has higher influence of numerical components, making it much harder even for a stronger transformer model to beat the baseline.

A similar point can also be made when comparing the transformer setup with Bi-GRU-CNN (strong at capturing local more keyword-like patterns), where the introduction of Huber loss caused equal performance with the RNN/CNN implementation.

Interestingly, we did not see a large improvement in MAE over transformer baseline, potentially indicating that the use of Huber loss mostly enhanced the explained variance for the objects with the target value close to its median value. This setup was further tested with the larger model (118M parameters), *multilingual-e5-small*, where it achieved 0.723 in R^2 score and MAE of 0.254, which was significantly better than the Bi-GRU-CNN baseline.

6.2 Impact of the added cross-attention layer

Introduction of the cross-attention layer on the top of the BERT head with Huber loss provided a large increase in the quality of the model, reaching R^2 score of 0.671 (+0.016 from the simple feature representations concatenation setup and +0.026 from the baseline) and improving MAE from 0.286 to 0.279 over the setup without the cross-attention layer. However, we could not find the statistical significance of this improvement.

A more advanced way to transfer the information between two distinct features, mediated by the attention mechanism, may allow more accurate predictions to be made. This setup, scored the highest among all *rubert-tiny* setups, was further tested with the larger 118M model, *multilingual-e5-small*, where it achieved R^2 score of 0.729, closest to the CatBoost performance and outperforming Bi-GRU-CNN architecture.

Interestingly, the effect of adding a cross-attention layer was about 2-3 times less for the larger model compared to *rubert-tiny-turbo* (+0.006 versus +0.016). It is tempting to speculate that such a difference lies in the different ability of the model to produce compact sentence representations. The representations provided by a bigger model are to the larger extent enriched by the context

and simple concatenation of two features representations would work well by itself. Yet, for a model with fewer parameters, sentence representations may not capture the context to the same degree, which makes application of cross-attention more beneficial.

6.3 Blending CatBoost and transformer predictions

Combination of the best transformer setup (Huber loss, added cross-attention layer, larger model) with the baseline CatBoost approach by averaging their predictions, as expected, led to the improved metrics compared to each of the methods separately. Mean R^2 score increased to 0.769 (+ 0.035 from CatBoost-only and + 0.040 from transformer-only setups), while mean MAE dropped to 0.229 (- 0.019 and - 0.025 respectively). The difference in the performance in both metrics was statistically significant when compared to the both best transformer and CatBoost setup. Interestingly, the prediction by the combined model had much less noise as in the transformer-only approach, likely due to the inherently high prediction confidence of the CatBoost model.

Additionally, by comparing the effect of the presence of an extra cross-attention layer on the performance of the combined model, we found a negligible difference in the quality of the models (R^2 of 0.769 with and 0.770 without cross-attention layer), but with much more confident predictions of the model without the cross-attention layer. So, this custom setup (combined predictions of the transformer model trained on Huber loss + the CatBoost model), designed and implemented in the current work, achieved **state-of-the-art results** on our dataset.

Since the cross-attention layer had no effect on the combined model average performance (but had an effect for the solo transformer quality), we conducted the same experiments using lightweight *rubert-tiny-turbo* and observed a similar pattern (presence of the layer gives + 0.004 for the combined model vs + 0.016 for the solo transformer, full results not shown). As covered before, the target variability may have two distinct components: numeric/categorical variability (e.g, years of experience and skills) and semantic variability (e.g., vivid vs official tone of the text); in line, our initial proposal was that while CatBoost can strongly target the first variability component, transformer-based approach is better at explaining semantic-related variability.

Hence, we hypothesize that the cross-attention layer has very little (if any) effect when the transformer model is combined with the CatBoost one because such improved information flow between the representations on the two features helps explain categorical variability of the data. The cross-attention setup enriches job description representations with categorical context of the Feature 2, improving its pattern extraction. The CatBoost model likely already captures these relationships regardless of the extra information from the transformer, so the introduction of the cross-attention layer cannot explain more of the target variability.

6.4 Negative Results

Below we cover the experiments whose results did not provide any improvements in the metrics.

6.4.1 Separate transformer model per each textual feature

We have not observed any increase in the quality of the predictions when two transformer models were used per two features. This indicates that the single model has enough ability to generalize on the different types of the texts.

6.4.2 Impact of TSDAE pre-tuning

TSDAE pre-tuning did not cause any improvements in the metrics, but led to the higher noise in the predictions. As the approach was originally developed for rather short sequences, it may be that in the case of fine-tuning a transformer model on long texts, the use of TSDAE requires a different strategy for dataset preparation, such as the source of the introduced noise and its frequency. Due to the higher variance in the metrics, we decided not to proceed further with this approach.

6.4.3 Impact of an extra [MASK] pooling

Combined [CLS] and [MASK] tokens embeddings pooling for the job description feature displayed reduction in the quality of the model (as seen by a drop in R^2 score from 0.655 to 0.599). It can be that despite their strength in the masked language modeling task, BERT-based models in their vanilla setup are not very suitable for such a regression approach via [MASK] tokens. A failure of the model to express the numerical value via the mask token embedding may come from the nature of the masked language modeling (MLM) task, where the model learns to predict a missing token via a classification task and not regression. This goes in line with the report showing that the transformer architecture performs better at approximating piecewise constant functions than the smooth ones [Nath et al., 2024]. A similar concept found a reflection in the studies by Thawani et al. [Thawani et al., 2023], where the numerical reasoning of the transformer models was improved by introducing special number-related tokens and performing MLM on this specific subset.

7 Conclusion

Here, we developed an approach to predict a salary based on the provided job description which achieved the state-of-the-art performance compared to the previously published setups. The approach consists of a *e5-small* shared transformer encoder trained on Huber loss, whose predictions are combined with those of CatBoost model.

We started with the data collection, fetching job postings with salaries within the IT sector from two sources, hh.ru and getmatch.ru. After the data cleaning and feature engineering, we evaluated three baseline approaches: i) a combination of CNN and RNN architectures (Bi-GRU-CNN); ii) CatBoost regressor; and iii) BERT-based model with a downstream regressor MLP head.

In our attempt to improve the quality of the predictions, we addressed several problems characteristic to the task: i) presence of two distinct textual features; ii) skewed distribution of the target; iii) domain-specific vocabulary of the descriptions; and iv) combination of both numeric/categorical and textual features.

For the first problem, we attempted to enhance the information transfer between these features by introducing the cross-attention layer on the top of the BERT head, which demonstrated strong increase in the performance. Using two separate encoders for the two textual features did not improve performance compared to a single shared encoder, suggesting that the model's capacity to generalize across text types was sufficient with shared parameters.

As for the second set of problems, we reduced the effect of the outliers on the model by using Huber loss instead of MSE as a loss function in the regression. This modification indeed increased the quality of the prediction as measured by R^2 score, yet MAE did not see the same improvement.

We also tried to tackle the domain specificity of the textual data by either pre-tuning the transformer encoder via i) training it to reconstruct the noised sentences of the job description corpus (TSDAE) or by ii) performing an extra pooling via [MASK] token embedding as an estimate to the salary number to predict. Unfortunately, none of these approaches displayed the increased quality of the prediction, with the latter [MASK] token-related pipeline even rendered the model to perform worse.

The biggest improvement was observed when we combined the predictions of the CatBoost and the transformer models. Due to their natural ability to target different types of patterns (numbers and categories for the former, semantic quality for the latter), blending of the two helps to reduce the unexplained variability of the target compared to their separate usage. This custom setup developed in the current study, with the transformer model quality enhanced by Huber loss and an extra cross-attention layer, achieved state-of-the-art results on our dataset, scoring higher than the explored baseline approaches.

Future work can explore several directions to build upon this study. First, improving domain-specific pre-tuning strategies for transformer models could enhance their ability to handle highly specialized vocabularies. Second, refining numerical representation within transformers, such as incorporating number embeddings or custom pre-training for numeric data, could mitigate limitations observed with the [MASK]-based pooling. Lastly, applying this hybrid approach to broader datasets or additional industries may reveal its generalizability and limitations. Integrating other ensemble techniques, such as stacking or weighted averaging, may further improve prediction quality.

References

[Huber, 1964] Huber, P. J. (1964). Robust Estimation of a Location Parameter. *The Annals of Mathematical Statistics*, 35(1):73 – 101.

[Krishujeniya, 2023] Krishujeniya, K. (2023). Predicting salaries of data professions using machine learning. *https://krishujeniya.medium.com/predicting-salaries-of-data-professionals-using-machine-learning-a77856f8a7b9*.

[Le and Dupont, 2022] Le, T. and Dupont, M. (2022). Regression with text input using bert and transformers. *https://lajavaness.medium.com/regression-with-text-input-using-bert-and-transformers-71c155034b13*.

[Li et al., 2020] Li, S., Jin, X., Xuan, Y., Zhou, X., Chen, W., Wang, Y.-X., and Yan, X. (2020). Deep transformer models for time series forecasting: The influenza prevalence case. *https://arxiv.org/pdf/2001.08317*.

[Ma, 2020] Ma, B. (2020). Job salary prediction for analytics professionals.

[Mumminen, 2023] Mumminen, A. (2023). A data-driven approach to salary prediction. *https://medium.com/@mummineniamar/a-data-driven-approach-to-salary-prediction-158f6cb8f121*.

[Nath et al., 2024] Nath, S., Khadilkar, H., and Bhattacharyya, P. (2024). Transformers are expressive, but are they expressive enough for regression?

[Sergeeva, 2024] Sergeeva, E. (2024). Salary prediction based on job description in the it sector. Master's thesis, HSE University, Moscow, Russia.

[Shilo, 2022] Shilo, P. (2022). Job salary prediction with transformers. *https://www.kaggle.com/code/pelegshilo/job-salary-prediction-with-transformers*.

[Thawani et al., 2023] Thawani, A., Pujara, J., and Kalyan, A. (2023). Estimating numbers without regression.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need.

[Wang et al., 2021] Wang, K., Reimers, N., and Gurevych, I. (2021). Tsdae: Using transformer-based sequential denoising auto-encoder for unsupervised sentence embedding learning.

[Wang et al., 2022] Wang, L., Yang, N., Huang, X., Jiao, B., Yang, L., Jiang, D., Majumder, R., and Wei, F. (2022). Text embeddings by weakly-supervised contrastive pre-training.

[Wang et al., 2024] Wang, L., Yang, N., Huang, X., Yang, L., Majumder, R., and Wei, F. (2024). Multilingual e5 text embeddings: A technical report. *arXiv preprint arXiv:2402.05672*.

[Zhang et al., 2024] Zhang, W., Chen, L., and Wang, Y. (2024). Transformers are expressive, but are they expressive enough for regression? *https://arxiv.org/pdf/2402.15478*.