UDC 004.853

Transforming raw corporate texts into instruction dataset for fine-tuning generator within a RAG system

Eliseev V.O. (Institute of Applied Mathematics and Mechanics)

Maksimova A.Y. (Institute of Applied Mathematics and Mechanics)

Bondarenko V.I. (Donetsk State University)

We address the problem of incorporating domain-specific knowledge into large language models (LLMs) to be used in intelligent assistants systems. We propose an algorithm for building a specifically formatted instruction dataset designed to fine-tune an LLM as a generator component within a Retrieval-Augmented Generation (RAG) system. The practical aspect of our work involved constructing an instruction dataset from raw corporate texts related to Donetsk State University (DonSU). The resulting dataset contains over 25,000 input–context–output records and will be used to fine-tune an LLM for its role as the generator in DonSU's RAG-based intelligent assistant.

Keywords: Retrieval Augmented Generation, RAG, LLM, Fine-Tuning, Instruction Dataset, AI-assistant

1. Introduction

Providing students and applicants with up-to-date information is crucial for any university. With the availability of powerful large language models (LLMs) in both open-source (e.g, LLaMA) and commercial (e.g., GPT-4) formats, it has become possible to create intelligent assistants capable of performing question-answering (QA) over corporate data and automating this communication process. This approach is not constrained by the university domain and can be used wherever QA search over a knowledge base is applicable.

Although modern LLMs demonstrate impressive capabilities, they typically lack knowledge about internal processes within organizations unless such information is publicly available and included in their training data. As a result, integrating this "hidden" knowledge into LLMs remains a significant challenge for developers of intelligent assistants.

The most common approaches to incorporating domain-specific knowledge into LLMs are building Retrieval-Augmented Generation (RAG) systems [11] and fine-tuning LLMs [3]. They enabling chosen LLM to work with unseen during pretraining domain-specific data with reason-

able effectiveness. However, both approaches suffering from their key limitations, making using them separately less efficient. These limitations will be described in more detail in Section 2.

In this paper, we propose an approach that combines supervised fine-tuning with RAG. We also introduce a method for constructing a dataset that enables an LLM, after fine-tuning on it, to effectively serve as a generator within a RAG system. Traditional instruction datasets are typically constructed as collections of input-output pairs, where the input represents a user query or instruction, and the output corresponds to the expected model response. In this work, a different approach is introduced: datasets are formulated as input-context-output triples. In this configuration, the context provides external, domain-specific information, which must be interpreted by the generator model to produce a relevant and accurate response.

The practical part of our research are based on data from official Donetsk State University (DonSU) website. Our experiments involve these steps:

1. Collecting raw texts from the website;
2. Selecting relevant texts to proceed;
3. Generating synthetic classical instructional dataset using an LLM;
4. Building a vector database over the relevant texts;
5. Adding the *context* field according to the input from the built vector database to generated instructional pairs.

The source code for each step and experiment has been made publicly available; corresponding links are provided in the relevant sections.

2. Related Work

The most common approaches for incorporating domain-specific data into LLMs are Retrieval Augmented Generation (RAG), which serves as the simplest baseline to implement and supervised fine-tuning, which is typically a more complex and resource-intensive process.

The most straightforward approach for adapting LLMs to domain-specific data is fine-tuning. Previously, assistant models were fine-tuned using dialogues that were either manually annotated or extracted from external sources, such as Twitter conversations and Reddit comments. With the emergence of instruction-based training for LLMs, OpenAI researchers fine-tuned GPT-3 on manually curated instruction data, leading to the development of InstructGPT [15], the precursor to ChatGPT. However, creating manually annotated instruction datasets for training assistant models outside industrial settings remains challenging and resource-intensive.

As a result, synthetic datasets generated by powerful LLMs or constructed using the Self-Instruct approach [18] are now commonly employed for this purpose.

There are several fine-tuning techniques exists, such as fine-tuning the whole model weights, which is costly and infeasible in non-industrial settings; adapting specific model layers while freezing the rest [12]; training the bias vectors while freezing everything else [20]; adapter tuning [4]; and others. The best way to fine-tune LLM, given the available resources for implementing an assistant, is Low-Rank Adaptation (LoRA). LoRA gives qualitative results by training a relatively small number of parameters while also providing flexibility in modifying the model's behavior during runtime [5].

By tuning an LLM on a domain-specific dataset containing classical instructional input-output pairs, we make the model simply memorize the output facts and their relations to inputs. This is not an ideal solution for building a QA system for a university, because the information about almost any university frequently changes. As a result, we would either need to continuously retrain the model on up-to-date data, which is costly and resource-intensive, or rely on lately added external sources, which the model, after instruction tuning, would struggle to utilize effectively.

A RAG also is used to enrich a large language model with domain-specific knowledge. The first mention of them was introduced by the FAIR team in 2020 [11]. Authors proposed using two models for QA search: a retriever (specifically, Dense Passage Retriever) and a generator — a transformer model. As a generator authors suggested BART-large [10], a pre-trained seq2seq transformer [17] with 400M parameters.

The retriever is responsible for vectorizing domain-specific texts, which are typically split into smaller chunks. Vectorization enables the construction of an n-dimensional representation that captures the semantic meaning of the text. The concept of vector representations was first explored by Mikolov et al. [14] during the development of the word2vec model. During RAG inference, the retriever also encodes the user query in such a way that document chunks most likely to contain the answer have the highest vector similarity to the query. The user query, together with the most relevant context chunks, is then passed to the generator, which produces a human-readable text serving as the final response.

This "Naive RAG" approach has been improved by adding metadata to document chunks, using it for document ranking (e.g., by including the document's publication date, which helps assess its current relevance), and pre- and post-retrieval processing.

A major limitation of the RAG framework lies in its reliance on the retriever, which may yield imperfect results [1]. Consequently, the generator may fail to produce a relevant response from the retrieved chunks. This typically occurs when none of the chunks contain the correct answer or when they include information about concepts only weakly related to the query's semantics, leading the model to generate incorrect outputs. The most effective strategy to address such cases is to train the model to recognize when it lacks the necessary information and to produce a predefined fallback response.

3. Proposed Method

To address the aforementioned limitations, we propose a combination of fine-tuning and RAG. Specifically, we aim to train an LLM not only to answer questions using domain-specific knowledge integrated into its weights during fine-tuning, but also to learn to extract the most relevant information from the document chunks provided by the retriever.

We propose extending the classical instructional dataset, traditionally composed of input-output pairs, by introducing a third component — *context*. The context is retrieved from a vector database constructed from the texts used to generate the original instructions, by selecting the top-n chunks most relevant to the input. Providing data as input-context-output triplets within a formatted prompt — matching the format used during assistant inference — in the fine-tuning process is expected to enhance the model's extraction capabilities, which are crucial for a generator in RAG system. Using this approach, we preserve one of the key advantages of fine-tuning — the ability to control the model's alignment and produce more domain-specific answers — while retaining a major benefit of RAG: the flexibility to update the knowledge base without the need to retrain the LLM.

This idea has been analyzed by Lin et al. [13], but this work was focused on training models to serve as generators in RAG systems working with multi-domain data in English. In our work, we focus on constructing a dataset from raw corporate texts in Russian related to a single domain of specific university — DonSU.

The official website of Don State University (DonSU) was used as the source of raw texts, as it provides structured and machine-readable content. The sequence of performed steps is shown in Figure 1.

Next, in this paper, we focus on the process of constructing a dataset in the format described above. By relevant texts, we refer to the textual content of the parsed HTML pages from the

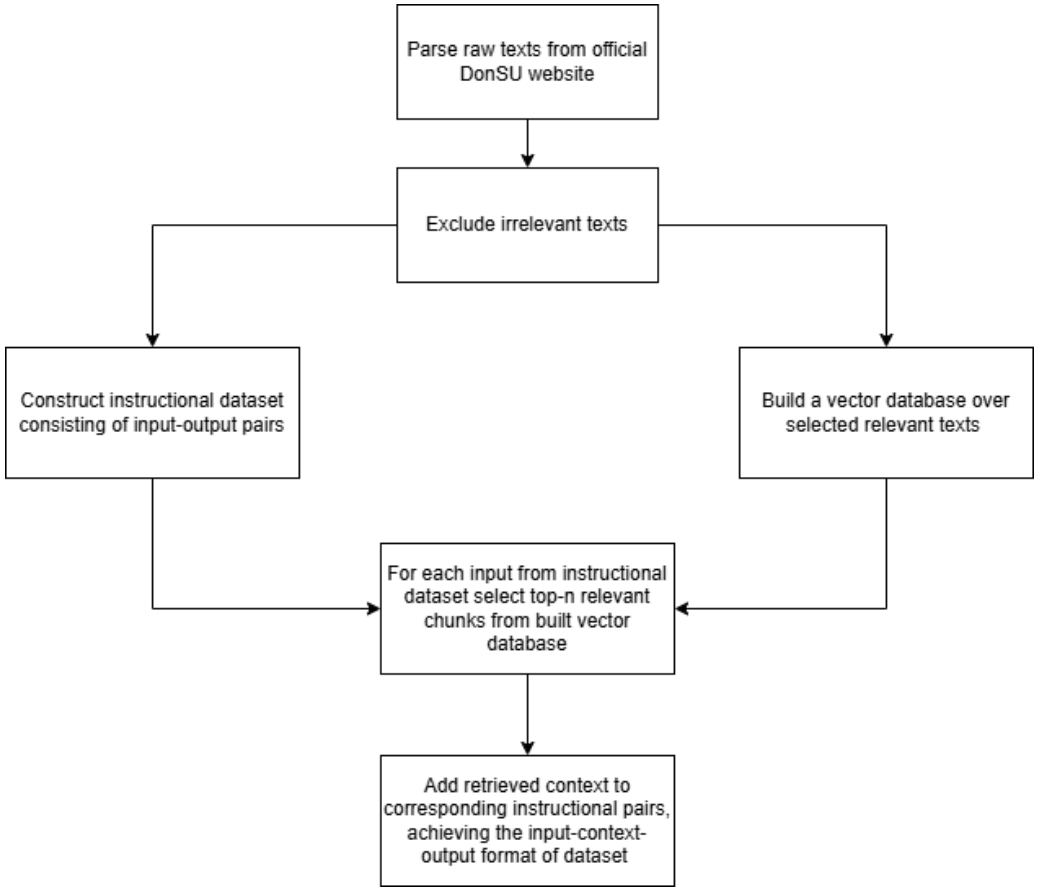

Figure 1. Performed steps during the practical part of the research.

university's website. Additionally, we downloaded all documents in textual formats (such as .pdf, .docx, .pptx, etc.), but they will only be used to populate the inference vector database.

All the steps performed, along with links to the corresponding source code, are described in detail in the following sections.

We leave the actual fine-tuning of the generator, its evaluation during RAG system inference, and the comparison of tuned models with and without the context field for future work.

4. Collecting Of Raw Texts

As the source of domain-specific data about DonSU, we chose its official website, as it provides both structured and machine-readable content.

We created a set of scripts which is open-source and available at `https://github.com/EliseevVadim/WebsiteCrawler`. The provided tools enable both parsing text data from a specified website and resulting text files processing features, including removing empty ones, merging them into a single document which is suitable for fine-tuning models via next-token prediction [2]. One of the script's arguments is a content selector, that identifies the specific

block on each HTML page on the website containing the main textual content. Putting the text content into the block with same CSS-selector is mandatory and makes the university websites machine-readable. The user can also define the depth of website traversal; if not specified, the script either crawls all available pages or stops once the defined depth is reached. Additionally, it is possible to configure whether textual files in various formats should be downloaded along with the saved HTML content. In our case, such files were downloaded, but only the texts extracted from parsed HTML pages were used in the subsequent research stages. The downloaded files were reserved for later use in populating the inference vector database.

Upon execution, the script initiates data collection from the root page of the website and recursively traverses all accessible internal hyperlinks. When encountering a URL that references an HTML page, the script extracts the main textual content and stores it in a separate file. For URLs referring to downloadable documents, the corresponding files are retrieved if this functionality has been explicitly enabled by the user. During the HTML parsing process, additional metadata — such as page titles and last modified timestamps are also collected. These metadata are expected to be beneficial for downstream tasks, including instruction generation and relevance-based ranking in the retrieval component of the vector database.

After website crawling conducted on December 11, 2024, a total of **2,912** text files were generated from the extracted content of parsed HTML pages. In addition, **3,546** documents in various formats were downloaded, comprising a total data volume of **14.3** GB. This volume of data is considered sufficient for constructing a domain-specific knowledge base to support the development of the intelligent assistant. These 2,912 files originated from HTML pages will be used in our research and will populate both RAG-powered instruction dataset and inference vector database.

5. Selecting The Relevant Texts

Analysis of the text files retrieved in the previous step revealed that some of them lacked relevant information suitable for either generating instructions or constructing the vector database for subsequent research and RAG system inference. Such files typically originated from intermediary pages containing only hyperlinks to other pages of the website or direct links to downloadable documents. Consequently, these files must be excluded from further processing.

To filter such files we decided using LLM-as-a-judge. A prompt that incorporates the content of a specific text file generated in the previous step along with its title, corresponding to the

title of the originating web page was designed. The prompt instructs an external LLM to assess the usefulness of the provided text for constructing an instruction dataset intended for LLM fine-tuning. Furthermore, a rating scale was included in the prompt to categorize each file based on its relevance.

The original prompt that was sent to LLMs was made in Russian and it is available at `https://github.com/EliseevVadim/TextsEstimator/blob/main/prompts/texts_evaluation.txt`. The English translation of the prompt is presented in Figure 2.

```
Read the text below, obtained from parsing a web page of an educational organization, and evaluate
its educational value and usefulness for training an LLM in building an instruction dataset (i.e.,
assess how well the provided text can be used to generate question-answer pairs) that will be used in
an intelligent assistant.
Consider the following criteria in your evaluation:
Content of the text.  Pay special attention to links and mentions of resources from 2022 and later,
as they are the most relevant.  Some texts are merely textual representations of links, and sometimes
entire pages contain only links.  Such files cannot be used to build a relevant instruction dataset,
so these cases should receive low scores.
File name.  The file name may indicate the topic or importance of the text and complement its
content, especially if it contains contact details.  If the name expands the context of the provided
information to a level where meaningful instructions and answers can be generated, it increases the
text's value.
Rating Scale:
1:  The text has no meaningful content, consists only of links, or serves as a placeholder without
useful information.
2:  The text contains links and headings with minimal relevant information, including outdated or
irrelevant data.
3:  The text is suitable for creating a limited number (fewer than 5) of general instructions but has
low value.
4:  The text is suitable for creating a significant number of instructions that reflect both general
and specific aspects of the organization's operations, with a sufficient share of relevant data.
5:  The text is highly structured, contains numerous relevant details (from 2022 and later), and
allows for the creation of a wide variety of valuable instructions.
Important:
The response must be a single number (from 1 to 5).  Comments, explanations, or additional information
are strictly prohibited.
File name:
***
Content:
***
```

Figure 2. Designed prompt for filtering irrelevant texts.

Next, three LLMs were selected: Gemma2-9b-it, LLaMA-3.1-70B-Instruct, and LLaMA-3.3-70B-Instruct. Prompts corresponding to all available text files were sent to each model using the NVIDIA LLM inference service using the OpenAI API. In addition to obtaining assessments for file filtering, we aimed to identify the most suitable LLM for instruction generation.

Due to input sequence length limitations imposed by some models, we restricted the evaluation to files smaller than 11 KB, a threshold determined through analysis of the file size distribution. For larger files, a default score of 5.0 was assigned, based on the assumption that longer texts are more likely to contain substantial and relevant content.

After querying the selected LLMs to evaluate the textual content extracted from the parsed HTML pages, we obtained the distributions of file evaluation scores, which are presented in Figure 3.

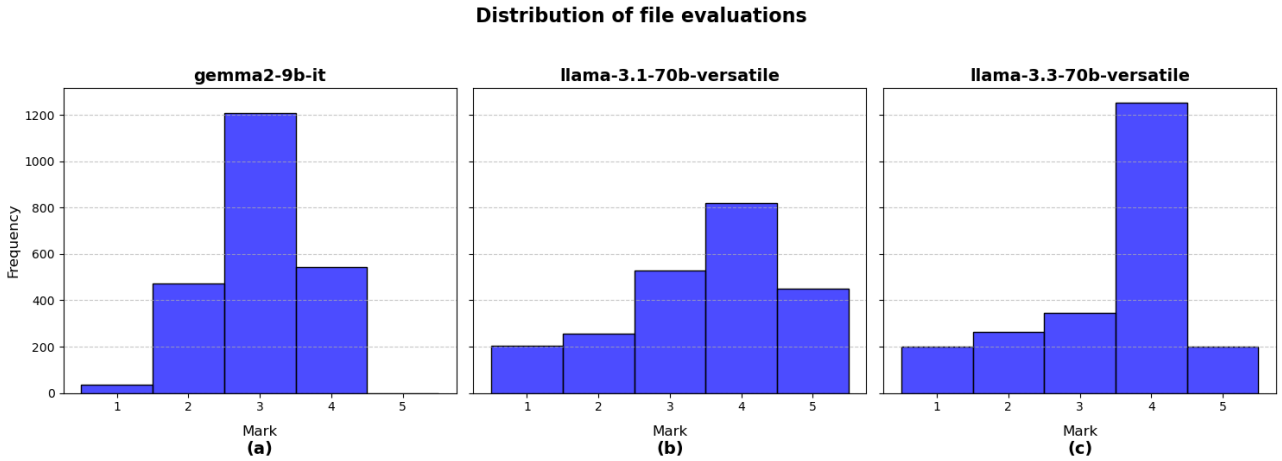

Figure 3. Distribution of file evaluations from Gemma2-9b-it (a), LLaMA-3.1-70B-Instruct (b), LLaMA-3.3-70B-Instruct (c).

As shown in the figure, all score distributions are right-skewed, with the mode occurring at the value just below the maximum. Notably, the Gemma2-9b-it model never assigned the highest score of 5.0, which may suggest its limited suitability for this type of domain-specific text evaluation. In contrast, the LLaMA-family models produced broadly similar distributions, with the most recent version, LLaMA-3.3-70B-Instruct, most frequently assigning a score of **4.0**.

Subsequently, we computed the average evaluation scores assigned by the models for each text file and analyzed the maximum divergence in scores for individual files. The greatest observed discrepancy between model evaluations was 3 points, occurring in 14 cases. Following a manual review of these instances, we concluded that the identified disagreements were not

significant enough to warrant the exclusion of the corresponding files from the datasets used for instruction generation and vector database construction.

Based on the analysis of the average score distribution across text files, we applied a thresholding criterion and excluded all files with an average score of ≤ 2.0. Such a low score typically indicates that either all models assigned a rating of 2.0 — implying that the text is nearly unusable for downstream tasks — or that the majority of models assigned a score of 1.0, marking the file as entirely irrelevant. Following this filtering step, **2,337** text files were retained, which we consider sufficient for constructing a qualitative instruction dataset. Given the assumption of approximately 10 input-output pairs per file, the resulting dataset is expected to contain over **23,000** entries. This outcome is particularly notable considering the narrow focus of our domain that contains only information related to a single university. For example, creators of RAG-Instruct reported assembling a dataset of 40,000 instruction pairs derived from the entire English Wikipedia corpus.

We additionally investigated the potential relationship between file size and the average evaluation score by constructing a correlation matrix, presented in Figure 4. As shown, the correlation between file size and average score is positive, with a value of **0.66**, indicating a moderately strong association. This finding supports the assumption that larger files are more likely to be assigned higher quality scores.

The analysis also demonstrates that models from the LLaMA family contributed most significantly to the overall average score, suggesting their higher suitability for the evaluation of raw crawled text data. Based on these results, we selected LLaMA-3.3-70B-Instruct as the primary model for instruction pair generation. The source code of the project, including the file evaluation pipeline, is open-source and available at `https://github.com/EliseevVadim/TextsEstimator`.

6. Generating Instructional Pairs

After having a list of relevant files, we proceed to construct a classical synthetic instructional dataset. In order to do this we decided using LLM prompting again. Unlike the previous section, we will utilize only one model — LLaMA-3.3-70B-Instruct, as it exhibited the best performance in evaluating the text files.

We designed a prompt that incorporates the file's title, its content, and the last update date. The model is instructed to generate the maximum number of domain-specific instruc-

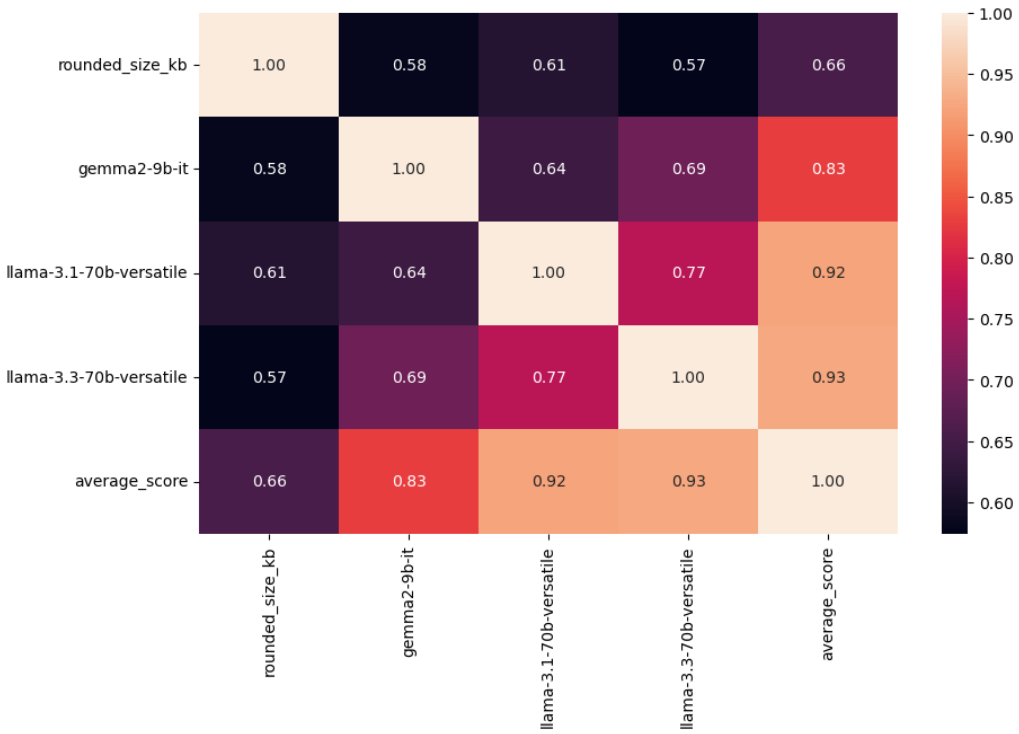

Figure 4. Correlation matrix between file size and model scores.

tional pairs, taking into account the suggested thematic directions and constraints. The original prompt also was built in Russian, it is available at `https://github.com/EliseevVadim/InstructionsGenerator/blob/main/data/prompts/instructions_creation_RU.txt`. The English translation of the prompt is presented in Figure 5.

For each relevant file, we submitted a prompt to the LLaMA-3.3-70B-Instruct model for instruction generation, following the procedure outlined in Section 4. This process resulted in the generation of 24,854 instructions, which constitutes more than half of the RAG-Instruct dataset.

Upon examination, we found that certain files, despite being deemed relevant through manual review, produced only a limited number of instructions. In light of this, we analyzed the distribution of instruction counts per file and decided to regenerate instructions for those files that initially yielded **three** or fewer instructions.

On the second pass of instruction generation, we adjusted the seed value and reduced the temperature (from 1.0 to 0.7) and top-p (from 1.0 to 0.8) parameters when calling the LLM. These lower values of temperature and top-p lead to more deterministic responses from the model, and are expected to minimize the occurrence of variances such as empty outputs or the generation of low amount of instruction pairs for files that contain enough of relevant

information.

```
Based on the HTML page data obtained from parsing the website of Donetsk State University:
Page Title:
***
Date of Last Content Update:
***
Content:
***
Generate the maximum possible number of instructions with model responses (including both long and short responses) that fully cover the provided text,
especially its factual aspects.  The instructions should be suitable for fine-tuning an LLM as an intelligent assistant for Donetsk State University.
Each instruction must be formatted as JSON with input (question) and output (answer).  Avoid generating irrelevant content that is not specifically
related to the activities of Donetsk State University.  If the text contains no meaningful information (e.g., a list of links, advertisements, or other
utility data), generate a stub with "error":  "This text does not contain useful content for generating instructions".
IMPORTANT!
The provided content often pertains to specific local aspects of the university's activities.  Therefore, DO NOT generate overly generic questions such
as "What programs are offered at DSU?" or "Who is a professor at DSU?" because the objects described in specific files are not the only ones in the
university's context, and such questions are counterproductive.
If the content mentions a faculty member, instead of a question like "Who is a professor of the Department of Physical Education and Sports on the
<title> page?" with the answer "<Name of the professor>", generate "Who is <Name of the professor>?" with the response based on the page content.
Do not respond with "A professor described on the <title> page"; instead, use the actual content of the page.  This logic applies to all other
instructions as well.
Additionally, the page title is provided, which can better reveal the text's content.  Use it and its connection to the text to construct
higher-quality instructions.
Moreover, the date of the last content update for each page is indicated.  Use this information to improve the quality of the generated instructions.
Avoid asking questions like "When was the page content last updated?" as this question is not relevant; the date should only be used to add context to
the content.
Avoid using pronouns or generic nouns when generating questions.  Instead, use named entities from the page title or as appropriate to the content.
Additionally, when listing facts in the model's response, include all available facts without truncating them with phrases like "and others."
The model being trained will serve as an intelligent assistant for an educational organization.  Therefore, this aspect must be the primary
consideration when creating instructions and responses-all must be in the context of the specific educational organization being referenced.
The text was obtained by automatically crawling the university website.  This prompt is also generated automatically by substituting data obtained
during the crawl.  Therefore, it may contain irrelevant or useless information.  For such texts, generate JSON with the error field containing the
value "This text does not contain useful content for generating instructions." Do not generate instructions for meaningless content!
Possible Types of Instructions:
1.  Questions to extract facts or details from the text (if factual material is present, questions must cover all of it).  2.  Questions requiring
analysis or comparison of information.  3.  Any other questions applicable for fine-tuning an intelligent assistant for DSU. Instructions and
responses must include both concise and detailed formats, fully covering the entire text (every part of the text MUST strictly participate as part of a
response).
Output is allowed ONLY in JSON format; any other textual content is strictly prohibited!  All output must be presented in Russian language only!
```

Figure 5. Designed prompt for generating instructions.

As a result, we generated an additional 806 instructions from 91 "problematic" files, increasing the total dataset size to **25,633** instructions. This dataset volume is deemed sufficient for the subsequent fine-tuning of the LLM. The source code of the instructions generation project is available at `https://github.com/EliseevVadim/InstructionsGenerator`.

7. Extending Instruction Dataset By Relevant Context

After successfully creating classical instruction dataset, firstly we built the vector database over relevant text files selected at Section 5. Before constructing the vector database, we needed to select an appropriate retriever model. The "best" retriever should possess an encoder-only architecture, as described by Karpukhin et al. [7], since our goal is to obtain high-quality embeddings rather than generate text output, which is the function of a decoder. Additionally, an effective retriever must be capable of processing long sequences, as emphasized by Khattab and Zaharia [8], and generating embeddings with a high dimensionality d.

Since we are working with texts in Russian and our intelligent assistant will be producing responses in Russian as well, it is preferable to use a model specifically trained on Russian texts rather than one adapted from other languages. Therefore, we selected the Giga-Embeddings-instruct model as the retriever.

The selected model, Giga-Embeddings-instruct, is designed to process contexts up to 4,096 tokens and generate embeddings with a dimensionality of d=2,048. It is based on the GigaChat-Pretrain-3B architecture, where decoder attention has been replaced with encoder attention, and incorporates Latent Attention Pooling [9] for aggregation. This model is suitable for the task at hand, as it contains only 2.5 billion parameters, ensuring efficient execution on personal devices. Despite its relatively small size, the model demonstrates high-quality vectorization and ranks second in the ruMTEB benchmark as of December 27, 2024.

Following the selection of the retriever model, the next step was to determine the chunk size and chunking strategy for document processing. Since the vector database at this stage is intended solely for the creation of a fine-tuning dataset, we opted to divide the documents into relatively small chunks of **500 tokens**, with a **50-token** overlap. This approach was selected to prevent the model's context window from becoming overloaded during fine-tuning. The tokenizer of the LLaMA-3.1-8B model was employed to define the chunk size, as this is the model slated for fine-tuning with the dataset. Furthermore, to avoid redundancy and ensure the quality of the vector database, it was essential to remove duplicate text files prior to chunking. Duplicate content could result in identical fragments being retrieved during vector search, negatively affecting the efficiency of the RAG system.

Upon removing duplicates and splitting the documents into chunks, we obtained a total of **9,935** chunks. Using them, we constructed a vector database using the FAISS framework. During the embeddings generating, we applied normalization [19] enabling the use of cosine

similarity [16] for effective similarity search over the vector database.

After constructing the vector database, we proceeded to enhance our instructional dataset, which consists of input-output pairs, by incorporating relevant context. This context was obtained through the retriever model, which conducted a similarity search within the vector database based on the input field. In this manner, we will adapt the model to the specific retriever in use and train it to extract the correct answer from the retrieved documents during fine-tuning. In this process we will provide the model with **three** documents along with the input. This limitation was imposed to prevent overloading the generator's context window, as using a greater number of documents could introduce noise and negatively affect the quality of the generated answers [6].

For each input x from our synthetic instruction dataset we retrieved three semantically closest chunks c using cosine similarity $\cos(\theta)$. It can be found as:

$$\cos(\theta) = \frac{c \cdot x}{\|c\|\|x\|} \tag{1}$$

where $c \cdot i$ is a scalar product of vectors that can be found as:

$$\mathbf{c} \cdot \mathbf{x} = \sum_{i=1}^{d} c_i x_i \tag{2}$$

and $\|c\|$, $\|x\|$ are Euclidian norms of vectors c and i that can be found as:

$$\|\mathbf{c}\| = \sqrt{\sum_{i=1}^{d} c_i^2}, \quad \|\mathbf{x}\| = \sqrt{\sum_{i=1}^{d} x_i^2} \tag{3}$$

The cosine similarity takes values in the range $[-1, 1]$, where 1 indicates that the vectors are identical, 0 corresponds to orthogonality (i.e., no semantic similarity), and -1 signifies that the vectors are diametrically opposed. Consequently, the higher the value of $\cos(\theta)$, the closer the input i to chunk c. In this work, for each input, we retrieved the three passages with the highest similarity scores. Then we added these passages to our dataset as the *context* field. The source code of extending instructional pairs with relevant context from the vector database is open-source and available at `https://github.com/EliseevVadim/InstructionsGenerator`.

8. Future Work

Following the creation of the RAG-based instruction dataset, we plan to fine-tune the generator model using the LoRA approach [5]. For this task, we selected the LLaMA-3.1-8B model,

an open-source model that combines relatively high performance with a moderate number of parameters, enabling fine-tuning process even on limited computational resources.

In addition, we intend to develop a new vector knowledge base. It will include not only the documents retrieved by the method described in this work but also text extracted from various file formats gathered through website crawling.

Given the need to store substantial volumes of vectorized textual data and to ensure regular database updates, we are considering persistent storage solutions. We intend to use pgvector for this purpose. For this moment we already developed the tool, allowing administrators managing the vector database by adding a new documents and removing irrelevant text chunks from it.

We anticipate that applying the proposed method for generating a RAG-based instruction dataset will significantly enhance the generator's performance. Moreover, the model is expected to maintain high-quality answers even as domain knowledge evolves, by effectively leveraging an easily updateable knowledge base.

9. Conclusion

In this study, we proposed a method for constructing a RAG-based instruction dataset intended for fine-tuning a generator within the RAG system of an intelligent assistant for a university. The main contribution of our work lies in the development of a system specifically trained to operate on domain-specific data related to DonSU. Unlike previous studies that primarily utilized existing datasets, we created a dataset from scratch by extracting information from the university's website and applying a sequence of preprocessing and transformation steps outlined in this paper. We described the full pipeline, from parsing raw web data to enriching a classical instruction dataset with additional context retrieved from a vector database based on semantic similarity to the input. As a result, we compiled a dataset comprising over **25,000** input-context-output triplets, ready for use in fine-tuning the generator model.

In future research, we intend to fine-tune the LLaMA-3.1-8B model on the constructed dataset, expand the vector knowledge base using the full set of collected documents, and implement a complete RAG pipeline. This system is expected to deliver high-quality, contextually relevant responses, maintaining robustness regardless of updates are made to the underlying knowledge base.

Funding The study was carried out with the financial support of the Ministry of Education and Science of the Russian Federation within the framework of the state task on the topic "Development and improvement of intelligent classification and forecasting methods for pattern recognition and modeling of information processes" FREM-2024-0001 (Registration number 1023111000141-9-1.2.1)

References

1. Barnett S. et al. Seven failure points when engineering a retrieval augmented generation system //Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering-Software Engineering for AI. 2024. P. 194–199.
2. Brown T. et al. Language models are few-shot learners //Advances in neural information processing systems. 2020. Vol. 33. P. 1877–1901.
3. Gururangan S. et al. Don't stop pretraining: Adapt language models to domains and tasks // Proceedings Of The 58th Annual Meeting Of The Association For Computational Linguistics. 2020. P. 8342–8360.
4. Houlsby N. et al. Parameter-efficient transfer learning for NLP //International conference on machine learning. PMLR, 2019. P. 2790–2799.
5. Hu E. J. et al. Lora: Low-rank adaptation of large language models //ICLR. 2022. Vol. 1. N. 2. P. 3.
6. Izacard G. et al. Unsupervised dense information retrieval with contrastive learning // Trans. Mach. Learn. Res. 2022.
7. Karpukhin V. et al. Dense Passage Retrieval for Open-Domain Question Answering // EMNLP (1). 2020. P. 6769–6781.
8. Khattab O., Zaharia M. Colbert: Efficient and effective passage search via contextualized late interaction over bert // Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval. 2020. P. 39–48.
9. Lee C. et al. Nv-embed: Improved techniques for training llms as generalist embedding models // Proceedings of the International Conference on Learning Representations (ICLR). 2025.
10. Lewis M. et al. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension // Annual Meeting of the Association for Computational Linguistics. 2019.
11. Lewis P. et al. Retrieval-augmented generation for knowledge-intensive nlp tasks // Advances in neural information processing systems. 2020. Vol. 33. P. 9459–9474.
12. Li X. L., Liang P. Prefix-tuning: Optimizing continuous prompts for generation // Proceedings Of The 59th Annual Meeting Of The Association For Computational Linguistics And The 11th International Joint Conference On Natural Language Processing (Volume 1: Long Papers). 2021. P. 4582–4597.
13. Lin X. V. et al. Ra-dit: Retrieval-augmented dual instruction tuning // The Twelfth International

Conference on Learning Representations. 2023.

14. Mikolov T. et al. Distributed representations of words and phrases and their compositionality // Advances in neural information processing systems. 2013. Vol. 26.
15. Ouyang L. et al. Training language models to follow instructions with human feedback //Advances in neural information processing systems. 2022. Vol. 35. P. 27730–27744.
16. Salton G., Wong A., Yang C. S. A vector space model for automatic indexing // Communications of the ACM. – 1975. Vol. 18. N. 11. P. 613–620.
17. Vaswani A. et al. Attention is all you need //Advances in neural information processing systems. 2017. Vol. 30.
18. Wang Y. et al. Self-instruct: Aligning language models with self-generated instructions // Annual Meeting of the Association for Computational Linguistics. 2022.
19. Yi J., Kim B., Chang B. Embedding Normalization: Significance Preserving Feature Normalization for Click-Through Rate Prediction // 2021 International Conference on Data Mining Workshops (ICDMW). IEEE, 2021. P. 75–84.
20. Zaken E. B., Ravfogel S., Goldberg Y. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models // Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers). 2021. P. 1–9.