

**M A S A R Y K
U N I V E R S I T Y**

FACULTY OF INFORMATICS

**Automation of the Request
for Quotation Process using
Large Language Models**

Master's Thesis

BC. MARTIN KŇAŽOVIČ

Brno, Spring 2025

**M A S A R Y K
U N I V E R S I T Y**

FACULTY OF INFORMATICS

**Automation of the Request
for Quotation Process using
Large Language Models**

Master's Thesis

BC. MARTIN KŇAŽOVIČ

Advisor: doc. RNDr. Petr Sojka, Ph.D.

Department of Visual Computing – Faculty of Informatics

Brno, Spring 2025



Declaration

Hereby, I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during the elaboration of this work are properly cited and listed in complete reference to the due source. While preparing this thesis, I used Grammarly for grammar corrections and generative AI for faster code writing and debugging.

Bc. Martin Kňáždovič

Advisor: doc. RNDr. Petr Sojka, Ph.D.

Acknowledgements

I want to thank my supervisor for his guidance throughout the thesis process and for the valuable advice provided during the PV211 and PV212 courses. I am also grateful to Jan Najvárek for his advice, leadership, and the opportunity to work on this project. Finally, I would like to thank everyone who supported me and contributed to the refinement of this work in any way.

Abstract

The request for quotation (RFQ) process is a tedious business procedure for sales departments since it usually involves processing customer quotations in unstructured text, which has to be mapped to the company product catalog and sent back to the customer as a quotation. This thesis aims to partially automate this process by employing large language models (LLMs) for product structurization on customers' emails and automating the product mapping process with embedding search. The initial part of this thesis is dedicated to zero-shot prompting product extraction on RFQs for special geotechnical constructions for the Czech company GEOMAT, focusing on prompt engineering techniques to enhance the extraction pipeline. The next part discusses low-rank adaptation fine-tuning of open-source LLaMA LLMs on synthetic datasets based on GEOMAT data. The final phase of this work extends the pipeline with a full-fledged application to help sales teams process incoming RFQs by extending the previous pipeline with embedding search based on cosine similarity and reranking of matched products, enabling quotation assistants to provide necessary documents and easily select products for quotation.

Keywords

Request for quotation, Request for proposal, Large Language Models, Document Analysis, Document Automation, Language model fine-tuning, Synthetic data generation, Prompt engineering, Data Structuring, GPT-4o, LLAMA 3.1, Low-Rank Adaptation, Semantic Similarity, Business process automation, Enterprise AI, Procurement automation

Contents

1	Introduction	1
1.1	General Overview of RFQ Process	1
1.2	Challenges of the RFQ Process	1
1.3	Opportunities for Improvement	3
1.4	Industry Partners	3
1.5	Structure of the Thesis	4
2	State of the Art	6
2.1	Natural Language Processing in the Context of RFQ Automation	6
2.2	Utilizing Large Language Models for Information Extraction	6
2.3	An Empirical Study on Information Extraction using Large Language Models	7
2.4	Automated Analysis of RFQs using Natural Language Processing for the Technology Domain	8
2.5	Generative AI for Sales	8
2.6	Large Language Models for Generative Information Extraction: A Survey	9
2.7	Automate RFQ Response Generation Process Using FastText Word Embeddings and Soft Cosine Measure	10
2.8	ReqNet: An LLM-Driven Computational Framework for Automated Requirements Extraction from Unstructured Documents	10
2.9	Conclusion	11
3	Implementation Design and Methodology	13
3.1	Data Characteristics	13
3.2	Application Flow	19
3.3	Extraction Process	21
3.4	Design of Model Fine-Tuning Experiment	25
3.5	Comprehensive RFQ Automation	30
3.5.1	Storing the Product Data	30
3.5.2	Searching Methodology	35
3.5.3	Offermate's Frontend	37

4	Evaluation of Solutions and Experiments	42
4.1	Testing the Solution	42
4.2	Model Fine-Tuning	44
5	Discussion and Future Work	57
5.1	GEOMAT's Solution	57
5.2	Fine-Tuning Experiment	58
5.3	Offermate – RFQ Process Automation	59
6	Conclusion	61
A	Overview of Thesis Archive	63
	Bibliography	64

List of Tables

3.1	Most frequently inquired parameters for different GEO-MAT product categories	15
4.1	Cross-Validation results of all fine-tuned folds ranked . .	47
4.2	Model Performance Benchmark on TEST and A100 Datasets	48

List of Figures

3.1	Request for quotation in the Czech language for road construction from GEOMAT	13
3.2	Redacted example of GEOMAT's RFQ for construction material delivery to Bulgaria.	14
3.3	Distribution of absolute product count per category . . .	16
3.4	Distribution of products per RFQ	17
3.5	Distribution of unique product categories per RFQ	18
3.6	Most commonly appearing duos of product categories in RFQs	19
3.7	GEOMAT system diagram describing application flow . .	20
3.8	The Offermate system diagram describing applications flow, split into product extraction and product matching endpoints.	35
3.9	UI showcase of the proposal page where the quotation assistant should upload relevant RFQ documents to start the quotation process	38
3.10	Example of results from the product extraction phase based on the default input data as shown in Figure 3.9 . .	39
3.11	Example of matched product based on the extracted data, shown in the Figure 3.10	40
3.12	Showcase of quoted products in Offermate	41
4.1	Training loss for all folds on 75 training steps	45
4.2	Training loss for all folds on 75 training steps grouped . .	46
4.3	Validation for all folds on 72 validation steps	46
4.4	Distribution of error types per selected models	49
4.5	Comparison of F1 score across best performing models and LLAMA baseline	51
4.6	Comparison of weighted accuracy across best-performing models and LLAMA baseline	52
4.7	Comparison of precision, recall and F1 relation for GPT models	53
4.8	Comparison of precision, recall and F1 relation for LLAMA models	54
4.9	Comparison of precision, recall and F1 relation for GPT-4o-mini and best performing folds	55

1 Introduction

Request for quotation (RFQ) is a tedious and error-prone process that occupies the sales departments of many companies daily. It is important for companies that operate on an RFQ basis since their success in converting quotations into orders directly affects their earnings and growth. That is why we shall discuss how the RFQ process works, its caveats, and how we plan to improve this process with generative AI.

1.1 General Overview of RFQ Process

First of all, let us illustrate the process itself. The process starts with a potential customer sending an RFQ to a company, often composed of an email and attachments with additional specifications. The email usually contains general information, such as the reason for the request (e.g., construction, invitation to tender) and the expected delivery of the requested goods and services. The attachments, most often PDF files, specify the products in question in detail – product names, descriptions, product codes, and required amounts. However, collateral files, such as technical documentation, drawings, or tender information, are often included, which can complicate and slow the quotation process.

The RFQ must then be processed by a quotation specialist who will review all the provided documents and extract all relevant information to check if the business can satisfy the customer's needs. Afterward, the quotation specialist generates the proposal, which may or may not include the exact products or services the customer requested, but instead, available ones. The last step of this process is to price and finalize the quotation and send it to the customer for confirmation or further negotiations.

1.2 Challenges of the RFQ Process

There are many caveats to the RFQ process, which, when not handled properly, can result in losing the deal. Therefore, it is essential to identify and resolve these challenges appropriately. One of the most

critical aspects of securing the deal is the required time to answer, the interval from when the RFQ is received until it is sent to a potential customer.

In the Harvard Business Review, James B. Oldroyd et al. researched companies' ability to respond to leads and the time it took them to answer.¹ Based on data from 2241 U.S. companies, they report that the queries they received when reacting to a lead within one hour were seven times as likely to qualify as prospects. Other notable findings are that only 37 % of companies managed to respond within one hour, and the average response time was 42 hours. When discussing RFQ, the required time to answer can range from minutes to days, giving faster responders an edge over slower ones.

Tatu Virtanen et al. also looked at different types of leads [1] and reached a similar conclusion to Oldroyd et al.'s work (regarding the effect of time to respond on lead outcome). They analyzed IT opportunities, and one of the examined variables was „Lead Time“, which measured the time in quarters from receiving a lead to its conclusion (i.e., being won, lost, or canceled). In their work, they reported that longer lead time negatively impacted sales success. Specifically, increasing the „Lead Time“ by one quarter leads to an increase in the probability that the lead will be canceled versus being won and an increase in the probability of losing the lead versus winning it. Therefore, minimizing this metric can significantly increase sales, since if it takes too long to answer an RFQ, the potential customer will most likely accept a deal from another company.

There remain many other problems in the RFQ process, such as the nature of this process being prone to human-related errors when processing a substantial amount of quotation inquiries, as well as other issues like team scaling and RFQ distribution within the processing team. Still, our work will focus mainly on improving the time required for an answer.

1. James B. Oldroyd et al., The Short Life of Online Sales Leads, Available at: <https://hbr.org/2011/03/the-short-life-of-online-sales-leads>

1.3 Opportunities for Improvement

Since we have already identified the issues of the RFQ process, let's discuss the opportunities for improvement. The main problem from our point of view is the time required to process RFQs, since if companies take too long to answer them, it usually results in losing a deal. The current and most straightforward solution to this is to increase the sales department size and hire more quotation specialists, which, on the other hand, incurs more man-hour costs for the company. What seems promising in this field is not just hiring more people for the same role but improving the efficiency of the process itself. So, a more modern solution could be to divide this process and create at least partial automation for parts that could be automated.

To achieve this, we propose an approach that will first utilize large language models (LLMs) to partially automate and enhance this process and later use information retrieval methods and models to go further. The LLMs have proven over the past years that they can be used for more than just chatting but even for coding, math, research, and other fields. Based on the nature of RFQ data being unstructured text or in a form that can be transformed to unstructured text (i.e., textual images, quotations in PDFs), we see great potential in utilizing LLMs for the extraction of key information from this data, which could speed up the process itself.

1.4 Industry Partners

Improvements in this field would benefit companies dealing with medium to large-scale RFQ processing. Therefore, we have partnered with Artin Ltd., a Czech software company, respectively, with its co-founder Ing. Jan Najvárek, Ph.D., to address this issue in the local Czech industry. Firstly, we have established cooperation with GEO-MAT Ltd., a company specializing in special geotechnical constructions and processes, which receives a few hundred RFQs monthly. We have developed our first solution for them, which ingests their RFQ data, extracts important product information, and provides it over an API. Later, we cooperated with the largest Czech distributor of electrotechnical material, ELKOV Ltd., to whom we extended

and improved our in-house product, which we developed based on GEOMAT's solution. Therefore, we will release some data from the mentioned companies anonymized based on the agreement with Mr. Najvárek.

1.5 Structure of the Thesis

In this chapter, we describe the fundamentals of the RFQ process, challenges, and possible improvements with automation by means of LLMs and declare our industry cooperation.

In the next Chapter 2 on page 6, we will describe state-of-the-art for the RFQ process, its connection to natural language processing, and relevant literature, which will be the basis for our upcoming work.

Then, in Chapter 3 on page 13, we will describe all our solutions and experiments, split into three parts. First, we will focus on utilizing LLMs for product extraction and categorization of RFQ data in cooperation with GEOMAT. The goal of this solution will be to create a product extraction pipeline that will ingest RFQ from GEOMAT and provide structured product data over API.

The next part of this thesis will be dedicated to improving the LLMs for product extraction by fine-tuning the LLAMA 3.1 8B model and comparing it to the already available bigger model ChatGPT-4o, which we utilized in the first part. The problematic part of this is that fine-tuning LLMs requires a large quantity of data (or at least data of smaller size but higher quality). At the same time, there is a lack of annotated data from our industry partners, whom we disclosed in the previous section. This issue is partially solved by creating a synthetic dataset that works like [2] and using techniques such as Low-Rank Adaptation (LoRA [3]) to maximize the efficiency of our available options. The technique LoRa and model fine-tuning will be disclosed more closely in Section 3.4 on page 25.

The last part will be dedicated to creating software allowing a quotation specialist to extract important data from incoming RFQs and map it to the existing product catalog, further improving this process. This way, the quotation specialists will be required to verify the extraction process, price suggested products, and recommend other products from their catalog if necessary. We will create this

software by extending the first part of this work (the LLMs for data extraction) and combining it with the company's product catalog and information retrieval techniques and models, which we will disclose in Section 3.5 on page 30.

In Chapter 4 on page 42, we will describe testing methodologies, provide results of the GEOMAT solution, explore the fine-tuning experiments with an in-depth performance review, and discuss results of selected models.

This work's possible improvements and limitations are then discussed in Chapter 5 on page 57, and the last Chapter 6 on page 61 summarizes our achieved results and completed work.

2 State of the Art

This chapter will discuss common approaches to automating the RFQ process. Several studies have used LLMs and other natural language processing (NLP) techniques for information extraction on textual data. However, literature on the complete automation of the RFQ process remains scarce. Therefore, we will first describe how NLP is relevant to the RFQ process and then focus on the relevant publications.

2.1 Natural Language Processing in the Context of RFQ Automation

Natural Language Processing is a field of artificial intelligence focusing on techniques that allow machines to understand text or voice in natural language. The main goal of this thesis is to partially automate the RFQ process, which consists of processing natural language texts. There is a whole subtask dedicated to making sense of information in documents called information extraction (IE), which will, in our use case, focus on product data that is found in RFQs.

This process of information extraction, which transforms unstructured text into structured data, has been described by Yang et al. in a comprehensive survey on IE based on deep learning [4]. Even though they discussed transform-based BERT models, which were precursors to today's LLMs, they have stated that „pre-trained language models have achieved superior performance in a variety of NLP tasks, including entity and relationship extraction“. Therefore, we want to expand on this and explore the possibilities of current LLMs regarding product data extraction.

2.2 Utilizing Large Language Models for Information Extraction

The paper's [2] authors tried to extract meaningful information from real estate transaction documents utilizing the LLaMA 3 model, which they fine-tuned using synthetic and real contract data. The authors

experienced similar problems with data scarcity since legal documents are often private. Still, they generated synthetic data by distillation from LLM, grounded by their limited actual data. Then, they proceed with parameter-efficient fine-tuning of Llama-3.1-8B on their synthetic dataset. The authors reported an F1 score of 0.722 for llama-3-8b-instruct and 0.874 for fine-tuned versions of the same model. The most significant difference, which highlights the importance of fine-tuning, can be observed in the match rate metric, where they consider the answer correct if it obtained an F1 score greater than 0.7, going from 0.63 on the baseline model to 0.968 on the fine-tuned version.

The main takeaways of this work are that even relatively small 8B parameter language models can be used for successful data extraction, that we can successfully use small real-world datasets to generate larger synthetic ones, and that the model can learn useful patterns even on synthetic data.

2.3 An Empirical Study on Information Extraction using Large Language Models

Authors of a comprehensive empirical study [5] evaluated OpenAI’s GPT 3.5 and GPT 4 models on fourteen different information extraction subtasks with sixteen different datasets for three given prompting scenarios: zero-shot, few-shot in-context learning and few-shot chain-of-thought (COT). The evaluation was then compared with SOTA performances obtained from literature, and results show that GPT 4 surpassed SOTA performance in zero-shot setting only for the Aspect-level Sentiment Classification, confirming findings from previous papers that further fine-tuning may be required to achieve SOTA performance. The authors also note that ICL improves results compared to zero-shot settings. The performance achieved in the COT scenario did not consistently enhance performance over simpler ICL, and both methods still lag behind SOTA.

2.4 Automated Analysis of RFQs using Natural Language Processing for the Technology Domain

The paper's [6] authors' main goal was the identification of new emerging technology trends in RFQs from the United States Department of Defense. The authors proposed two approaches, one utilizing models from the classical NLP package SpaCy and a more modern approach utilizing BERT models. Models from both categories are presented in their base versions and fine-tuned ones. Fine-tuning of these models was done based on the "custom technology taxonomy" dataset, which, as the authors' results indicated, proved helpful in minimizing the number of incorrect entity classifications. The final results of this analysis report that the SpaCy Fine-tuned Small model achieved the highest precision of 0.675 from all presented models, even beating the Bert Base Fine-tuned NER, which achieved a precision of 0.526.

2.5 Generative AI for Sales

This study [7] author created a proof-of-concept retrieval augmented generation (RAG) system to create responses for RFQs. They mainly focused on investigating the quality of retrieved RAG contexts and created responses based on different document chunking techniques and sizes; for the experiment, authors used recursive character chunking with document sizes of 250, 500, 750, and 1000 with 50 character overlap for each. They also investigated the parent-child chunking method with document sizes 500 and 750 for children documents, 1500 for parent documents, and 50 character overlaps for each document. The experiment was then split into two parts based on the input data – simple and advanced requirements. The results of this study are that parent-child chunking with 750 characters for children documents outperformed all other configurations in quality of generated response on simple requirements, and recursive character chunking with 1000 characters for documents outperformed all other configurations on advanced requirements. However, averaged on all sub-metrics, the overall winner is parent-child chunking.

2.6 Large Language Models for Generative Information Extraction: A Survey

The automation of the RFQ process can be treated as the task of IE since the chosen LLM must be able to extract products and their parameters from the customer’s inquiry. The authors of this comprehensive survey [8] divided the IE task into three subtasks: Named entity recognition (NER), Relation extraction, and Event extraction. This thesis will focus only on the observations about the NER subtask and other general knowledge. The authors observed that, despite similar backbone LLMs being used to train more specialized models, the in-context learning paradigm models were significantly outperformed by supervised fine-tuning and data augmentation paradigms. In fine-tuning the models, the authors discussed the following techniques.

- **Inverse generation** – This technique utilizes LLMs to generate questions based on provided structural data. We believe a similar approach could generate synthetic RFQs based on existing product catalogs, which could help reduce the (annotated) data scarcity problem.
- **Data distillation** – The following technique utilizes the power of a larger language model to generate synthetic data for fine-tuning another model. Data distillation could also reduce the data scarcity problem with prompt grounding via RAG or in a more straightforward context.
- **Chain-of-thought** – The last discussed technique queries the model repeatedly to generate reasoning on the given problem and validate the approach further. The CoT could be helpful in unclear RFQs where it is hard to determine the customer’s needs from the inquiry.

From the general takeaways, the authors recommend focusing on prompt design since multiple works confirm its significant impact on LLM performance.

2.7 Automate RFQ Response Generation Process Using FastText Word Embeddings and Soft Cosine Measure

This paper [9] presents an alternative approach to answering RFQs by creating embeddings from historical RFQ datasets and providing answers to inquiries by looking up the most relevant historical answers. The authors of this paper evaluated multiple types of word embeddings. They decided to use FastText embeddings since they claim that FastText generates better embeddings for rare words and can construct embeddings even for words not initially present in the training data. For the similarity metric, authors decided to use soft cosine measure (SCM), which they preferred to Word Mover's distance similarity based on SCM's more efficient computation (more on the topic of complexity of SCM can be found here [10]). The authors then reported 70+ percent correctness based on a manual review of answers from the testing dataset. The main takeaway from this paper is that for some business domains, the inquiries remain consistent over time, and even more straightforward solutions may be sufficient for solving partial RFQ automation.

2.8 ReqNet: An LLM-Driven Computational Framework for Automated Requirements Extraction from Unstructured Documents

The authors of this paper [11] were trying to solve a problem similar to ours: information extraction from unstructured documents, with the main difference being the data domain and that they performed classification on top of extracted information to determine if input documents were requirements or not. The authors used seven different LLMs in standalone and ensemble regimes for this experiment. The extracted information was classified in two settings: self-classification by LLM, which performed the extraction, and via an external classifier (long short-term memory network, gated recurrent unit, and their variants). The mentioned approaches were then evaluated on

two benchmark datasets, PURE and Dronology, and an independent request for information test set. The results are reported as follows:

- **Standalone LLM-based predictive pipelines** – Even though there were no decisive winners, the Longformer_{large} model appeared in top-performing models on both PURE and Dronology datasets.
- **LLMs with External Classifier-based Predictive Pipelines** – The authors reported that integrating deep learning models as external classifiers significantly helped on Pure and Dronology datasets. However, the performance of the LLMs mostly deteriorated on the RFI test set.
- **Top performing standalone LLMs vs LLMs with external classifiers** – The authors concluded that LLMs with external LSTM and GRU classifiers outperformed the standalone LLMs approach.
- **Ensamble of multiple LLMs with external classifiers** – Based on the reported accuracy, the ensemble of ALBERT + BERT + XLNet with an external LSTM classifier outperformed the same configuration, which used a GRU classifier on the Dronology dataset and the RFI test set.

Lastly, the authors evaluated their proposed model ALBERT + BERT + XLNet + LSTM against a plethora of models and machine learning algorithms (ChatGPT, Gemini, Bert, XLNet,...) on all three mentioned datasets, with their proposed model performing best in each scenario. The main takeaway from this paper is that an ensemble of multiple LLMs might prove helpful when solving information extraction tasks.

2.9 Conclusion

Our main conclusion from the presented papers is as follows:

- Most of the above-mentioned papers performed fine-tuning of their models to achieve reasonable performance, while simpler

approaches like direct utilization of ChatGPT to solve information extraction reported worse performance than proposed models or SOTA performance on a given task.

- The researchers [2, 6] managed to achieve sufficient performance for their use-cases even with relatively smaller models, which opens a question: Can smaller models provide reasonable performance in information extraction?
- Historical RFQs could be helpful for the automation of the RFQ process [6] and can be used to create more synthetic data required for fine-tuning [2].
- For the indexation of documents in a vector database, it's better to use larger document chunks, and techniques like parent-child chunking could improve overall performance. [7]
- Ensembles of multiple LLMs could improve the performance of the information extraction process. [11]

3 Implementation Design and Methodology

We have already introduced the RFQ process, its caveats, and possible improvements we want to implement together with SOTA for the RFQ and request for proposal process. Before discussing our realization, we should establish a theoretical backbone of the technologies used. Specifically, we will discuss the nature of the data we have from our industry partners, database technologies used, means of creating synthetic data, and information retrieval models we have used for product matching.

3.1 Data Characteristics

One of the most important aspects of any AI or machine learning application is always its data. As we have already discussed, the data is mainly unstructured text or attachments, which can be transformed into unstructured text, but let's look at what actual production data from GEOMAT looks like.

Aspe Firma: [redacted]
Příloha k formuláři pro ocenění nabídky

Stavba: [redacted] - obchvat, oprava silnice
číslo a název SO: [redacted] Výměna konstrukce vozovky
číslo a název rozpočtu: [redacted] Výměna konstrukce vozovky

Poř. č. pol.	Kód položky	Varianta položky	Název položky	jednotka	Počet jednotek	CENA	
1	2	3	4	5	6	jednotková	celkem
						7	8
38	289971		OPLASTĚNÍ (ZPEVNĚNÍ) Z GEOTEXTILIE netkaná separační geotextilie typu S2 dle TP 97, min. 300g/m2 (mech. odolnost proti proláčení min. 3kN) I/16: 1388*1,2*1,1*1,2=2 198,592 [A] Rampa C krajnice a vsakovací ryha: 16,5*1,2*1,1*1,2+34*(4)*1=162,136 [B] vyztužený svah I/16: 9*(258)+365=2 687,000 [C] Celkem: A+B+C=5 047,728 [D]	M2	5 047,728		0,00
54	57476		VOZOVKOVÉ VÝZTUŽNÉ VRSTVY Z GEOMŘÍŽOVINY S TKANINOU sklovláknitý pletený geokompozit min. 100/100 kN/m a protažení do porušení nejvýše 3%, oka 25x25 mm, se stabilitou vázaných bodů min. 50 N, bod měknutí ochranného povlaku min. 230 °C a netkanou PES textilií 20-40 g/m2 a CBR min. 30 N pro zajištění instalace. Výztužná vložka bude splňovat požadavky ČSN EN ISO 10319 a TP 115. v napojení různých vozovkových souvrství silnic I/16 a I/32 3*11,5*2=69,000 [A]	M2	69,000		0,00

Figure 3.1: Request for quotation in the Czech language for road construction from GEOMAT

As seen from Figure 3.1, some RFQs are delivered in a very clean format, with precise specifications of what is requested from the company – products with detailed parameters, in a certain quantity and quantity type, just waiting for the price specification.

From: [REDACTED], Bulgaria
Sent: [REDACTED]
To: [REDACTED]
Subject: Request - [REDACTED]

Dear Helena,

How are you, I hope you are doing great?
We are following now a project with the need of
16 000 m2 self-adhesive asphalt reinforcing grid
100 / 100 and we are looking for alternative od
Adforce with parameters from below:

[cid:image005.jpg@05CACD0B.30A71CC0]

Would be possible for you to give an
alternative?
Best regards,
[REDACTED]

Figure 3.2: Redacted example of GEOMAT's RFQ for construction material delivery to Bulgaria.

This email example in Figure 3.2 shows that some inquiries may not be as clean and structured as one presented in Figure 3.1 since it does not directly specify which products are requested. Even though the above-specified RFQ email is of relatively good quality, many inquiries lack essential product parameters, are written in a foreign language, do not contain specific information required to finish the quotation process and require a quotation assistant to converse with potential customers to resolve the ambiguities. This can potentially inflate the size of the RFQ and the probability of human error since the quotation assistant has to navigate lengthy conversations.

In terms of ELKOV RFQs, we want to note that their characteristic is similar to what can be seen in GEOMAT RFQs, with the main difference being that ELKOV RFQs are focused on electrotechnical materials and GEOMAT RFQs are focused on geotechnical materials. Therefore, we will focus more on the GEOMATs data since their

3. IMPLEMENTATION DESIGN AND METHODOLOGY

solution requires more precise data extraction based on the need to categorize and correctly determine the parameters of products in their RFQs.

Table 3.1: Most frequently inquired parameters for different GEOMAT product categories. Category abbreviations: TEX = Geotextiles, GRI = Geogrids, REI = Asphalt reinforcements, MAT = Erosion blankets, NET = Erosion networks, CEL = Geocells, WAT = Waterproofing, PEG = Pegs, COM = Geocomposites, PAV = Grasspavings

Parameter	TEX	GRI	REI	MAT	NET	CEL	WAT	PEG	COM	PAV
Density (g/m ²)	X	X		X	X		X			
Material	X	X	X	X	X			X		X
Function	X									
Tensile Strength (kN/m, kN)	X	X	X	X	X					
Radial Stiffness		X								
Burst Resistance CBR (kN)	X									
Dynamic Burst Resistance (mm)	X									
Opening Size (mm)									X	
Elongation (%)	X									
Throughput									X	
Width							X			
Length							X	X		
Category (S1, S2)	X									
Thickness (mm)		X					X		X	
Self-Adhesive			X							
Height						X	X			X
Cells count (pcs/m ²)						X				
Type (Single/Double-sided)									X	
Load Capacity										X

What has made GEOMAT's RFQ data processing pipeline more demanding than simple product extraction from RFQ data is that it was necessary to categorize all inquired products into multiple categories that GEOMAT sells. This decision was made because each product category has different parameters. Therefore, the subset of twelve categories, seen in the Table 3.1, was created with their required parameters, which should represent the most commonly inquired products.

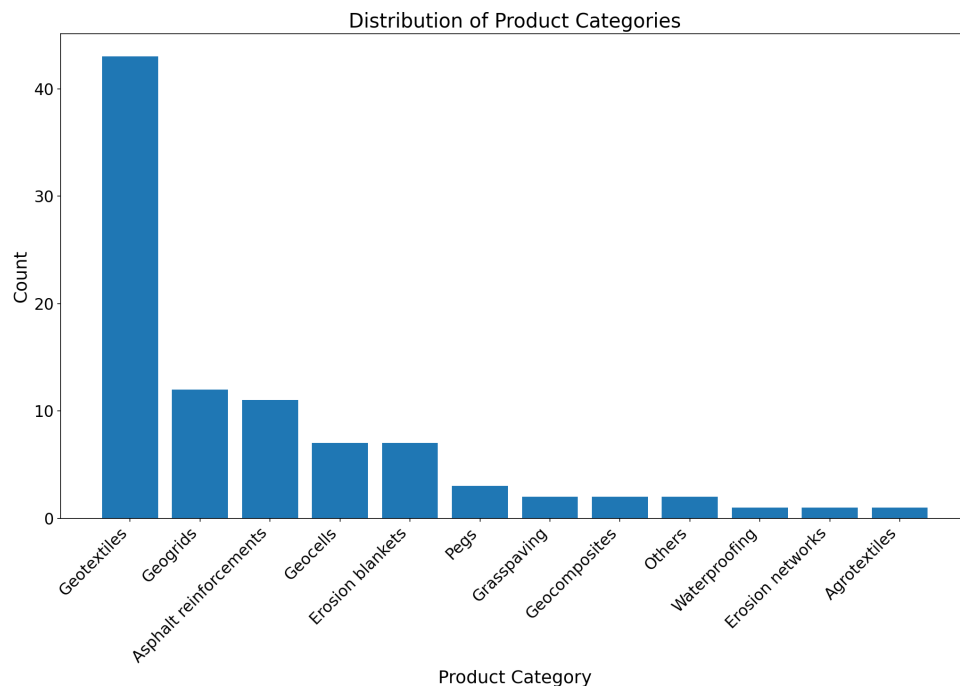


Figure 3.3: Distribution of absolute product count per category

As shown from the Figure 3.3, product category distribution is uneven, with the most prominent categories being geotextiles, geogrids, and asphalt reinforcements (a special case of geogrids). This data is heavily skewed, and the total count for geotextiles is 43 products, while the second-highest product category (geogrids) has a product count of just 12. This difference makes the geotextile products appear approximately 3.5 times as likely as geogrids. It is important to note that the data for this figure is a limited sample provided by GEOMAT and

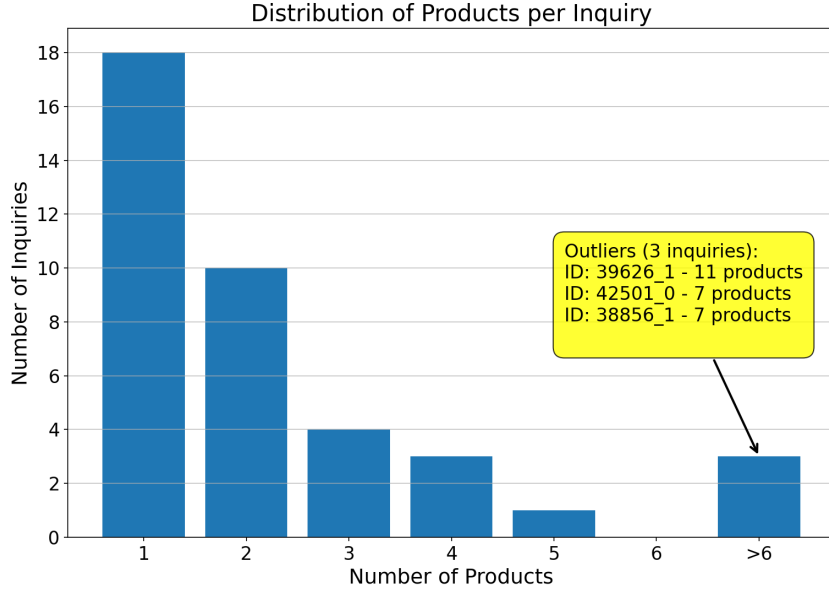


Figure 3.4: Distribution of products per RFQ

annotated by us. It comprises 39 RFQs with 92 products, and we will use it to describe GEOMAT data in the following figures. The file can be found in thesis attachments by the name `combined_samples.json` in `finetuning/Datasets`. (General overview of thesis archive is in Appendix A on page 63.)

In terms of parameters, the unevenness can also be observed in the Table 3.1, with density, material, and tensile strength being the most prominent. On the other side, parameters like load capacity, type, or throughput were almost nonexistent in the data we worked with during implementation.

Another important observation about GEOMAT data can be seen in Figure 3.4, and that is that most RFQs in the dataset we have been working with had one or two products, and larger RFQs appeared rarely (this contrasts with data we have received from ELKOV, which usually had more than four products).

Figure 3.5 highlights the difference between product categories that appear in RFQs and the number of unique categories in those

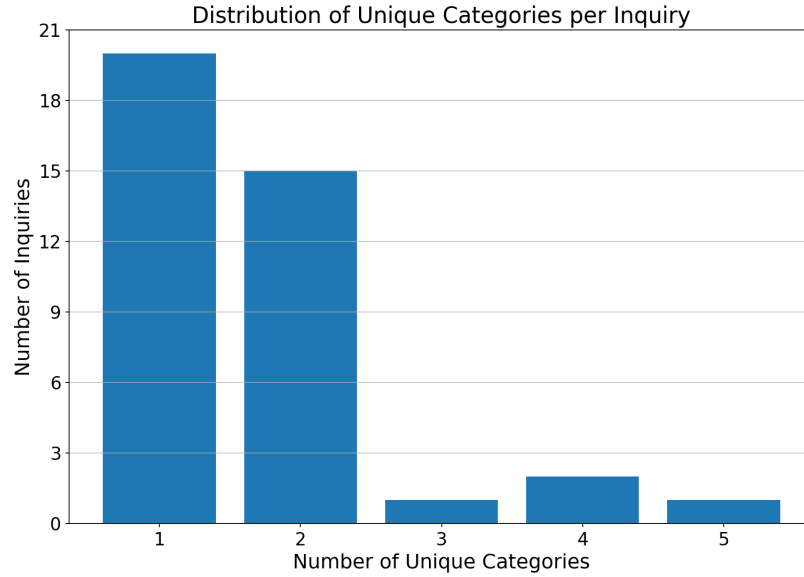


Figure 3.5: Distribution of unique product categories per RFQ

RFQs. In the previous Figure 3.4, we can see that there had been some RFQs that had more than one or two products, but Figure 3.5 shows us that even though they had more products, they were most likely of the same category. This further confirms skewness towards the most frequent categories, and we estimate a lack of variance between RFQs.

Lastly, the Figure 3.6 displays several co-occurrences of pairs of categories. Based on the figure, we can tell that geotextiles most commonly appear together with geogrids, asphalt reinforcements, and geocells. It is important to note that category combinations that appeared together only once were omitted from the graph. However, we can still see that the most frequently appearing categories co-appear. This only highlights the under-utilization of other product categories and confirms the above-mentioned biases. On one hand, this is beneficial since it will be easier to create an LLM prompt that will cover most of the products, but it will be harder to tune it for the uncommon product categories.

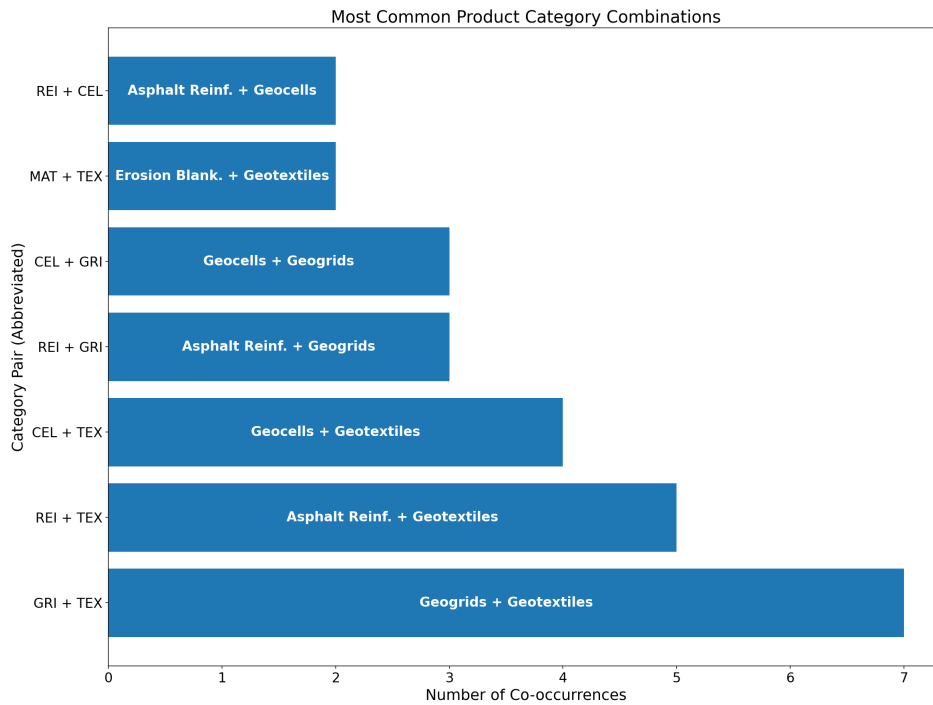


Figure 3.6: Most commonly appearing duos of product categories in RFQs

3.2 Application Flow

Now that we have described available data from GEOMAT, we can further discuss the theoretical implementation of this system, for which we have prepared the following system diagram in Figure 3.7. Based on the agreement with GEOMAT, we have proposed a system that will be available as an API. It will ingest RFQ data and provide a structured JSON output of inquired products, which will be used in their enterprise resource planning (ERP) software for easier RFQ quotation. Regarding technical details, the Python backend uses the Flask and Waitress packages for its API, the extraction endpoint is protected by an API key to prevent unwanted requests, and the application is containerized by Docker for easy deployment.

The interesting part of the system can be seen in Figure 3.7 as a red box called „Automated RFQ core“, which takes the core of all the

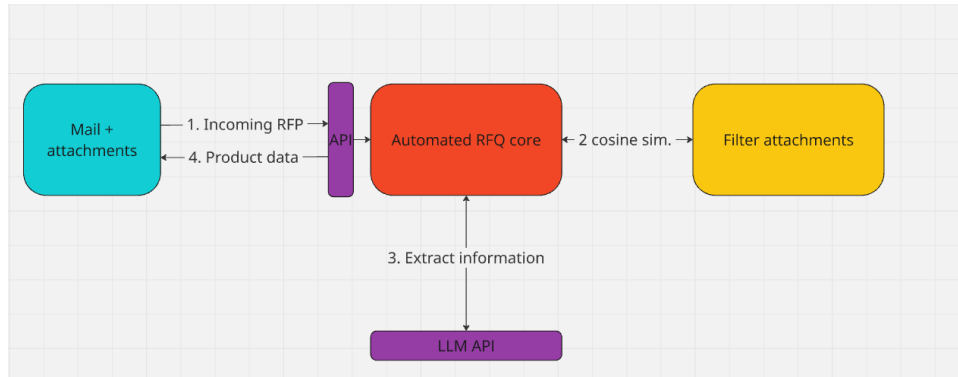


Figure 3.7: GEOMAT system diagram describing application flow

logic and processing of the RFQs. The first step of the process is to parse input data. The email is taken as is, but the PDF attachments must be loaded, and text must be extracted. PDF text extraction is achieved using the PDFPlumber package, which performs pure text extraction without Optical Character Recognition (OCR). If this fails and no text is retrieved, our fallback is PyMuPDF, which renders PDF pages into images, and then the Tesseract OCR engine performs OCR on the PDFs. This dual method ensures that text will be extracted from the PDFs since we have encountered cases where simple text extraction was unsatisfactory.

After the text has been successfully extracted, one necessary step must be performed before product extraction. In the application flow Figure 3.7, it is represented as a yellow box called „Filter attachments“, which performs document filtering based on cosine similarity [10]. As mentioned in the introduction chapter, the attachments might contain unnecessary documents in the RFQ process. Processing them would not only bloat the requests to our LLM provider and ensure higher processing costs but most likely decrease the accuracy since the LLM would have to process an unnecessarily large amount of data.

Firstly, to be able to perform the cosine similarity filtering, we must first process the documents into vector form. Since we are utilizing the Langchain ecosystem for this application, we have decided to process documents with `RecursiveCharacterTextSplitter` that will split all the documents into chunks of length 1024 characters with an overlap of 128 characters between chunks so that information loss based on

splitting is minimized. Then, the document chunks are processed into OpenAI embeddings with model `text-embedding-ada-002`¹, which has been chosen for its ease of use, since we will also be using OpenAI's GPT-4o model for product extraction. The last thing to note regarding document chunk embeddings is that they are stored in the in-memory vector database ChromaDB until the attachments are processed.

When all of the chunks have been embedded, the cosine similarity for each chunk will be computed with a predefined list of keywords that have been chosen based on experiments on the provided dataset. This relevance score will then be used to select the most relevant document based on the score itself and a reward for shorter documents. The reward slightly increases the score of shorter documents since irrelevant documents, like technical documentation, tend to be larger files. Therefore, only the most relevant document will be used for further processing, and other attachments will be disregarded.

3.3 Extraction Process

Now that the most relevant document has been chosen, we can discuss the product extraction. This process consists of utilizing the GPT-4o model over the API with our comprehensive system and human prompts, which have been iteratively improved on the provided dataset. The naive base solution initially targeted only the three most popular categories, which are mentioned in Figure 3.3, accounted only for product name, category, and quantity, and only a few parameters, and the solution could not process PDFs at this time. The naive solution's prompts were compact, at only 15 lines for both the system and human prompts. The latest production-ready system prompt spans 171 lines, which is more than 1000 words, and the human prompt is 60 lines long, around 450 words.

Listing 3.1: System Prompt: General response requirements

```
**Response Requirements:**  
- The response must be a valid JSON object without  
  any markdown formatting ('''json).
```

1. `text-embedding-ada-002` model description available at: <https://platform.openai.com/docs/models/text-embedding-ada-002>

- All numeric values must be provided as numbers without thousands or millions separators (e.g., "1,191.905" should be "1191.905")
 - Decimal numbers should use a single dot as decimal separator (e.g., "1.5" for one and a half)
 - Organize offered products into the following groups, each containing at least 'name', 'quantity', and 'quantityUnit' properties:
- [...]

Given its importance in the extraction process, we will describe selected noteworthy parts from both prompts, starting with the system prompt in Listing 3.1. This part defines our general response requirements. We are declaring that the LLM should return a response as a valid JSON object without any formatting, specifying numeric separators since most of the inquiries will use Czech-style numeric separators, and defining that each product must contain at least the name, its quantity, and the amount unit. We are also requesting that numbers not use separators to ease parsing. The part omitted from this example defined that products should be grouped because the same schema was presented in Table 3.1.

Listing 3.2: System Prompt: Categories are requested in specific orders

```

**Group Mapping (Priority Order):**
1. **asphaltreinforcements**
   - "Vyztuhy do asfaltu / bitumowana" → "
     asphaltreinforcements"
   - "skelne vlakno", "fiberglass" → "
     asphaltreinforcements"
2. **geotextiles**
   - "Geotextilie / Geowloknina / Geotkanina" → "
     geotextiles"
[...]
```

The following Listing 3.2 describes two essential things. First, it organizes the priority of the product categories, which, in our testing, appears to affect category assignment. Category prioritization was necessary, especially for the „asphalt reinforcement“ and „geogrids“ categories, which the LLM often confused, and on „erosion blankets“ and „erosion networks“ categories, which faced the same problem.

The next vital thing was the introduction of keywords, which were chosen based on our tests on the GEOMAT's dataset. This choice has resulted in better categorization success overall. This list was further refined in Listing 3.3, where each category has many defining keywords in multiple languages. For example, we can see that „erosion networks“ were defined in Czech, English, and German. Other categories, such as „asphalt reinforcements“, required more specific keywords frequently appearing in the product data, like „fiberglass“, their primary used material.

Listing 3.3: System Prompt: Keywords to match categories

```
**Group Mapping (Priority Order):**
- **asphaltreinforcements:**
  - Keywords: "asfaltu", "vyztuzeni asfaltoveho
    povrchu", "skelne vlakno", "fiberglass",
    "VYZTUZNE VRSTVY", "VOZOVKA", "sklo",
    "vyztuzena geotextilie", "bazi skelných
    vlaken", "bitumowana"
[...]
```

```
- **erosionnetworks:**
  - Keywords: "protierozni site", "erosion nets", "
    networks", "geosit", "netz", "geomanet"
[...]
```

We have also developed a list of additional instructions the LLM should follow, displayed in the Listing 3.4. The main reason was to emphasize a few instructions that LLM overlooked. Still, we have introduced quantity preference since RFQs are email conversations, and in case the quantity amount was offered from the GEOMAT side, we want to preserve this information. An essential instruction is „Default Values“, which tells LLM to set unknown expected parameters to -1 . Therefore, the „Default Values“ instruction helps to reduce a common problem with LLMs, which is hallucinations, since if the information is not present in the RFQ, the LLM will be less likely to try to make it up.

Listing 3.4: System Prompt: Additional instructions to capture separators, quantity preference, boolean representation, and default values

```
**Additional Instructions:**
- [...]
```

```
- Quantity Preference: If both requested ('
  Poptavane mnozstvi') and offered ('Nabizene
  mnozstvi') amounts are available, prioritize the
  requested amount.
- Default Values: Do not assign default values
  to parameters. If uncertain, set them to '-1'.
- [...]
```

Our last example of the system prompts in Listing 3.5 shows how we use few-shot prompting to incorporate in-context learning to guide the model for better results. We have chosen multiple examples that LLM had problems evaluating correctly in the development, and provided their corrected version in the prompt. The positive examples include text extracted from the RFQ with the correct product JSON object. We also included one negative example of quantity multiplication because we encountered an issue where LLM was using other parameters for this multiplication (i.e., when three packages of fifty pegs are requested, the LLM should perform multiplication).

Listing 3.5: System Prompt: Examples as guidance

```
- Multiples Example:
  - Original Text:
    "Agrotextilie EK0+ - 157 g/m2 2*50 m
    [100 m2]: 1 role"
  - Extracted Product:
    '''json
    {{
      "name": "Agrotextilie EK0+ - 157 g/m2",
      "quantity": 100,
      "quantityUnit": "m2"
    }}
    '''
- [...]
- Incorrect Multiplication Example:
  - Original Text:
    "900 m2 geotextile with width of 4 meters"
  - Incorrect Extraction:
    'quantity: 3600' (900 * 4)
  - Correct Extraction:
    'quantity: 900', 'width: 4', 'widthUnit: "m"'
```

Regarding the prompt template, we primarily use it to define parameters required for product categories and relevant parameter-specific instructions. These parameter-specific instructions are described in the same manner as additional instructions in Listing 3.4. The only noteworthy fact is that we support range values for specific parameters as shown in Listing 3.6. Therefore, we will omit longer showcases as we have done for system prompts to preserve brevity.

Listing 3.6: Template Prompt: Support for range values of certain parameters

```
[...]  
7. **Ranges:** Represent ranges as lists (e.g.,  
"10-20mm" becomes '[10, 20]',  
"300 to 500 g/m2" → '[300, 500]').  
[...]
```

The last part of the product extraction process is the post-processing of the LLM response. First, we check for duplicate products that differ only in quantity and merge them into a single entity. The next step is a material consistency check. For example, materials like „polypropylene“ may be retrieved in different languages based on the inquiry language, so we replace these full names with corresponding abbreviations. (i.e., PP for the polypropylene) We also have to verify product fields containing the unit type of „m2“ since the LLM tends to replace the number two with a square symbol. Finally, we translate all the product object’s keys into Czech as agreed upon with GEOMAT. This concludes the implementation part of GEOMAT partial RFQ automation, and we will return to the results evaluation in the next chapter.

3.4 Design of Model Fine-Tuning Experiment

The next goal of this work was to compare the performance of available open-source LLMs to commercial GPT-4o, which we have used for the GEOMAT solution. Based on the reviewed work in Chapter 2, we have decided on a relatively smaller LLM LLAMA 3.1 with eight billion parameters (we will refer to this model with this size just as LLAMA 3.1 since there are multiple sizes for 3.1 version), to fine-tune it to our product extraction scenario. To better extend this comparison, we

will also be using GPT-4o-mini, which should be a more adequate comparison to LLAMA 3.1. To summarize this, we will compare the performance of GPT-4o, GPT-4o-mini, and LLAMA 3.1 8B in its base and fine-tuned versions on two datasets, which we will disclose in Chapter 4.

As we have already mentioned in Chapter 1, there is a lack of annotated datasets we could use for fine-tuning, so we have decided to create a synthetic dataset. As a starting point, we will use the GEOMATs dataset discussed in this chapter, which will serve as a baseline for our synthetic dataset.

Our first approach in this generation was to utilize GPT-4o for complete sample generation, i.e., the RFQ email containing the products and the ground-truth extracted products were generated by the LLM in the JSON format. We have chosen this approach because we wanted to utilize the creativity of generative AI models and avoid the generation of a dull dataset based on predefined factors. The good thing about this approach was that GPT-4o generated convincing RFQs, which initially seemed promising. However, this approach has failed, since the GPT-4o failed to create a consistent product list and the ground-truth data. We have tried to utilize this generation and fix the inconsistencies by post-processing the results. However, we have decided that ensuring the correctness of our fine-tuning data was a bigger priority than the creativity of the RFQs. It is important to note that this has also incurred costs for data generation, which are negligible for a single sample but should be considered when generating hundreds or more samples.

Therefore, we have settled on a generation with fixed variability based on the existing product pool from the GEOMATs dataset. The process of generating a synthetic dataset can be described as follows:

- User must specify the number of samples they want to generate.
- Product pool is constructed based on the provided dataset, and available categories are counted.
- The Target number of products for the sample is generated in the range of $\langle 4, 12 \rangle$, which should ensure that the generated dataset has, on average, more products than the GEOMATs dataset. This is done because we want to improve the model's ability

to correctly extract products from RFQs with higher product counts.

- Product sampling is done based on weighted sampling. Rare categories (those having fewer than three products) have modified weights based on how many rare products are available without repetition (if products were used, the weights are very low).
- Sampling then begins from rare categories. Hence, they get appropriate representation, and the rest of the products are then sampled from the more frequent categories, with the maximal rare categories being present calculated as:

$$\min \left(\text{len}(\text{rare_categories}), \left\lfloor \frac{\text{number_of_products}}{3} \right\rfloor \right).$$

- Each product is then augmented based on the permissible modifications for the given product category, with a 70 % chance of adding a parameter from this list. The product quantity is also varied with a maximal change of 50 % from the original value in both directions. There is also detection for implicit parameters since some products have material incorporated in their name.
- A simple validation check is then performed to verify that all numeric values generated for the synthetic product appear in its name and, therefore, in the product list of the RFQ.

Regarding the variability of this dataset and the conversational nature of the RFQ inquiries, we have also prepared a modifiable configuration, which manages not only the parameters variable in the generation process but also the template itself. Therefore, the project, company, and sender names are also variables, giving the RFQs a more natural feel. We have generated three synthetic datasets for this experiment: a_100, a_256, and a_1024, with numbers representing the number of RFQ samples present in a particular dataset. The smallest one will be used for training verification after fine-tuning, since it is just the first hundred samples from the other two datasets. The a_256 dataset was our initial dataset, which we wanted to use for the fine-tuning. We wanted to keep the dataset relatively small since we had a relatively small pool of products and did not want to overfit the

model. However, the `a_256` has proven insufficiently small, and therefore it was used just for training and debugging. The training was then performed on the largest dataset of 1024 samples. It is important to note that since we are using fixed seed for the generation, the smaller datasets are subsets of the bigger ones, i.e., $a_{100} \subset a_{256} \subset a_{1024}$; therefore, the bigger datasets are just extensions of the smaller ones.

To get the most performance out of the training, we have chosen to use LoRa, which means that the original model weights were frozen during the training, and instead, the LoRa low-rank adapters were trained. We have chosen the rank of these matrices to be `lora_rank` = 16, the alpha factor of `lora_alpha` = 16, which in this configuration results in a scaling coefficient of 1.0. Therefore, this adapter directly adds the modified weights to the model weights. We have made this choice to better manage the required memory and the effectiveness of the fine-tuning. In terms of future work, we would recommend experimenting with increasing the `lora_rank` and scaling factor to see how it would affect the results. Increasing the rank of LoRa adapters and the scaling factor could positively impact the results since it would force the model to pay more attention to LoRa weights. We have also set `lora_dropout` = 0.05, which results in dropping 5 % of the training input to act as regularization and prevent overfitting. It is important to note that `lora_target_linear` was set to True, so the low-rank matrices were applied to all standard linear layers.

Since we had access to the setup of two NVIDIA L40 GPUs, which totaled 96 GB VRAM, we have decided not to use quantization to maintain the maximum model performance. Based on the available VRAM, we have decided to go for 2048 tokens as the model context length. This context length should be appropriate considering our dataset. When tokenized, the input RFQs are in the lower hundreds of tokens, and ground truth data tends to be in the 500 to 1000 tokens range. However, we have encountered RFQs where ground truth data exceeded 1024 tokens, and therefore, we have opted for the next power of two as the context length.

For the LoRa precision, we have used the brain float 16 (bf16) type, which should reduce memory and computation while helping training stability. When comparing bf16 to classical fp16, the main advantages of bf16 are that it preserves a larger exponent range, making it less prone to underflow/overflow, and requires no loss scaling.

We have chosen the AdamW optimizer, which should prevent the over-regularization issue of the basic Adam optimizer by applying weight decay directly to weights and not gradients. We have also combined this with a warm-up ratio of 0.1, so for the first 10 % of the training steps, the learning rate (LR) will linearly increase from 0 to our maximum of $2e - 05$, which should help stabilize the training. Then, we apply a cosine LR scheduler so that LR will slowly decay following a cosine curve to zero, preventing overshooting the minima. For the batch sizes, we have used `micro_batch_size = 4` and `gradient_accumulation_steps = 4`, resulting in an effective batch size of 16.

In terms of other training optimizations, we have also utilized flash attention [12], which is a more memory-efficient and faster implementation of classical attention, leveraging fused kernels for improved performance. Additionally, we have used fully sharded data parallelism (FSDP [13]), a multi-GPU training technique to manage GPU memory efficiently since it shards model parameters, gradients, and optimizer states across all available GPUs. We have opted for the FSDP over simpler distributed data parallelism (DDP) since DDP replicates the whole model on each GPU and only splits training data, which could hinder our approach to not using model quantization. The Axolotl library provided all of the mentioned optimizations, which helped us achieve efficient multi-GPU training.

As for the general training setup, we have used a cross-validation setup with five folds and three epochs. We have chosen this fold setup because, in the initial debugging training runs, we noticed that each fold’s performance was not dramatically different. Therefore, it would be unreasonable to go for more folds, lower the training data for each fold, and train twice as many models. Currently, on the above-mentioned hardware, the training of one fold took a bit more than one hour, and this was just fine-tuning, meaning that inference on our testing datasets is not included. In future experiments, we consider training even fewer folds or rather focusing on training one or two models for more epochs.

The chosen fold split resulted in 820 samples used for the training and 204 samples for validation. The training also utilized the system prompt we presented in previous chapters to capture our task better. We used causal language modeling, which is standard for text

generation. Therefore, the model was trained using the cross-entropy loss function, which measures the divergence between the predicted probability distribution and the ground truth token at each position in the sequence.

Lastly, the model was checkpointed every six training steps from 75 steps, and the best model was loaded and stored at the end. This was helpful for the model evaluation, together with the weights and biases platform, which we have used for model debugging. However, the model results and particular evaluation methods will be discussed in the next chapter.

3.5 Comprehensive RFQ Automation

After delivering the GEOMAT solution, we wondered what the most reasonable upgrade we could do for the RFQ process automation. After delving more into the sales side of the process, we realized that we could speed up the part that matches the products to the company's product catalog. This idea comes from the fact that after the quotation assistant examines the incoming RFQ, he must verify the products that his company has available. Many companies today use ERP software for resource management, meaning their catalog is digitized, and we have already managed to extract products from the incoming RFQs. Therefore, the only interconnecting piece was searching the catalog for desired products. For this purpose, we have decided to create an application called Offermate, and we will thoroughly describe it now.

The application can be split into two main parts: the backend and the frontend. First, we will describe the backend, giving insight into how the application processes the client's catalog, maintains original product extraction, and utilizes information retrieval techniques to perform searching. The frontend part will describe how the quotation assistant can utilize this application to perform the RFQ process efficiently.

3.5.1 Storing the Product Data

The first important part of this system would be storing and ingesting the client's catalog. When developing the application, we discussed

potential requirements with multiple clients. When we were asking about their product databases, we learned that they consist of many catalogs based on data from multiple distributors. The length of the total database can range for some clients from a hundred thousand products up to a million or two products. Therefore, we had to prepare a solution that could be scalable and could perform reliably on such data. Initially, we considered using ChromaDB, which we had tested for the GEOMATs solution, but later, we opted for the industry standard PostgreSQL (PSQL) database. We knew from the start that we would have to work with vector product representation, but this presents no issue in the PSQL, even though it's not marketed as a vector database. There is a clever solution to using vectors in PSQL: the pgvector extension, which supports not only storing vectors but also indexing and similarity search. Therefore, we have created the following schema for the product table.

`product_code` VARCHAR(255), PRIMARY KEY. Products use a unique string identifier, which is sometimes present in the RFQs. (Especially if a customer is repeating their past order)

`description` TEXT, NOT NULL. A textual description of the product, which helps capture the product's purpose and features.

`sales_text` TEXT, NOT NULL. Sales text descriptions are similar to product descriptions and often extend them.

`list_price` DECIMAL(15, 2), NOT NULL. The suggested price for the product.

`metadata` JSONB. A flexible field for storing all other product-related data that is currently unused but may be helpful in the future. One example is a product EAN code, but other fields like ecological fees, notes, and suppliers are stored here.

`sales_text_embedding` vector(384). A 384-dimensional numerical vector representing the semantic content of the `sales_text` and `description`. We have decided to also use `description` field for the embedding since both of the fields describe the product well, and sometimes `description` contains information not present in

`sales_text`. The `pgvector` extension provides this type, and we will discuss its performance-related enhancements shortly.

`created_at` `TIMESTAMP WITH TIME ZONE`. An automatically generated timestamp marking the product record's date and time it was first imported.

`updated_at` `TIMESTAMP WITH TIME ZONE`. An automatically generated timestamp that updates to reflect the date and time of the last modification to the product record.

This way, we are building a future-proof solution that maintains even less relevant data, which can be used whenever required. In terms of classical optimizations, we are using database indexes on `product_code`, `description`, and `sales_text` since we are assuming that those columns will be relevant for look-up queries. The `embeddings` column also utilizes a special type of index called Inverted File with Flat Compression (IVFFlat)². This index creates regions of similar vectors called clusters by applying the well-known k-means algorithm on embedded vectors. Then, an inverted index is created to map corresponding cluster centroids to their underlying vectors. Later, when the database is queried, the nearest clusters are identified to the query, and other clusters are disregarded; therefore, the search space is pruned. In our implementation, we have decided to let the IVFFlat index create 100 clusters for our data. At the time, we did not know the precise number of records for the catalog, but the generally advised formula for a table with fewer than one million rows is $\text{clusters} = \text{number_of_products} / 1000$. The last important thing to note regarding embeddings is that before any RFQ processing can happen, the whole product catalog must be parsed, and the embedding must be precomputed and stored. (The embedding process itself takes several hours when utilizing only the CPU. However, there is research already on how this approach can be scaled efficiently [14].)

The next important part in designing our backend was model choices. Since we have had good experience using OpenAI models on the GEOMATs solution, we have decided to keep it as the LLM for product extraction. Other interesting choices were models for the

2. What Are IVFFlat Indexes in Pgvector and How Do They Work; Available here

embeddings. We wanted to approach this systematically, so we have picked our models based on the Massive Text Embedding Benchmark [15], available in the interactive form at Huggingface. Our first choice for the embedding model was the widely used all-MiniLM-L6-v2³ model, which maintains reasonable performance at its relatively small size of 22 million parameters. It is important to note that this task was constrained because we had to encode and run the inference on the CPU. However, we were unsatisfied with the results, so we decided to scale up the model and settled on the Multilingual-E5-small [16]. This model has multiple advantages; it is the second-best performer in its parameter range at 118 million parameters, achieving a mean task score of 55.76. Let us compare it to other bigger models. Their mean-task performances are in the lower sixties, which offers diminishing returns, especially if we want to run this relatively fast on the CPU. A natural question would be why we have not picked the best-performing model in the hundred-million category. This is because Multilingual-E5-small has seen 94 languages during its training, including Czech, Slovak, Polish, and German, which are important for us since we expect to receive RFQs in them. The best performing Lajavaness/bilingual-embedding-small⁴ has only seen English and French data, which made it a relatively easy choice to choose Multilingual-E5-small.

We have also decided to use the Reranker model in our retrieval pipeline to enhance our retrieval abilities further. While the embedding search can provide a list of relevant candidate products, it may struggle with correct prioritization. Therefore, we have decided to first retrieve 20 candidate products for each product in the RQF, then we narrow this by applying the reranker.

Given that our hardware capabilities were a limiting factor, we have decided to select the cross-encoder/ms-marco-MiniLM-L6-v2⁵, which is a popular ranking model offering reasonable performance at just 22.7 million parameters. It processes query-document pairs

3. all-MiniLM-L6-v2 model card available at: <https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2>

4. Lajavaness/bilingual-embedding-small model card available at: <https://huggingface.co/Lajavaness/bilingual-embedding-small>

5. ms-marco-MiniLM-L6-v2 model card available at: <https://huggingface.co/cross-encoder/ms-marco-MiniLM-L6-v2>

together (based on its cross-encoder architecture). It is, therefore, able to attend to both inputs, allowing for more precise product ranking than just vector-based similarity. Afterward, we keep only the top four relevant products based on the reranked preferences, which should provide a sufficient sample for the quotation assistant.

However, before we can explain our search methodology and the process flow, we shall discuss one more improvement we have made to this application. GEOMAT's solution utilized relatively simple attachment processing, which we knew would be insufficient for more generalized applications. Currently, the application supports `xlsx`, `pdf`, and `image` file types. The data extraction work for each file type is as follows.

`xlsx` Reading the Excel files is the most straightforward method from the supported file types. This approach utilizes the Pandas library and its method `.read_excel()`, which works straight „out of the box“.

`image` Reading the image attachments is finally supported and utilizes more advanced OCR technology than the one we had in the GEOMAT solution. The PaddleOCR package provides the OCR model. Specifically, we are using the `SVTR_LCNet` [17] model, which is a lightweight text recognition network based on the SVTR (transformer-based architecture) with a convolutional neural network backbone. Even though we have not concluded thorough testing, we have achieved better results on our data at a glance than with the EasyOCR or pytesseract packages. To further enhance the text extraction and resolve the grammar and character issues, we send the extracted text for validation to GPT-4o, which completely resolves these issues.

`pdf` Reading the PDF files has been reworked from GEOMAT's solution. The process still relies on the combination of PyMuPDF for simple text extraction but utilizes our new advanced OCR approach. Therefore, if simple text extraction fails, we utilize the same extraction performed on the images for the affected PDF page, which yields better results overall.

3.5.2 Searching Methodology

To explain our search methodology and the whole process flow, we have prepared a diagram similar to the one we used for GEOMAT.

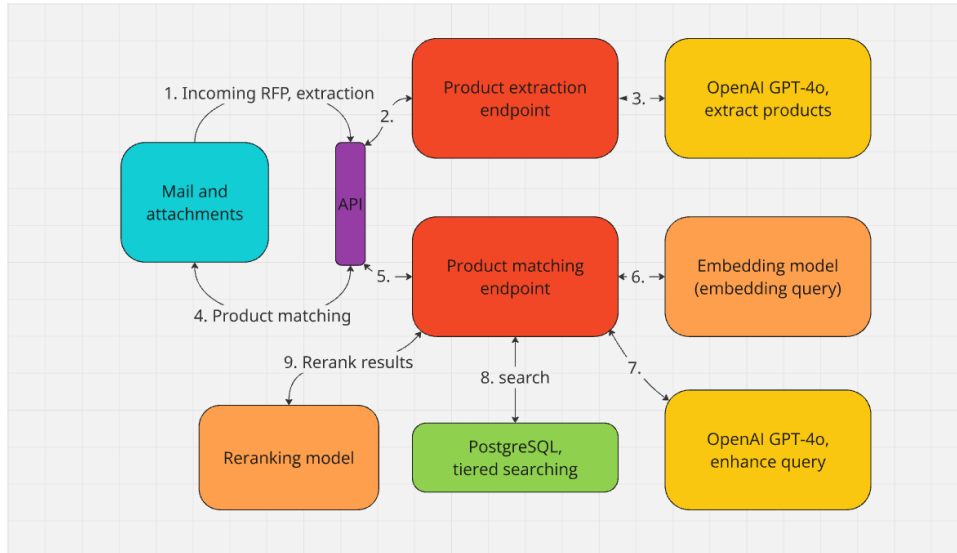


Figure 3.8: The Offermate system diagram describing applications flow, split into product extraction and product matching endpoints.

As shown in Figure 3.8, the RFQ data is sent to the API in the same fashion as presented in Figure 3.7. However, the process is then divided into two parts: first, product extraction, and second, product matching. Even though the endpoints are separated, they are called sequentially. The reason is not to let the quotation assistant idly wait but to provide him with data as soon as possible. The product extraction process begins with our new data ingestion and is then performed in the same way as in the GEOMATs solution. The main difference is that this application does not require parsing such constrained data as GEOMATs, and therefore, product extraction can be achieved with a simpler prompt. When comparing the extraction prompts, the first big difference is that the prompt template contains just the ingested RFQ text, and no further instructions are specified. The system's prompt totals 880 words, a further reduction than GEOMAT's prompt, which has over a thousand words. Probably its most significant difference

is that this system's prompt focuses heavily on correct product code extraction. The mentioned product codes can uniquely identify the product and drastically speed up the matching process. Another important thing is product specification, where this prompt does not care about categorizing found product and its parameters but instead focuses on gathering as much information as possible, which can be used for the matching process. Lastly, the output schema definitions, unit descriptions, and usage of positive and negative product extraction examples remain.

After the product extraction is complete, the product matching process can begin. For this, we introduce tiered search, which tries to map extracted products by trying multiple approaches to find the right match. The first method is straightforward product code matching, where we verify that the extracted product code exists in the database. This is a simple SQL query, and as we mentioned, we are using an index, which makes this check fast.

If we cannot find an exact match by the product code, the `ILIKE` query is executed on the product code column, where we check if the found product code matches any substring in the database. The purpose of this query is that it may happen that the customer wrongly copies the product code and omits some parts of it, or the extraction process may extract an incomplete product code. Even though this query is slower than exactly matching the product code due to indexing, it is still relatively fast.

In the next part, we tried to utilize the metadata we collected, so we added one more matching attempt. If the product extraction process finds the product EAN code, the application will try to match it to the EAN codes in the metadata object. Based on the RFQ data we have worked with from ELKOV, we did not see EAN codes appearing often, but if they are present in the extracted products, it is yet another easy way to match products precisely. There is a performance concern here since metadata is not indexed, but if we see high utilization of this type of search based on production data, it is rather a quick issue to resolve.

The last and most important matching attempt is semantic matching. This matching type only executes if a product cannot be precisely matched based on previous methods. The first step of this method is to take the product specification, which has been extracted and is

represented by a JSON object, and create a query text from it. In the initial version, we extracted textual descriptions and concatenated them together, but we achieved better results with the query enhancement process. This process involves calling GPT-4o on the extracted products to construct one unified search query. The main advantage of this approach is that GPT-4o can understand the information in the object and fill in missing details necessary for finding the match. Inspiration for this approach was a query enhancement technique that can be seen in retrieval-augmented systems, where LLM reformulates the input query into multiple queries before the search is executed.

After the enhanced query has been constructed, it is embedded into its vector representation by the `multilingual-e5-small`. The cosine similarity is then computed for this query vector and product embeddings in native SQL by the `Pgvector` extension, and the twenty most relevant products are then retrieved. These products are then passed to the reranker model with the query, and only the top four relevant products are chosen. This process is applied to all extracted products. It is the most time-consuming process compared to previous methods, which is logical since cosine similarity must be computed for many vectors, and Reranking is rather a slow process when utilizing the CPU.

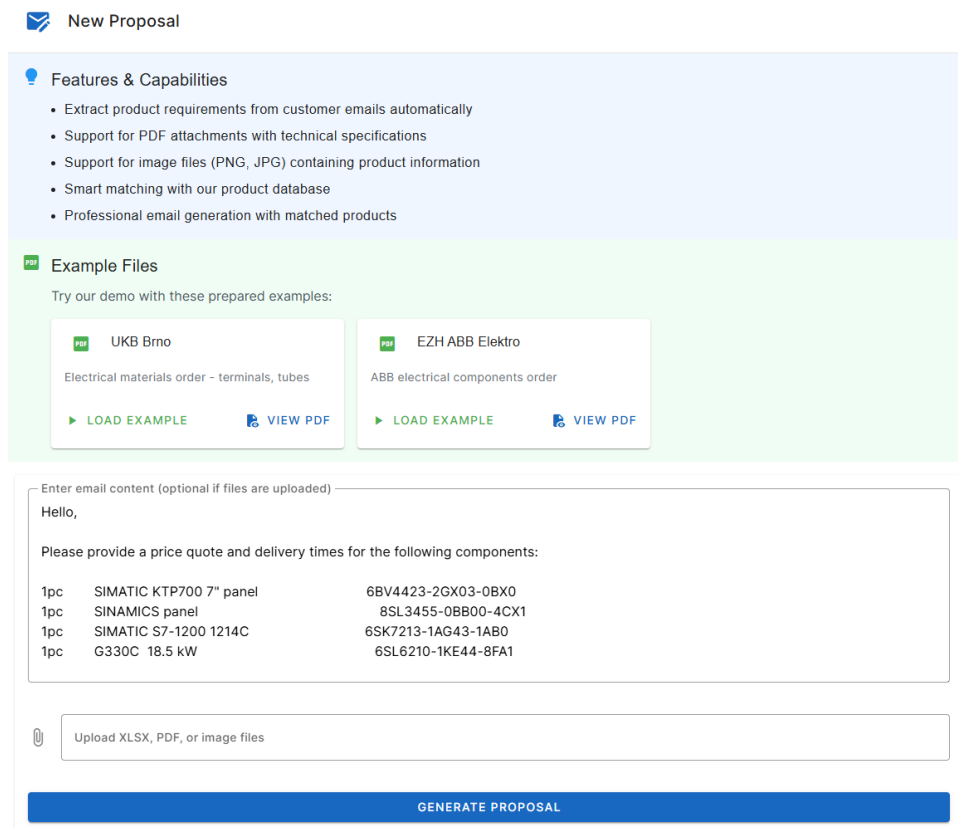
The last alternative matching process is a tri-gram similarity search on the description field, which utilizes the enhanced query. This approach has proven to retrieve relevant results when thresholded at 0.3 similarity. Its primary purpose is to add variety to the search results. Since the requirements for similarity are relatively high for this method, it only sporadically extends the results provided by similarity search. Afterward, the results are collected and returned as JSON for the frontend application, which concludes the matching process.

3.5.3 Offermate's Frontend

The frontend is a JavaScript single-page application (SPA) that serves as an interface for the quotation assistant. Specifically, the Vue.js framework is used since it enables features such as two-way data binding, SPA routing, and componentization, which enables faster development. Standard packages such as Axios for Ajax requests and Pinia for application state management are used.

3. IMPLEMENTATION DESIGN AND METHODOLOGY

Regarding CSS libraries, Tailwind is used for quick styling, along with Vuetify, a UI components library. The frontend consists of two main pages: database management and quotation generator. Database management lets users upload a product catalog, review, and search already ingested products. This page is rather simple, so we will focus more on the generation of quotations.



The image displays a web interface for generating a proposal. At the top, there is a 'New Proposal' button. Below it, a section titled 'Features & Capabilities' lists several features: extracting product requirements from customer emails, supporting PDF attachments with technical specifications, supporting image files (PNG, JPG) containing product information, smart matching with a product database, and professional email generation with matched products. This is followed by an 'Example Files' section, which shows two example files: 'UKB Brno' (Electrical materials order - terminals, tubes) and 'EZH ABB Elektro' (ABB electrical components order). Each example file has a 'LOAD EXAMPLE' button and a 'VIEW PDF' button. Below the example files, there is a form for entering email content. The form includes a text area with the following content: 'Hello, Please provide a price quote and delivery times for the following components: 1pc SIMATIC KTP700 7" panel 6BV4423-2GX03-0BX0 1pc SINAMICS panel 8SL3455-0BB00-4CX1 1pc SIMATIC S7-1200 1214C 6SK7213-1AG43-1AB0 1pc G330C 18.5 kW 6SL6210-1KE44-8FA1'. Below the text area, there is a file upload button labeled 'Upload XLSX, PDF, or image files'. At the bottom of the form, there is a large blue button labeled 'GENERATE PROPOSAL'.

New Proposal

Features & Capabilities

- Extract product requirements from customer emails automatically
- Support for PDF attachments with technical specifications
- Support for image files (PNG, JPG) containing product information
- Smart matching with our product database
- Professional email generation with matched products

Example Files

Try our demo with these prepared examples:

UKB Brno

Electrical materials order - terminals, tubes

[LOAD EXAMPLE](#) [VIEW PDF](#)

EZH ABB Elektro

ABB electrical components order

[LOAD EXAMPLE](#) [VIEW PDF](#)

Enter email content (optional if files are uploaded)

Hello,

Please provide a price quote and delivery times for the following components:

1pc	SIMATIC KTP700 7" panel	6BV4423-2GX03-0BX0
1pc	SINAMICS panel	8SL3455-0BB00-4CX1
1pc	SIMATIC S7-1200 1214C	6SK7213-1AG43-1AB0
1pc	G330C 18.5 kW	6SL6210-1KE44-8FA1

[Upload XLSX, PDF, or image files](#)

GENERATE PROPOSAL

Figure 3.9: UI showcase of the proposal page where the quotation assistant should upload relevant RFQ documents to start the quotation process

The product quotation generator page is the primary part of the application with which the quotation assistants will interact most of the time and is shown in Figure 3.9. To start quotation generation, the sales assistant must provide at least an email conversation or a

3. IMPLEMENTATION DESIGN AND METHODOLOGY

quotation document, which has been described in the backend part. However, the application has a prefilled email example in the input box for demonstration purposes to make this interaction more intuitive. The new user can also load two bundled demo quotations to examine the extraction and matching process on quotation files. These demo features will be turned off later since their purpose is to introduce the application and ease the onboarding, and the input box prefill would interfere with daily use.

 **Generated Proposal**
Review and select products for your proposal

EXTRACTED PRODUCTS

MATCHED PRODUCTS

 **Products from Inquiry** 4 items

Product Details	Specifications
6BV4423-2GX03-0BX0	☐ Description: SIMATIC KTP700 7" Panel ☒ Quantity: 1
8SL3455-0BB00-4CX1	☐ Description: SINAMICS Panel ☒ Quantity: 1
6SK7213-1AG43-1AB0	☐ Description: SIMATIC S7-1200 1214C ☒ Quantity: 1
6SL6210-1KE44-8FA1	☐ Description: G330C 18.5 KW ☐ Power: 18.5 KW ☒ Quantity: 1

Figure 3.10: Example of results from the product extraction phase based on the default input data as shown in Figure 3.9

After submitting the data, the user will first receive extracted products and can examine the found parameters. This is illustrated in Figure 3.10, where we have run the extraction on the prefilled email example, which we will use in the following figures to demonstrate the application flow. The important thing to note from the extraction is that captured parameters are not fixed, meaning the LLM should determine which input data represents the product parameters. It is only suggested that products should be represented by product code, description, and quantity, while other parameters remain optional.

However, we are still talking about partial RFQ automation because the application requires human interaction and verification. Therefore, the sales assistant should always verify the extracted data against the provided documentation and is still required to select which products will be quoted. After the matching process, the user can switch tabs and review the matched products.

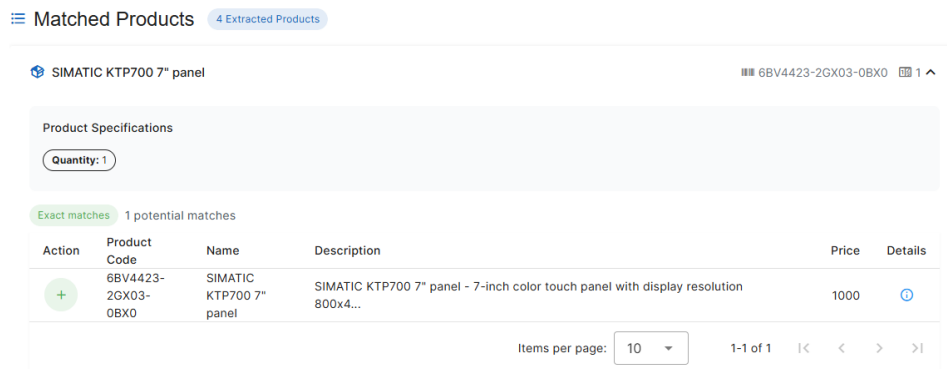


Figure 3.11: Example of matched product based on the extracted data, shown in the Figure 3.10

As seen in the Figure 3.11, the matches are grouped under their extracted product, and the matching method is also shown. In the presented example, the matching method is „Exact matches“ since the product code, which was present in the email, directly matched the product code in the database, and therefore only one product was retrieved. However, in terms of semantic matches, multiple products would be retrieved, and the quotation assistant would pick the most relevant one. The interface also allows the quotation assistant to easily select which products should be quoted by clicking the plus button next to the retrieved product and viewing product details for further product examination. This is particularly useful if the quotation assistant is unsure of product parameters from its name. There is also an option to manually look up products from the database if the quotation assistant knows of a more relevant product they want to add to the quote.

3. IMPLEMENTATION DESIGN AND METHODOLOGY

Selected Products 2 items

+ CCC + XXX EXPORT TO CSV

Kód	Kód d...	Množ...	Slev...	Slev...	Cena	Cen...	Název	Mj
6BV44	9394€	1			1000		SIMATIC KTP700 7" panel	KS
Poznámka								
0								
8SL34	3956€	1			1000		SINAMICS panel	KS
Poznámka								
0								

Email Generation

Email Style

☒ Predefined ☐ Custom

professional and direct

GENERATE EMAIL

Figure 3.12: Selection of multiple products for quotation, which are available either for the CSV export in ELKOV ERP format or automatic email generation

Selecting a product for quotation will reveal the last section shown in Figure 3.12, which contains a list of selected products and enables the user to generate a quotation email easily or extract the data to CSV format to import it into the ERP system. The CSV extraction is currently tailored to ELKOV's needs but could be easily modified to meet other customers' needs.

In terms of other features, the application also supports Czech and English localization, which is made possible with the vue-i18n library. Other features are planned for the future of this application, such as quotation history, user management, and direct email inbox integration. However, this set of features should already provide quotation assistants with a powerful tool to simplify the quotation process. We have confirmed this based on the interest from the industry partners, especially ELKOV.

4 Evaluation of Solutions and Experiments

In this chapter, we will discuss more closely what GEOMAT's acceptance criteria were and how the solution was evaluated. Then we will talk about our fine-tuning experiment, how the training progressed, what the evaluation metrics were, and the achieved performance.

4.1 Testing the Solution

Before the project started, we had agreed with GEOMAT that the solution would have to achieve at least a 70 % success rate on the relevant metric, or the solution would not be accepted. Even though GEOMAT would prefer stricter criteria of absolute accuracy (i.e., success means that extraction was flawless; otherwise, extraction failed), we have settled on using weighted accuracy. This metric will ensure that quotation accuracy will be fairly based on evaluating its underlying parts and then averaged across all evaluated RFQs. The metric verifies the extraction process as follows per one RFQ.

- category The First condition checks if categories from ground-truth data are present in the extracted products. This is the preliminary condition for correct categorization since if extracted products have a wrongly assigned category, they will not receive any points. Therefore, we do not award any points for correct categorization but subtract them so that products with categories not present in the ground-truth data will not get checked.
- name Each product from the ground truth data must be present in the extracted products, and therefore, we expect that there will be a name match. This is a hard requirement for awarding any further points since this serves as a mapping from the ground-truth data to extracted products. The condition for this is that names are at least 75 % similar based on cosine similarity. Therefore, all the product names from the ground-truth data and the extracted products must be embedded. (Choice of embeddings was discussed in Section 3.2). A point reward for correctly specifying the name is 50 % since this establishes that the ground-truth product was found and has the correct category.

quantity Another important evaluation criterion is product quantity. The reward for this field totals 25 % but is split into two parts: the actual quantity value and quantity type. For the quantity value, there is an agreed tolerance of 10 % from the ground-truth value, and the quantity unit must match exactly. However, both fields contribute to the total reward evenly, with 12.5 % per field.

specification The last evaluation criterion is product specifications, contributing to the overall product evaluation of 25 %. As multiple specifications may exist for each product, this reward is split evenly among them. However, some parameters may have present units (i.e., tensile strength, density) and, therefore, are evaluated in a similar manner as a quantity, with the only difference being that the parameter value accounts for 75 % of the reward of the specific parameter, and the unit evaluates only for 25 %. This is because the parameter units are usually not so important since they can be reliably estimated. The rewards from all specifications are then considered since the rewards for specifications are averaged.

This metric allows us to focus on important parts of the extraction process since rewards for correctly matching a name (and, therefore, category) are much more important than matching a single specification value. However, we have also observed that matching all parameters proves a bigger challenge for the LLM, which inspired us to perform the model fine-tuning. We have decided to use weighted accuracy in the fine-tuning process with the F1 score but with slight modifications in its computation compared to how it is used in the GEOMAT solution. The main difference is in mapping the ground-truth data to extracted products. This will be done by evaluating the highest potential weighted accuracy (not considering specifications) for extracted products with ground-truth data. Therefore, a category check is later required, and the award for matching the category correctly is 25 % of product points, and the name reward is decreased from 50 % to 25 % points.

The computation of the F1 score follows the standard procedure of computing true positives (TP), false positives (FP), and false negatives (FN). All the mentioned fields are checked similarly, such as when

computing weighted accuracy; however, when fields match, we award one true positive point. If the fields do not match, we award one false negative point since there is a field mismatch and one false positive point since there is redundant information the model should not have provided. When ground-truth data contains some field missing from the extracted product, we award one false negative point. If an extra field is added to the extracted product, we award it with one false positive point. For both of these missing cases, it can be more than one point if the ground-truth data matches the object field (like quantity, which is presented by quantity value and unit). Afterward, the metrics are computed in the following way.

$$\text{Precision} = \frac{TP}{TP+FP}; \text{Recall} = \frac{TP}{TP+FN}; \text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

As we have already mentioned, the GEOMATs dataset consists of 39 samples; however, initially, it had only twenty samples. Therefore, the first twenty samples were primarily used for debugging the prompt, and from the other nineteen samples, only selected specific cases were used for debugging. This is still a relatively small subset of incoming RFQs, so the main goal was not to do a scientific evaluation of training to validation split but rather to capture the nature of the GEOMATs data as best as possible. The evaluation was run separately on those two parts, and based on the metric described above, we report that the solution achieved 94.8 % on the dataset of twenty samples and 96.4 % on the dataset of nineteen samples. Therefore, we report an aggregated performance of 95.6 % based on the weighted accuracy metric. There was also acceptance testing on the GEOMATs side, where they wanted to evaluate if they deemed the results satisfactory based on the real usage, and they reported a performance of 82 % based on their preferences. However, they did not back up this number with data.

4.2 Model Fine-Tuning

Since we have described the fine-tuning experiment thoroughly in the previous chapter, we will focus on the reported training, validation, and test evaluation results and provide type error analysis. For a quick recapitulation, the following fine-tuning experiment was run for three

epochs as cross-validation training with five folds on 1024 synthetically generated samples based on the GEOMATs data.



Figure 4.1: Training loss for all folds on 75 training steps

The first results we want to present from the training metrics are loss values for the trained models shown in Figure 4.1. The value of the loss function steadily decreases for all folds except fold one, which exhibits training spikes and results in lower loss at those particular points. We believe this could happen because there is an outlier in the fold 1 data, perhaps more easily predicted than standard micro-batches. This trend for fold one can also be seen in all the following figures but is disregarded since the model produced from this specific fold was not a top performer in evaluation loss.

The following Figure 4.2 shows the grouped training loss for all the folds. This visualization supports our previous claim that training five models did not yield many different models since the only outlier is Fold 1, which is relatively unstable. Other fold reports similar results with only slight differences in their training loss.

4. EVALUATION OF SOLUTIONS AND EXPERIMENTS



Figure 4.2: Training loss for all folds on 75 training steps grouped

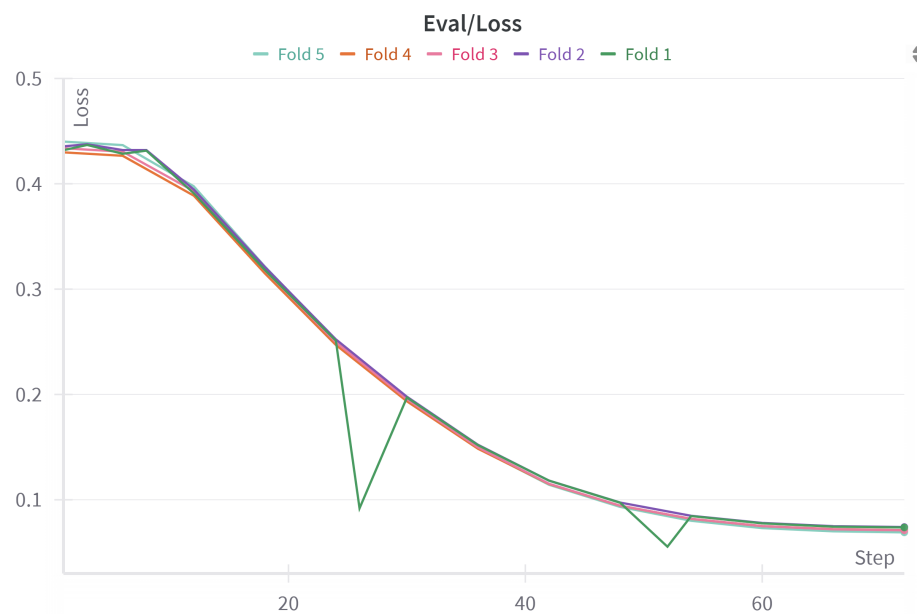


Figure 4.3: Validation for all folds on 72 validation steps

Figure 4.3 represents the validation loss metric on all 72 validation steps. However, based on the checkpoint storage and time requirements, the evaluation loss was evaluated only every 6th step, and the graph is interpolated. This explains why the training loss graph in Figure 4.1 continues for 75 steps, but the validation loss graph ends at 72. We can see the same behavior in the training loss graph in Figure 4.1, except for Fold 1, which shows two spikes; the validation loss remains similar across all the folds. From the graph, it is evident that the rate of evaluation loss decreases rapidly until around the 42nd step, so we believe the number of epochs could be lowered or at least not increased. We rather suggest that performance scaling should focus more on LoRa adapter ranks, prompt engineering or on bigger models overall.

Table 4.1: Cross-Validation Results: Final Training Loss (Step 75) vs. Best Validation Loss (Step 72)

Fold	Training Loss	Validation Loss	Ranking
1	0.079	0.0737	4th
2	0.072	0.0741	5th
3	0.069	0.0710	2nd
4	0.063	0.0711	3rd
5	0.073	0.0691	1st

The training and evaluation loss numbers from their last evaluated step can be seen in Table 4.1. Even though the differences are slight, we have ranked the folds based on the achieved validation loss, with the best three being folds 5, 3, and 4. All the folds will be evaluated on the latest GEOMAT RFQs testing dataset, which consists of 30 samples constructed for this final evaluation. The top three performing folds will also be evaluated on the mentioned *a_100* dataset so we can investigate how they performed on the data they have seen in the training. Further, only Folds 3 and 5 will be used for more clarity in the later comparisons since they were the best performers in our fine-tuning experiment.

4. EVALUATION OF SOLUTIONS AND EXPERIMENTS

Table 4.2: Model Performance Benchmark on TEST and A100 Datasets. Metrics are weighted accuracy, precision, recall and F1 score.

Model	Dataset	W. Acc	P	R	F1	Rank
GPT-4o	TEST	0.8314	0.8110	0.8113	0.7988	1st
	A100	0.7843	0.8962	0.7127	0.7912	
GPT-4o-mini	TEST	0.7252	0.8241	0.6642	0.7261	2nd
	A100	0.7115	0.8530	0.6558	0.7393	
Fold 5	TEST	0.6610	0.5602	0.6690	0.5949	3rd
	A100	0.7217	0.7020	0.7065	0.6999	
Fold 4	TEST	0.6381	0.5261	0.6419	0.5614	7th
	A100	0.7243	0.6895	0.7091	0.6950	
Fold 3	TEST	0.6624	0.5601	0.6655	0.5911	4th
	A100	0.7237	0.6932	0.7085	0.6955	
Fold 2	TEST	0.6177	0.5018	0.6197	0.5333	8th
Fold 1	TEST	0.6387	0.5445	0.6420	0.5712	5th
Baseline	TEST	0.5925	0.5803	0.6003	0.5631	6th
	A100	0.6596	0.6934	0.6431	0.6635	

Based on the results presented in the Table 4.2, we can see that GPT models dominate the benchmark based on the F1 performance. However, when comparing the best-performing Fold 5 and Fold 3 models on weighted accuracy with the GPT-4o-mini, which is a more appropriate comparison, the difference exceeds just 6 %. Regarding fine-tuning improvements, we can see only a 3 % difference in the F1 score on best-performing folds but almost a 7 % increase in weighted accuracy when comparing baseline and Fold 3. When comparing the recall, we can see that the best-performing fine-tuned models slightly overperform the GPT-4o-mini on the test dataset, but their problem is their poor precision. When considering the *a_100* dataset, we can tell that Folds 2 to 4 slightly improved their F1 score, which means that folds learned the training data only to a limited extent. Another interesting finding is that in terms of weighted accuracy, the GPT-4o model

4. EVALUATION OF SOLUTIONS AND EXPERIMENTS

performed 5 % better on the test dataset; therefore, its performance on real data was better than on the synthetic a_{100} dataset. There is a disclaimer to the evaluation that GPT-4o-mini and Fold 2 models had a problem evaluating the a_{100} dataset, which resulted in skipping three samples for GPT-4o-mini and one for Fold 2. However, we disregard this error since the contribution of 100 samples is minimal.

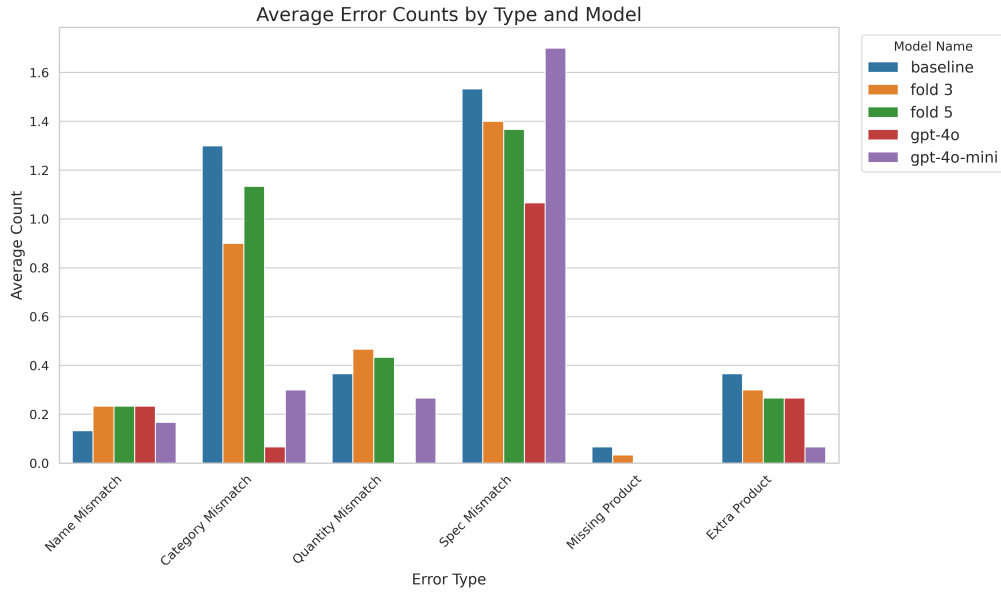


Figure 4.4: Distribution of error types per selected models

We have also conducted an error type analysis, as seen in the Figure 4.4. For this and the following comparisons, we have chosen the best-performing models: GPT-4o, GPT-4o-mini, Fold 5, Fold 3, and the LLAMA 3.1 Baseline model. The first metric we discuss is the error type relative to the average count. This can be interpreted as the sum of a particular type of error happening across all samples divided by the number of samples. The data shown has been created based on the test dataset, and we have six different types of errors: name mismatch, category mismatch, quantity mismatch, specification mismatch, missing, and extract products.

Based on the name mismatch, we can tell that this error does not happen that often, and what is surprising is that it happens fewer times for less powerful models like LLAMA 3.1 Baseline and GPT-4o-

mini. However, the rate of the name mismatch is relatively low for all compared models.

More interesting is category mismatch, as we can see that it rarely happens for the GPT models (almost non-existent for the GPT-4o) but is present once in every product on average for the LLAMA models. We can see that fine-tuning helped reduce this error since the rate for Fold 3 is around 0.9 and the baseline LLAMA performed around 1.3, but there is room for improvement when comparing fine-tuned folds with GPT models.

We can see that quantity mismatch never happened for the GPT-4o, and the GPT-4o-mini performs only slightly better than the LLAMA models. However, specification mismatch is another more prevalent error type, which happens around 1.4 times per sample on average for the LLAMA models. The best-performing GPT-4o achieved approximately one specification mismatch per sample. However, this type of error isn't that important since, in terms of importance, a single specification mismatch does not add up to much.

The missing product error is almost non-existent, which is a good sign, and the extracted product appears on average in one out of five RFQs. Based on this analysis and in terms of evaluation importance, the most significant improvements could be in category mismatch for the LLAMA models, which would significantly help to improve their performance. This is justified by the fact that the award for correct categorization is 25 % when using the weighted accuracy metric, while the product specifications category gets 25 % for all of the product specifications.

We have also created a box plot visualization of the F1 achieved performance on the test dataset, which can be seen in Figure 4.5. This visualization gives insight into the median performance and spread of achieved results. An interesting finding is that even though GPT-4o achieves the median performance of 86.5 %, there are still RFQs on which it fails drastically, with the lowest achieving only 33.3 %.

Another fact is that the GPT models are the only ones from the comparison that achieve perfect F1 scores for some of the RFQs. When comparing the best-performing Folds 3 and 5, we can see that their median performance is above 60 %, specifically 63 % for Fold 5. If we compare this with the median performance of the non-finetuned baseline LLAMA model, which achieves only 54.5 %, we can see an

8.5 % performance increase. What is concerning is the spread of the model performance. This is especially interesting since the graph comprises 150 individual evaluations, and we can see only two outliers present. For example, the performance spread achieves slightly more than 60 % for the GPT-4o. We believe this could be improved by prompt engineering or further fine-tuning, focusing on the RFQs that achieved lower performance.

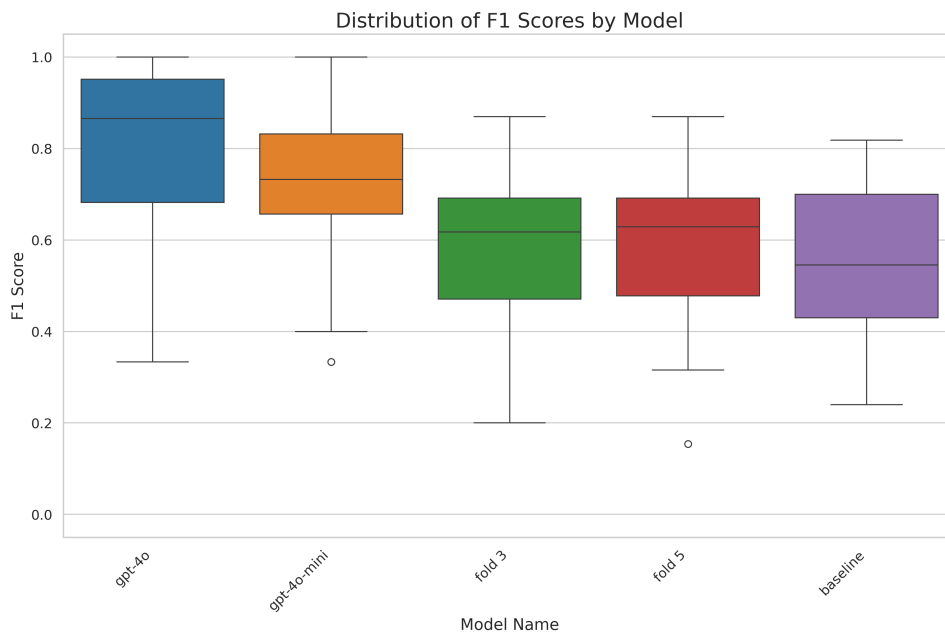


Figure 4.5: Comparison of F1 score across best performing models and LLAMA baseline

We also wanted to focus on the weighted accuracy performance of the models, so we have constructed similar boxplots focusing on this metric in Figure 4.6. The first big difference compared to the F1 scores is a tighter spread of the model performance. Weighted accuracy appears to be a more benevolent metric than F1 scores since the performance for the LLAMA models increased, even achieving perfect extraction scores. Another interesting finding is that when comparing GP-4o-mini vs. Fold 3 and 5, they achieved much closer performance to GPT-4o-mini, which contrasts with the same comparison on the F1 score. Still, it should be noted that Fold 3 and 5 achieve similar median

scores as they did on the F1 score, while the GPT-4o-mini achieves a median performance of 75 %.

We can see that models perform better on weighted accuracy when comparing their performance with achieved F1 scores. Considering this comparison with the distribution of errors in Figure 4.4, we believe that the cause is a large rate of specification mismatches. Weighted accuracy is less sensitive to specification errors since, regardless of their amount, they contribute only maximally 25 % of product reward. However, regarding the F1 score, errors in each particular specification contribute evenly to the product score. Therefore, the weighted accuracy describes how well we have identified each product, but the F1 score describes the overall accuracy of every piece of extracted information.

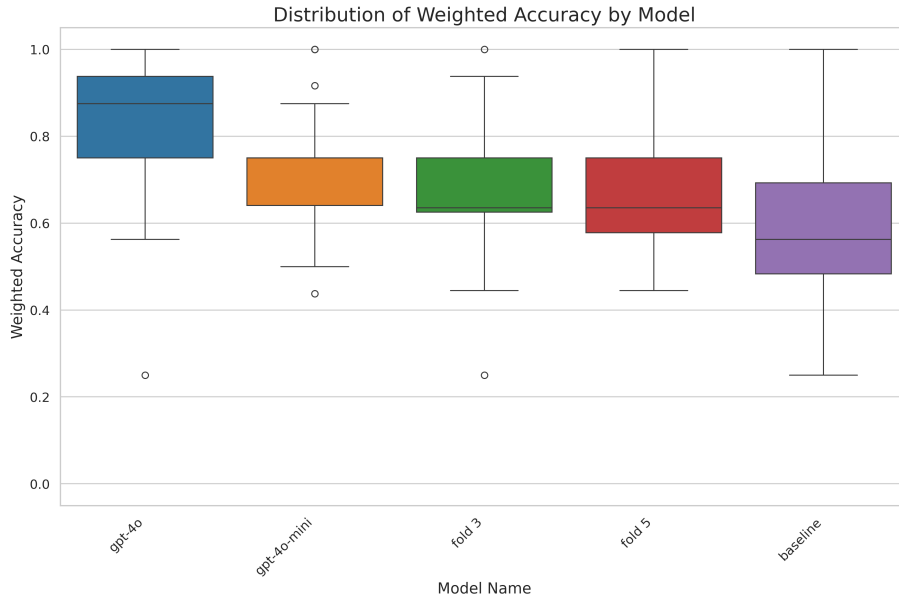


Figure 4.6: Comparison of weighted accuracy across best-performing models and LLAMA baseline

We have already seen the difference between the F1 score and weighted accuracy on the aggregated data. However, we also wanted to examine the F1 score performance on a per-sample basis. Therefore, we have created visualizations that show the relation between Recall, Precision, and their effect on the F1 score for multiple model

scenarios. The first such presented scenario can be seen in the Figure 4.7, where we compare the performance of GPT models. As we can see, the GPT-4o performance is more focused on $Recall > 0.8$ and $Precision > 0.8$. The GPT-4o-mini performance is more dispersed, and while it achieved complete Precision on multiple RFQs, quite a lot of RFQs did not reach 80 % precision. Regarding Recall, the GPT-4o-mini hardly breaks through 80 % performance. For the GPT-4o, we think that key performance improvement would be focusing on Precision for around 8 RFQ samples in the 40 to 60 percent range. Lastly, when looking at F1 contours, we can see that GPT-4o remains mainly in the 80 to 100 percent range, but GPT-4o-mini mostly remains in the 60 to 80 percent range. We believe this could be simply because the GPT-4o is a different class in terms of parameters compared to the GPT-4o-mini.

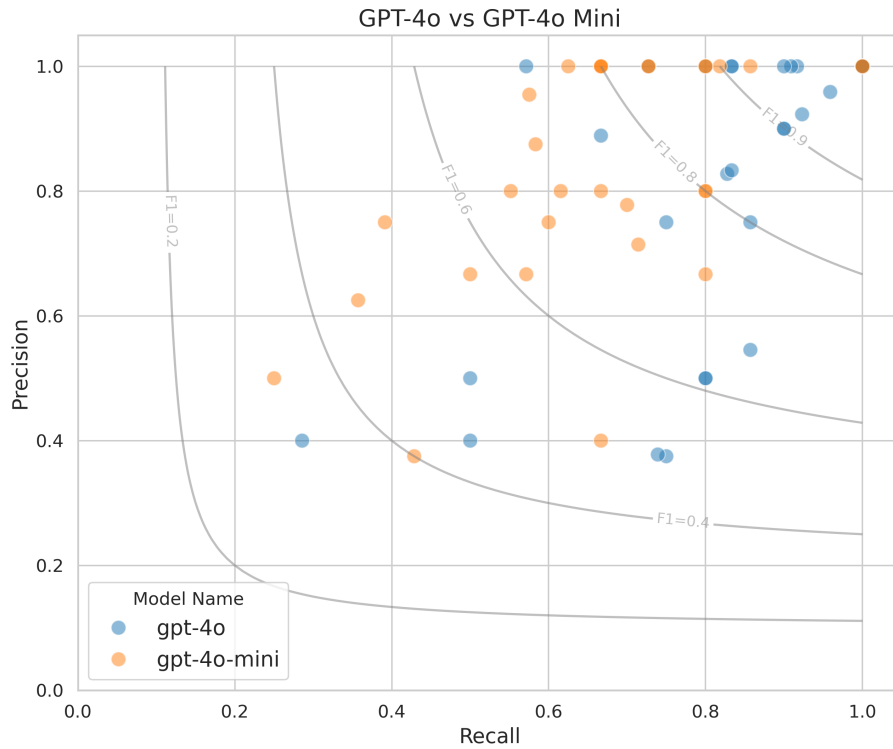


Figure 4.7: Comparison of precision, recall and F1 relation for GPT models

4. EVALUATION OF SOLUTIONS AND EXPERIMENTS

In the following comparison in Figure 4.8, we will more closely look at the performance of LLAMA models. The immediate difference we can see is that none of the samples achieved an F1 score greater than 90 percent, with most samples performing in the 40 to 80 percent range. The models tend to perform similarly, as we can see in the overlap in the data points, with the best performers being folds 3 and 5, almost reaching the 90 percent F1 score with few samples. We can see a surprising result of the baseline model reaching the best precision, yet no model could reach complete precision on even one sample. In terms of recall, we can see that a few samples reached perfect recall, with Fold 5 performing best overall. Based on this visualization, we can confirm that the bottlenecking factor in the F1 performance is precision rather than recall.

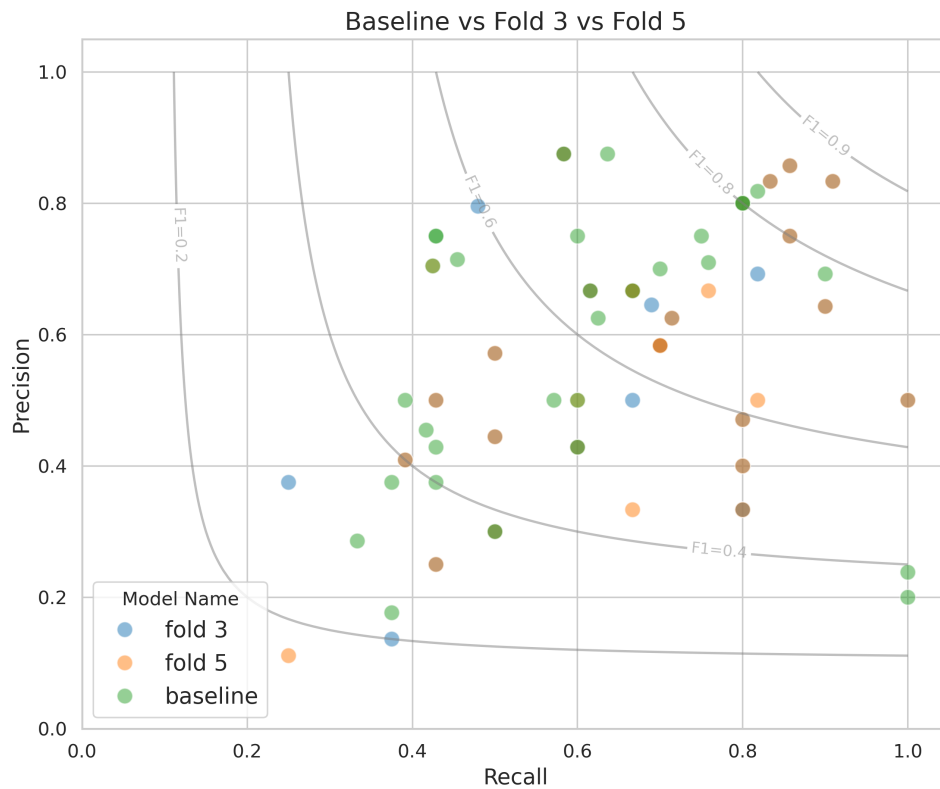


Figure 4.8: Comparison of precision, recall and F1 relation for LLAMA models

Lastly, we wanted to compare the performance of GPT-4o-mini with fine-tuned LLAMA models. This performance comparison can be seen in Figure 4.9, with a clear winner, GPT-4o-mini. We claim this because GPT-4o-mini manages to keep most of its F1 scores above the 60 percent range, while fine-tuned models have many samples in the 40 to 60 percent range. We can see that GPT-4o-mini manages to reach complete precision on multiple samples, which we establish that fine-tuned models have not managed to do. When considering achieved recall, we can tell that fine-tuned models either perform on par with GPT-4o-mini or even slightly outperform it. Therefore, we again confirm that fine-tuned models would benefit best if the precision were improved.

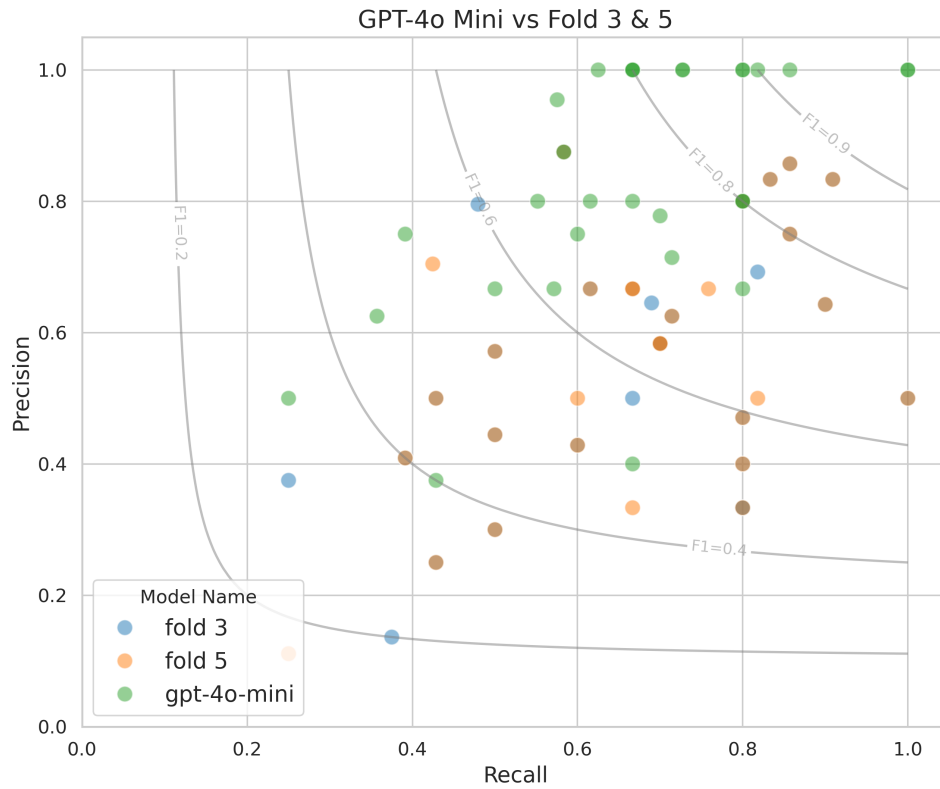


Figure 4.9: Comparison of precision, recall and F1 relation for GPT-4o-mini and best performing folds

Based on our quick analysis of predicted outputs from Fold 5, we confirm the findings from Figure 4.4 on page 49 that the model from Fold 5 often fails to categorize products and determine the quantity correctly. These errors primarily affect its precision performance. The recall is mainly affected by missing specifications, since we observed that if specifications were retrieved from the RFQ, they were determined correctly. The results for Fold 3 indicate similar problems to those of Fold 5. Therefore, we suggest that future work focuses on resolving the problems with the achieved precision.

5 Discussion and Future Work

Our main goal for this thesis was to enhance the RFQ process and help ease the work of quotation assistants. We have gone from a simple product extraction pipeline from GEOMAT through fine-tuning of LLAMA models based on synthetic data. We also developed a full-fledged application for RFQ process automation. This chapter discusses our key findings in each mentioned part, highlights what went according to plan and what did not, and proposes the next steps to resolve the existing issues.

5.1 GEOMAT's Solution

The GEOMAT's solution achieved a reasonably high average weighted accuracy performance of 95.6 % and was the cornerstone of our work. Based on non-empirical testing from the GEOMAT's side that achieved over 80 %, we do not think a flawless product extraction process will soon be achieved. Therefore, for such systems, it is still important to think of the human-in-the-loop concept and focus more on improving the process for humans rather than trying to replace them. Training such systems would require an extensive RFQ dataset, which we believe exists but is proprietary since it contains sensitive data for the involved companies. That is why we will be partially releasing the data we use, but in an anonymized form.

The prompt we designed in this phase remains relevant throughout our work, yet some flaws deserve improvement. One such issue is that we have declared product specifications in the prompt template, and they should be correctly placed in the system prompt. Besides that, instruction repetition should be investigated in terms of how much it helps the extraction process. It is also questionable if improvements could be made that would not require fine-tuning to improve product specification extraction, which is the most frequent error type for the GPT-4o.

The other remaining limitation is the ability to perform product extraction on larger RFQs, which we did not focus on when developing the GEOMATs solution, but encountered later in Offermate development. Other interesting possibilities that could be explored

are a comparison of other LLMs of GPT-4o size like DeepSeek-V3 or other thinking models, which were either not released when we were developing the GEOMATs solution or not as popular as they are today. While developing the solution, we considered comparing GPT-4o's performance to Anthropic's Sonnet or Opus models, but this has been taken out of the development scope.

The last important thing in developing the GEOMAT solution is using the GPT-4o model, which OpenAI regularly upgrades. Therefore, the performance we achieved in our work may change over time, and we hope it will provide GEOMAT with a better product extraction experience. GEOMAT has also reported that this solution has saved them approximately 160 hours per month, i.e., one full-time employee per month. This is a noticeable improvement, while they report that they only have to manage around 300 RFQs monthly.

5.2 Fine-Tuning Experiment

The primary goal of the model fine-tuning experiment was to explore open-source LLM options and improve product extraction performance, which we have achieved. This is an important aspect of AI solution development since companies dealing with sensitive data are reluctant to send it over the Internet and share it with other companies. This experiment consists of multiple phases, so we will address them individually, starting with synthetic data generation.

The main bottleneck of the synthetic data generation was the small number of RFQs used for the actual dataset generation. This has limited the variety of present RFQs and could be resolved by spending more time manually annotating incoming RFQs. When we were developing the GEOMATs solution, we were discussing whether we could, for example, reuse data quoted for past RFQs. However, this approach is problematic since the quoted products are not necessarily present in the RFQ. We have also tried to resolve this by creating the RFQ data directly with LLM, but we have found that this approach failed due to inconsistencies across the RFQ data with the ground truth. Nevertheless, if this is resolved, it could lead to a synthetic dataset of higher quality.

Regarding the training, we initially planned to check other open-source LLMs, such as Qwen models. They offer good performance-to-size ratios, especially with the newly released Qwen 3 models. The comparison of the 32B model would be particularly interesting. However, we had to cut this part due to the resources allocated to fine-tuning LLAMA models. The following important change for future work is training the models with a LoRa rank of 32 or adjusting the scaling factor more in favor of learned adapter weights. The LoRa rank could be even bigger, but the requirements for the VRAM would also have to be scaled together with the dataset size. Another interesting idea is to sacrifice performance by using a lower quant of the models (like FP8), conduct QLoRA [18] fine-tuning, and focus on larger models of 32B or 72B sizes. Addressing the training design, we suggest against using cross-validation training for future work since we report only slight variance in trained folds. Therefore, we suggest increasing LoRa parameters, using bigger models, and not creating multiple models.

To address the evaluation part, we suggest focusing more on the details of the results generated by the fine-tuned models, especially in terms of achieved precision. Precision issues could also be resolved by adjusting the prompt for the LLAMA models since the system prompt was built for and debugged on the GPT-4o model. However, all evaluated models should use the same system prompt for an appropriate and fair comparison.

5.3 Offermate – RFQ Process Automation

The last part of this work was dedicated to creating a full-fledged application for RFQ process automation. First, we will start with the backend, which was the most important part of this project. Based on the testing we did during the development with the ELKOV data, we see three main areas that would benefit from improvement.

The ELKOV data introduced a large RFQ to the process. While the application reliably extracts ten to twenty products per RFQ, its performance worsens when the product count increases to tens or, in rare cases, even hundreds. The issue could be resolved simply by segmenting the RFQ data into parts so as not to overflow the context

of the LLM. Secondly, the processing time for the whole process, especially on the larger RFQs, can reach into minutes, which is nothing compared to manual processing by the quotation assistant, but still considerable time that users have to wait for the finalization. This could be easily resolved if the models were moved from the CPU to a reasonable GPU, but this is currently out of the scope of this work. The last part is the matching process in terms of the quality of the matches. As we have already mentioned, since we were utilizing just the CPU, we did not go for the larger SOTA retrieval models, but this could further improve the quality of the matches. Another technique that could improve the matching quality could be querying the sales text and description by combining named entity recognition with ILIKE queries, which we propose as further work.

Regarding the frontend part of the application, it performs its job reasonably well but would benefit from a proper UI/UX redesign. The benefit of this work is that the backend and frontend are standalone parts, so other work could build upon the existing backend without using the same frontend. Feature-wise, we intend to implement history for the RFQ processing so that the sales assistant can easily return to previously quoted RFQs. Even though we do not plan to remove the human-in-the-loop approach, we believe that even further automation could be proposed. The main idea of further automation is to redirect all incoming RFQs to a shared inbox, which could be read and automatically processed by our RFQ automation. The matched and extracted products will be stored for the quotation assistant for the review process. This, however, requires much future work, which is out of the scope of this publication.

6 Conclusion

The main goal of this work was to improve the RFQ process with automation using LLMs and implement software for partial automation of this process. We have implemented two separate software systems, a simpler product extraction pipeline utilizing GPT-4o for GEOMAT Ltd., which helped them reduce 160 hours per month (one full-time employee), which they spent on processing approximately 300 RFQs each month. This solution has achieved 95.6 % average weighted accuracy on the data GEOMAT has provided and succeeded in GEOMAT's non-empirical acceptance testing with a score of 82 %. This has proven that using already available LLMs combined with prompt engineering can enhance companies' sales workflows.

The more sophisticated software was developed mainly as an RFQ automation product, of which ELKOV Ltd. took an interest. This software takes GEOMAT's automation a step further, enhancing the product extraction process with more options for input data using OCR and introducing product matching capabilities, which speed up RFQ processing time even further. The matching is based on a tiered search methodology utilizing multiple searching approaches to ensure relevant products are retrieved. One such approach is the similarity computation of embedded extracted product data with the client's product catalog, utilizing a PostgreSQL database with vector extension and indexes for reliable scaling even on tens to hundreds of thousands of products. Other information retrieval techniques, like LLM query enhancement and results reranking, ensure relevant products are retrieved. The result is a full-fledged application with a SPA frontend and a standalone backend, which enables quotation assistants to easily and quickly process incoming RFQs.

Our separate research goal was to also compare OpenAI's closed-source GPT-4o model performance with a smaller open-source LLAMA 3.1 8B model, which we have fine-tuned on synthetically generated data. The synthetic data was generated based on fixed generation from the existing pool of products with parameter variance for values. This resulted in a dataset of 1024 RFQ samples focusing on medium-sized RFQs with four to twelve products. The fine-tuning process utilized cross-validation training, LoRa with matrices

of rank 16 without model quantization, flash attention, and FSDP for multi-GPU training handled by the Axolotl package, which resulted in performance-efficient fine-tuning. We have further extended the comparison by inclusion of the GPT-4o-mini model and report an 8.5 percent increase in median performance and a 3.18 percent average performance on testing the RFQ dataset when comparing fine-tuned models with a non-fine-tuned LLAMA model. The fine-tuned models still fall short of the GPT-4o-mini model, which is explained by a thorough error and performance analysis phase. Further fine-tuning with more performance-heavy LoRa parameters, a higher quality RFQ dataset, or a choice of newer and more powerful LLM would be required to improve this performance further.

A Overview of Thesis Archive

General description of submitted archive. It is recommended to inspect README.md in the project root, since it contains a description of each project that is present in the thesis archive. Each underlying project folder also contains README.md with further instructions on running the specific project with a detailed directory structure.

```
root directory
├── README.md           ... General project instructions.
├── geomat/            ... GEOMAT solution.
├── finetuning/         ... Fine-tuning experiments.
└── offermate/         ... Offermate solution.
```

Bibliography

1. VIRTANEN, Tatu; PARVINEN, Petri; ROLLINS, Minna. Complexity of sales situation and sales lead performance: An empirical study in business-to-business company. *Industrial Marketing Management*. 2015, vol. 45, pp. 49–58. ISSN 0019-8501. Available from DOI: <https://doi.org/10.1016/j.indmarman.2015.02.024>.
2. ZHAO, Yu; GAO, Haoxiang. *Utilizing Large Language Models for Information Extraction from Real Estate Transactions*. 2024. Available from arXiv: 2404.18043 [cs.CL].
3. HU, Edward J.; SHEN, Yelong; WALLIS, Phillip; ALLEN-ZHU, Zeyuan; LI, Yuanzhi; WANG, Shean; WANG, Lu; CHEN, Weizhu. *LoRA: Low-Rank Adaptation of Large Language Models*. 2021. Available from arXiv: 2106.09685 [cs.CL].
4. YANG, Yang; WU, Zhilei; YANG, Yuexiang; LIAN, Shuangshuang; GUO, Fengjie; WANG, Zhiwei. A Survey of Information Extraction Based on Deep Learning. *Applied Sciences*. 2022, vol. 12, no. 19. ISSN 2076-3417. Available from DOI: 10.3390/app12199691.
5. HAN, Ridong; YANG, Chaohao; PENG, Tao; TIWARI, Prayag; WAN, Xiang; LIU, Lu; WANG, Benyou. *An Empirical Study on Information Extraction using Large Language Models*. 2024. Available from arXiv: 2305.14450 [cs.CL].
6. BEASON, Sterling; HINTON, William; SALAMAH, Yousri A.; SALSAMAN, Jordan. Automated Analysis of RFPs using Natural Language Processing (NLP) for the Technology Domain. *SMU Data Science Review*. 2021, vol. 5, no. 1. Available also from: <https://scholar.smu.edu/datasciencereview/vol5/iss1/1/>.
7. BILAL, Esha; ÅSTRÖM, Oscar. *Generative AI for Sales : A Study on the Potential of Retrieval Augmented Generation for Response Automation in the Request for Proposal Process in Telecommunications*. 2024. TRITA-EECS-EX, no. 2024:271.
8. XU, Derong; CHEN, Wei; PENG, Wenjun; ZHANG, Chao; XU, Tong; ZHAO, Xiangyu; WU, Xian; ZHENG, Yefeng; WANG, Yang; CHEN, Enhong. Large language models for generative information extraction: a survey. *Front. Comput. Sci.* 2024, vol. 18, no. 6.

- ISSN 2095-2228. Available from DOI: 10.1007/s11704-024-40555-y.
9. LIU, Ningwei; FENG, Charles; WU, Shuning; CHAN, Aaron; FULTON, Joel. Automate RFP Response Generation Process Using FastText Word Embeddings and Soft Cosine Measure. In: *Proceedings of the 2019 International Conference on Artificial Intelligence and Computer Science*. Wuhan, Hubei, China: Association for Computing Machinery, 2019, pp. 12–17. AICS 2019. ISBN 9781450371506. Available from DOI: 10.1145/3349341.3349362.
 10. NOVOTNÝ, Vít. Implementation Notes for the Soft Cosine Measure. In: *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. Torino, Italy: Association for Computing Machinery, 2018, pp. 1639–1642. CIKM '18. ISBN 9781450360142. Available from DOI: 10.1145/3269206.3269317.
 11. SALEEM, Summra; ASIM, Muhammad Nabeel; DENGEL, Andreas. *ReqNet: A Large Language Model Driven Computational Framework for Automated Requirements Extraction from Unstructured Documents*. Available at SSRN, 2025. Available from DOI: 10.2139/ssrn.5122555. Preprint.
 12. DAO, Tri. *FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning*. 2023. Available from arXiv: 2307.08691 [cs.LG].
 13. ZHAO, Yanli; GU, Andrew; VARMA, Rohan; LUO, Liang; HUANG, Chien-Chin; XU, Min; WRIGHT, Less; SHOJANAZERI, Hamid; OTT, Myle; SHLEIFER, Sam; DESMAISON, Alban; BALIOGLU, Can; DAMANIA, Pritam; NGUYEN, Bernard; CHAUHAN, Geeta; HAO, Yuchen; MATHEWS, Ajit; LI, Shen. *PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel*. 2023. Available from arXiv: 2304.11277 [cs.DC].
 14. EMANUILOV, Simeon; DIMOV, Aleksandar. Billion-Scale Similarity Search Using a Hybrid Indexing Approach with Advanced Filtering. *Cybernetics and Information Technologies*. 2024, vol. 24, no. 4, pp. 45–58. ISSN 1314-4081. Available from DOI: 10.2478/cait-2024-0035.

BIBLIOGRAPHY

15. ENEVOLDSEN, Kenneth; CHUNG, Isaac; KERBOUA, Imene; AL., Márton Kardos et. MMTEB: Massive Multilingual Text Embedding Benchmark. *arXiv preprint arXiv:2502.13595*. 2025. Available from DOI: 10.48550/arXiv.2502.13595.
16. WANG, Liang; YANG, Nan; HUANG, Xiaolong; YANG, Linjun; MAJUMDER, Rangan; WEI, Furu. Multilingual E5 Text Embeddings: A Technical Report. *arXiv preprint arXiv:2402.05672*. 2024.
17. LI, Chenxia; LIU, Weiwei; GUO, Ruoyu; YIN, Xiaoting; JIANG, Kaitao; DU, Yongkun; DU, Yuning; ZHU, Lingfeng; LAI, Baohua; HU, Xiaoguang; YU, Dianhai; MA, Yanjun. *PP-OCRv3: More Attempts for the Improvement of Ultra Lightweight OCR System*. 2022. Available from arXiv: 2206.03001 [cs.CV].
18. DETTMERS, Tim; PAGNONI, Artidoro; HOLTZMAN, Ari; ZETTLEMOYER, Luke. *QLoRA: Efficient Finetuning of Quantized LLMs*. 2023. Available from arXiv: 2305.14314 [cs.LG].