WIRTSCHAFTS UNIVERSITÄT WIEN VIENNA UNIVERSITY OF ECONOMICS AND BUSINESS

Bachelor Thesis

EFMD EQUIS ACCREDITED

Extracting supply chain information from text data using LLMs

Joshua Zelle

Date of Birth: 16.08.1999
Student ID: 11922754

Subject Area: Artificial Intelligence

Studienkennzahl: 033/560

Supervisor: Dr. Amin Anjomshoaa

Date of Submission: 21.03.2024

Department of Information Systems & Operations Management, Vienna University of Economics and Business, Welthandelsplatz 1, 1020 Vienna, Austria

DEPARTMENT FÜR WIRTSCHAFTSINFORMATIK UND OPERATIONS MANAGEMENT DEPARTMENT OF INFORMATION SYSTEMS AND OPERATIONS MANAGEMENT

Acknowledgements

I want to thank my supervisor, Amin Anjomshoaa, for his patience and ongoing support. I further want to express my gratitude to my colleagues at the Supply Chain Intelligence Institute Austria (ASCII), especially Peter Klimek and Georg Heiler for their tips and guidance. Also, tackling the Common Crawl challenge would not have been possible without their inputs and the resources from ASCII.

Contents

List of Figures

List of Tables

Listings

Abstract

Large Language Models (LLMs) have garnered significant attention in research, particularly their application to core natural language processing (NLP) tasks like information extraction (IE), which has seen a notable uptick in interest recently. Moreover, the deployment of LLMs across specialized fields, including medicine and industry, is expanding. A promising opportunity lies in supply chain analysis, an area where traditional methods often fall short in unraveling the complexities of modern, often non-transparent supply networks. Applying NLP advancements to these challenges in supply chain analysis, this thesis investigates the question: Can LLMs accurately extract supply chain information from text data? It delves into this through two subquestions: (1) how do different LLMs compare on this task and (2) how is the performance given different input texts?

To address these questions, we employ a ground truth classification framework within the semiconductor industry as a case study. Additionally, two custom datasets were curated as input for the LLMs: one derived from company descriptions sourced from the business intelligence database Orbis, and the other extracted from text found on company websites, utilizing the web-scale data archive Common Crawl. The LLMs were tasked with classifying semiconductor companies into stylized segments of the supply chain, thereby transforming the task into a hybrid problem involving IE and multi-label classification.

An interesting implication of this approach is that, if successfully tested against a ground truth, it could then be employed in novel scenarios, potentially uncovering previously unknown insights into supply chains. The results demonstrate that while LLMs show promise in this domain, their performance is significantly influenced by the source text quality. The study highlights the importance of general reasoning abilities in LLMs even in task-specific models, at least for more complex scenarios. It also identifies key challenges in the usage of LLMs combined with targeted web data extraction, setting a foundation for future advancements of such LLM applications in general and for AI-driven supply chain analysis in particular.

1 Introduction

Understanding supply chains has gained importance over the last years in the face of a global pandemic, natural disasters, geopolitical conflict, among other issues [35]. There is, however, a major challenge associated with achieving such an understanding, namely the lack of high quality, up-to-date relevant data. Companies are usually unwilling to make their supply relationships public and often do not even know their own supply chain beyond their direct vendors [49]. Even if firms declare certain supply chain information publicly, there is no systematic way such data is collected and curated into a comprehensive dataset usable for analysis. While there are commercial business intelligence datasets [21] [24], they are quickly outdated, costly and often limited in scope.

One way to solve this problem is for human analysts to conduct research on the internet by reading news articles or company websites. Depending on the desired scope of the supply chain network, such a task quickly becomes tedious to do manually. Additionally, it is crucial to frequently update the extracted network. Thus, an automated extraction process is needed. In this thesis we aim to develop such a process by building a Natural Language Processing (NLP) pipeline that takes a company's description text as input and extracts information about its role in the supply chain. Recent advances [40] in the capabilities of large language models (LLMs), such as GPT-4 [43] or LLAMA-2 [51], seem promising for this task. Thus, we explore the following main question with two subquestions:

> **Can LLMs accurately extract supply chain information from text data?**
>
> 1. How do different LLMs perform? For instance, GPT [43], LLAMA-2 [51], or Mistral [29].
>
> 2. How is the performance with different text inputs? Specifically, we compare two text sources: the relevant text from a company's website and a company's description from the business intelligence database Orbis [44].

To answer these questions, we attempt to classify companies from the global semiconductor industry into a stylized value chain as a case study. As a ground truth, we use a dataset by the Center for Security and Emerging Technology at Georgetown University from their report *The Semiconductor Supply Chain. Assessing National Competitiveness* [11].

In setting up this study, we aim to practically test the ability of LLMs to extract information from text and infer more abstract insights about a company's position within a stylized supply chain. By selecting this particular industry, we tap into a sector that is both vital to global technology and representative of today's supply chain complexity [14]. Ultimately, this investigation is expected to shed light on the practical utility of LLMs for supply chain analysis, potentially offering a new tool for analysts and industry professionals who need to stay informed about the rapidly changing dynamics of their sectors. Beyond the implications for supply chain mapping, it should also contribute to the growing field of LLM applications to information extraction (IE), which is a classic task in NLP [62].

The study is structured into seven core sections. Following this introduction, the literature review explores prior applications of machine learning in supply chain mapping, gives a succinct overview of the state of the art of LLMs and highlights relevant IE studies that leverage LLMs. Our methodology is presented next, detailing the approach, the datasets used, model implementation, and evaluation criteria. We then describe the curated datasets as well as the process followed to obtain them. The results section presents the performance of various LLMs on the two datasets. This is succeeded by a discussion on the limitations and potential avenues for future research. The thesis culminates with a conclusion that synthesizes our findings and their implications for AI-driven supply chain analysis and IE using LLMs in practical settings. Appendices provide technical details on the implementations used, as well as longer excerpts from the datasets.

2 Literature Review

2.1 Previous applications of machine learning to supply chain mapping

Here we want to present some of the studies that have been conducted on applying machine learning to obtaining supply chain insights. We also briefly mention alternative methods, other than using LLMs, that have been attempted. For instance, some authors [25] [31] [39] [6] framed the task of supply chain mapping essentially as a link prediction problem in a network and used various machine learning methods to predict supply chain relationships between companies based on either network topology [25] [31] or company attributes such as location, industry, firm size etc. [39] [6]. In this thesis, we explicitly do not follow this approach. Rather, we aim to directly extract supply chain information from text only, following an NLP approach.

This NLP approach was already followed by Wichmann et al. [60] in 2020. There, the authors used a news article corpus to extract a supply network. They looked at individual sentences where in a pre-processing step two or more entities, e.g. company names, were detected with named entity recognition (NER). Then, those sentences were manually classified using Amazon Turk. Amazon Turk is an online platform where individuals worldwide can perform small tasks for compensation [3]. This labeled data was used to fine-tune an LLM, namely Bidirectional Encoder Representations from Transformers (BERT) [17], to predict the type of relationship. Thereby, the authors, like in this thesis, framed the problem as a classification problem. The results were promising as a prove of concept but, as the authors discuss [60], leave room for improvement as NLP technology would advance. As shown in section 2.2, the field of NLP has indeed advanced significantly, making another attempt at this problem an interesting endeavour. This thesis aims to contribute to that endeavour.

Reference	**Approach**	**Comments**
[25], [31]	Network Topology	Employed machine learning to analyze and predict links based on network structures.
[39], [6]	Company Attributes	Used attributes like location, industry, size etc. for prediction.
[60]	NLP	Leveraged NLP to extract supply chain information from News articles.

Table 1: Summary of previous applications of machine learning to supply chain mapping

2.2 State of the art of LLMs

LLMs have experienced rapid advancements and made a significant impact across various domains, as highlighted in recent survey literature. Naveed et al. [40], for instance, provide a comprehensive overview. They report that the number of papers released with the keywords "Large Language Model" grew from 57 in 2018 to 32,900 in 2023 [40], confirming the recent surge in research interest in the field. What makes LLMs so appealing is their versatility and breadth of use cases [22]. While their core area is classic NLP tasks, they have proven to be very useful for other applications and endeavours as well [40]. For example, LLMs such as GPT-4 have successfully completed professional and academic language-based exams such as the American Bar exam for

lawyers, where GPT-4 scored among the top 10% of participants [43]. It still remains a challenge to measure LLM's capabilities, however [12]. The most prevalent measure is empirical, that is, curating a dataset with labels or some other mechanism to quantify language-handling prowess [66]. The Hugging Face Open LLM Leaderboard, for instance, keeps the average score of open LLMs on diverse benchmark datasets [41]. At the time of this writing, the highest performing model on this board is Smaug72B [45] with an average score of 80.48. Besides the rapid development, there still remain challenges. On the one hand, improving the performance of the models even more, but on the other there are also concerns regarding the interpretability of how LLMs exactly work or ethical considerations related to algorithmic bias or the impacts on society broadly [22]. Recently, there was also an increase in regulatory attention placed on LLM development [40], as for example the EU Artificial Intelligence (AI) Act [20].

2.3 Information Extraction (IE) with LLMs

Extracting structured information from natural text is an important task in general, and in particular when doing research on the web [63]. Due to its high potential utility, researchers have tried to apply AI to this task and IE has become a well-known task within NLP. Given the recent advances in LLMs discussed in section 2.2, there have been increasingly many attempts to leverage LLMs for IE [62]. Xu et al. [62] give an overview over these attempts in the first comprehensive survey of using LLMs for IE. A useful classification of the studies that [62] use is to cluster them based on (1) the learning paradigm used and (2) the task that is primarily tackled. We shall adopt this classification here to obtain a more structured overview of what has been done and how our study fits into this:

2.3.1 Learning paradigms

The three learning paradigms in IE with LLMs that [62] identified are: supervised fine-tuning (SFT), few shot and zero shot. SFT is akin to traditional supervised machine learning: the model is fed a set of labeled training data, based on which it updates its parameters to fit the data well. That is, to predict the labels from the training dataset given an input text. The difference to traditional machine learning is that here a pre-trained LLM is taken as a starting point and then only *fine-tuned* with training data for a specific task, in this case IE [66]. Here it should be emphasized that the amount of data used for fine-tuning is several orders of magnitude smaller than the amount of data required to train the base model [40]. Next, few shot learn-

ing leverages only a handful of labeled examples to train the model. The line to SFT is somewhat blurry but it still remains a useful distinction to capture the difference of fine tuning a model with datasets of a few hundred megabytes or gigabytes versus a few hundred specific examples [66]. Lastly, zero shot learning requires the model to perform a task without any task-specific training data. It relies on the general abilities a model has gained from being trained on vast and diverse datasets [40]. Xu et al. [62] argue that zero shot studies have only started giving competitive results to the other paradigms in recent months, suggesting that now is a promising time for such research. Thus it seems adequate that this thesis will take place in a zero shot setting.

2.3.2 Tasks

IE can itself be broken down into more sub-tasks, the most prominent of which are: Named Entitiy Recognition (NER), Relation Extraction (RE) and Event Extraction (EE) [62]. Beside these more specialized tasks there is also a broader objective which is called in the literature universal information extraction (UIE) [34]. Since our objective of classifying companies into value chain steps does not fit into one of the specialized categories but is still information extraction we assume our task to fall under UIE.

Given that our study falls under zero shot UIE we shall explore three relevant and recent studies from this category in more depth and present some of the other projects in Table 2 for succinctness (based on the classification proposed by Xu et al. [62]):

References	**Learning paradigm**	**Task**
[55], [23], [35]	SFT	UIE
[68]	SFT	NER
[32], [5], [13], [56]	few shot	NER
[54]	few shot	RE
[33], [8]	few shot	UIE
[48], [57], [59]	zero shot	UIE

Table 2: Summary of studies by learning paradigm and task.

2.3.3 Previous work in zero shot UIE

[59] developed ChatIE, which is a framework using the out-of-the-box ChatGPT model. The authors do not specify which version they use but given

the publication date it cannot be GPT-4, thus probably GPT-3.5. They turn zero shot IE tasks into a multi-turn question answering task where the model breaks down the task in small chunks. Unfortunately, the authors do not discuss how scalable they judge their approach to be, but given the many back and forths between the model, the bottle necks could be the length of the actual text possible to be analysed given maximum token input constraints of models and information loss.

[57] propose InstructUIE, a model that is fine-tuned to follow instructions and perform well on diverse IE tasks. The authors report that their model is competitive with state-of-the-art supervised methods such as various BERT [17] based models while outperforming out-of-the-box models like GPT-3.5 in zero-shot scenarios.

Similarly, [48] fine-tuned LLAMA-2-7B [51] to comply with instruction guidelines and propose GoLLIE (Guideline-following LLM for IE). For [48] guidelines are a mix of instructions, possible labels, examples etc. The authors argue that current out-of-the-box models would be capable for zero shot UIE but they struggle to closely follow guidelines, which is crucial for nuanced IE tasks [48]. To the best of our knowledge, GoLLIE achieves the best zero shot performance in IE at the moment, measured on diverse datasets and tasks [48].

3 Methodology

3.1 Approach

In this thesis, an experimental approach using real world data is followed. As discussed in section 2.3, in the field of IE, our research may be understood as a zero shot learning, universal IE setting, that is we do not fine tune LLMs on any additional data and our task does not fall under the typical IE applications such as NER, RE and EE. Furthermore, we set up the task as a multi-label classification problem, meaning that a company can be part of more than one class (in our case segment in the supply chain) [52].

Broadly speaking, our approach centers around using an LLM to categorize semiconductor firms according to their activities, based on relevant textual information about the firm. To illustrate this, consider the following hypothetical yet instructive example. Suppose the text T describes company x:

T = *"Company x stands at the forefront of the global semiconductor industry, renowned for its indispensable role as a supplier of premier raw materials. Our specialization in purifying top-tier silicon from sustainable, high-quality*

sources underscores our commitment to excellence and environmental stewardship in powering technological advancement..."

Then, we would feed this input text T to an LLM along with a prompt P, which could be as follows (note that this is a simplified example; the actual prompt used is detailed later in this section):

P = *"Given this company description T, which of the following categories best captures the company's activity: 'materials', 'tools'?".*

Subsequently, we obtain a predicted classification C for company x by inputting the prompt and text into the model. C is a subset of all possible classifications provided to the model in P, here {'materials', 'tools'}.

$$C = \text{LLM}(P + T)$$

where the "function" LLM denotes the process of obtaining a response from an LLM, given an input. In this example the predicted class should be 'materials'. It is important to note that the models we use have not been specifically trained for this task, hence the zero-shot setting.

To compare the model's classification C to a certain ground truth and evaluate its performance, we use the Georgetown dataset from the report *The Semiconductor Supply Chain. Assessing National Competitiveness* [11] as mentioned in the introduction (1). In this dataset the researchers assigned a company to one or more specific steps of the semiconductor value chain. For instance, company x prepares raw materials, company y does the chip design, company z fabricates the chips and packages them etc. We henceforth refer to this dataset as Georgetown dataset. For the companies in this dataset, we collect text data about the company's activities from two sources:

The first is firm description text from the business intelligence platform Orbis [44]. Orbis is a comprehensive company database curated by the business information publisher Bureau van Dijk and includes around 400 million firms worldwide. On those firms, the database offers information regarding the firm's financial position, address, legal status etc. but also an overview of its activity [61]. For our task we use the overview and similar fields.[1] The second source for the input text is the company's websites. For this we use the database of the non-profit organisation Common Crawl. Common Crawl crawls and downloads vast chunks of the Web every month and provides this data free of charge to the public [15]. Because the Georgetown dataset was compiled until 2021 [11], we use the last crawl from 2021 for our analysis.

This dataset setup has a few limitations though: (1) The state of the semiconductor value chain may have changed to some degree from the time

[1] *Disclaimer*: Orbis is a private database with restricted public access. We obtained the necessary data with an educational license provided by WU [61].

the Georgetown dataset was curated to the time our input texts originates. Since the Georgetown dataset was collected over the course of 2020 and 2021 [11], supply relationships may have changed, new players could have entered the industry and others could have gone out of business in that time window. Thus, if our approach cannot correctly classify a company, it may be due to other factors than just the model's failure. (2) Furthermore, some information contained in the ground truth dataset may come from different sources than the ones we have available in the form of website text and company descriptions. Hence, the model could not have found the required information within the available data to begin with. Despite these limitations, this approach of comparing against such a ground truth still seems informative and is widely used in similar projects in the literature [39] [31] [6] [60] [25].

After compiling the datasets, we feed this input text data about a company to an LLM. In this thesis we used the following models:

1. OpenAI's GPT-3.5-turbo which is presumably based on the GPT-3 [9] architechture but for which there was no paper published.

2. GPT-4 [43], the highest performing OpenAI model.

3. Mistral 7B-instruct-v0.2 [29] which is a high performing open-source model and is found as basis for many other models on the Hugging Face Leaderboard. Hugging Face is a leading platform for AI research, offering a vast repository of open-source LLMs that supports collaborative development and sharing [27].

4. CatPPT [2] which was the highest performing open-source model (with around 7B parameters) on the Hugging Face Leaderboard at the time of this writing [36].

5. GoLLIE [48] which we already mentioned in the literature review (2.3). It is supposed to be the best model for zero-shot UIE at the moment. It is based on the LLAMA-2 [51] model architecture.

3.2 Model Implementation

Given the models we selected we developed two different implementations: the OpenAI models were accessed via the OpenAI API and the Huggingface models were downloaded and run locally. We now briefly describe the implementation for both scenarios and then lay out the evaluation procedure. The implementations in Python code can be seen in appendices A and B, respectively.

3.2.1 OpenAI models

For the OpenAI models we used the function calling framework which is designed to lead to more deterministic outcomes and makes the task more reproducible [42]. This entails giving a more structured input to the model in the form of a json string like so:

```
json_schema = {
    "title": "Company",
    "description": "Identifying information about activities of a company based on its description.",
    "type": "object",
    "properties": {
        "category": {"title": "Company's Value Chain Category", "description": f"The companies predicted step in Value Chain, it should be part one of the following values: {categories_dict} I want you to just return the numbers of the class!", "type": "string"},
        },
    "required": ["category"],
}
```

Listing 1: JSON schema for OpenAI API function calling

And the prompt is prepared using a template:

```
prompt = ChatPromptTemplate.from_messages(
    [
        (
            "system",
            "You are an supply chain expert that classifies semiconductor companies based on their activities.",
        ),
        (
            "human",
            "Here is the description of a semiconductor company. Classify the firm based on its activity into one of these steps in the value chain: {input}",
        ),
        (
            "human", "Tip: Make sure to answer in the correct format, only return the numbers of your predicted class out of dict before. A firm can be part of up to 5 classes at the same time, seperated with comma. Keep in mind that big firms can make multiple steps while small firms probably specialize on just one step."
        ),
    ]
)
```

Listing 2: Prompt for OpenAI function calling

Here we see that first the model is given a system prompt which is supposed to steer the system in a certain direction, give it the right context in which to answer [42]. Then there are two *human* prompts which give the exact task with the input text and an additional tip to emphasize the desired output we want. We note that the `{input}` variable will later be replaced by the respective company description and that the `category_dict` in the `json_schema` contains a Python dictionary with the classes that are to be predicted as values and their index as keys. We chose to predict the numbers instead of the direct class names because without this work around the model often gave approximate class names or made up new class names that interfered with the evaluation procedure. With this setup the model returns a list of numbers which represent the value chain classes that the model predicts for company description. We used this setup for both, gpt-4 and gpt-3.5-turbo.

3.2.2 Hugging Face models

For the Hugging Face models we used, there is no comparable method available to the function calling in OpenAI to the best of the authors' knowledge. Therefore, we use the following prompt. Note that `categories` is this time a Python list and not a dictionary. In our experience, the Hugging Face models had no difficulty returning the desired classes. However, sometimes they did give additional text which is why we implemented a cleaning function which just extracted the classes from the model's output.

```
prompt = f"Here is a text from a semiconductor company website. Can you infer from this text what the company does? Classify it into these {len(categories)} classes which should be self-explanatory: {categories}.
 A firm can be part of 1 or many classes. For instance, big firms could engage in multiple activities while smaller firms tend to specialize in one class.
 Return your predicted class(es) as a python list with strings, e.g., ['Design'] or ['tool_resource', 'Fabrication']. Return only this list, nothing else.
 Here the website text:
 {description}

 REMINDER: only return your predicted class out of these {categories}, DO NOT use any other categories, and DO NOT return anything except the predicted list."
```

Listing 3: Prompt for Hugging Face models

3.3 Model Evaluation

After we obtained the predicted classes from the LLM for each company, we compare these classes to the ground truth from the Georgetown dataset. To quantify the models' performance we use the micro-averaged F1, Recall and Precision scores. We use the micro-averaged scores because they account for the label imbalance, which we will discuss in section 4, by aggregating the contributions of all classes. The definitions of the micro averaged scores are [38]:

Given a multi-label classification problem with a dataset of N instances, let $h(x_i)$ be the set of labels predicted by the classifier for the instance x_i, and let y_i be the true set of labels for that instance. Then, the micro-averaged metrics are defined as follows:

$$\text{Micro-averaged Precision} := \frac{1}{N}\sum_{i=1}^{N}\frac{|h(x_i) \cap y_i|}{|y_i|}$$

$$\text{Micro-averaged Recall} := \frac{1}{N}\sum_{i=1}^{N}\frac{|h(x_i) \cap y_i|}{|h(x_i)|}$$

$$\text{Micro-averaged F1 Score} := \frac{1}{N}\sum_{i=1}^{N}\frac{2 \times |h(x_i) \cap y_i|}{|h(x_i)| + |y_i|}$$

Where (for our case, without loss of generality):

- N is the total number of companies in the dataset.
- $h(x_i)$ represents the set of classes predicted by the classifier for company x_i.
- y_i represents the actual set of classes (ground truth) for company x_i.
- $|h(x_i) \cap y_i|$ is the count of classifications where the classifier correctly predicted the presence of company x_i (true positives).
- $|h(x_i)|$ is the total count of classes predicted by the classifier for company x_i (used for calculating recall).
- $|y_i|$ is the total count of actual classes for company x_i (used for calculating precision).

This means that each instance-label pair contributes equally to the overall metric, emphasizing the performance on the majority class and providing a more informative picture of the model's ability to generalize across the various labels. In contrast, macro-averaged scores treat each class equally, potentially inflating the performance of the model by giving equal weight to rare classes, which may not be representative of the model's overall performance [38].

What the F1, Recall, and Precision scores miss, however, is the nuanced view of the model's ability to categorize each company with complete accuracy, recognize them to a certain degree (partially correct), or fail entirely (completely incorrect). To capture this granularity, we integrate two additional metrics into our evaluation: the 'Exact Match Ratio' and the 'Partial Match Ratio', which are defined as follows (using the same notation as for the micro-averaged scores):

$$\text{Exact Match Ratio} := \frac{1}{N}\sum_{i=1}^{N}[h(x_i) = y_i]$$

$$\text{Partial Match Ratio} := \frac{1}{N}\sum_{i=1}^{N}[|h(x_i) \cap y_i| > 0]$$

where $[\cdot]$ denotes the Iverson bracket [28], which yields 1 if the proposition inside the bracket is true and 0 otherwise. For any proposition P, the Iverson bracket is defined as:

$$[P] := \begin{cases} 1 & \text{if } P \text{ is true,} \\ 0 & \text{if } P \text{ is false.} \end{cases}$$

The Exact Match Ratio measures the proportion of instances where the predicted set of classes exactly matches the true set, offering insight into the model's performance at a per instance level. Conversely, the Partial Match Ratio quantifies instances where there is at least one class overlap between the predicted and true sets, providing a sense of the model's ability to identify relevant classes, even if it does not capture them all. An overlap of negative classes is not taken into account, otherwise a model that always predicts no class would score very well on this metric which would be counter productive. These additional metrics afford a more comprehensive understanding of our model's performance, given the multi-label classification setting [38] [52]. In summary, these metrics are used:

- Micro-averaged F1-score

- Micro-averaged Recall
- Micro-averaged Precision
- Exact Match Ratio
- Partial Match Ratio

The full evaluation function we used can be seen in appendix C.

4 Datasets

This chapter delves into more detail into how the datasets were curated and processed. It also presents some exemplary excerpts.

4.1 The Georgetown dataset

The Georgetown dataset can be accessed via the Github repository [19]. Without going into too much detail we outline the rough structure of the dataset that we extract from the repository: We take the companies and what they produce, or rather what segment of the semiconductor value chain they are operating in. In total, there are 275 unique companies, of which we were able to match 232 companies with the respective companies in the Orbis dataset which is necessary for the next steps. For instance, for the Commoncrawl step we need to get the company websites from Orbis, as Georgetown does not include this data. And for the Orbis task we obviously need the company description data from Orbis. The matching was done using the Orbis search tool in their user interface. This matching process poses some challenges, mainly entity disambiguation, that is, what company exactly is meant when there is only the name, because there are usually many subsidiaries, different legal entities in different countries etc. [7]. Given that the Georgetown company names were rather vague and general, we attempted to find the main entity. For instance, there would just be the name 'Google' and we would have to match it to, e.g., Google LLC or Alphabet Inc. We did this to the best of our ability but want to highlight that this is a first point of uncertainty and source of error in the process.

Those 232 unique firms would then be found in the 'provision' table of the Georgetown repository [19] where every row represents one firm - provision pair. For example: ('Nvidia' , 'Logic chip design: Discrete GPUs'). Here we have 362 rows that means on average every company is in 1.56 segments. These segments come in two categorizations, one broad and one granular:

the granular one has 79 classes and the broader one five. Here is the head of the 'inputs' table of the repository which details the segments:

input_name	type	stage_name
Logic chip design: AI ASICs	process	Design
Crystal growing furnaces	tool_resource	NaN
Crystal machining tools	tool_resource	NaN
Photolithography	process	Fabrication
Packaging materials	material_resource	NaN

Table 3: Excerpt of the 'inputs' table

The *input_name* column is the granular classification where there are 79 in total. The *type* column is a classification into three types of input, namely: *process*, *material_resource* and *tool_resource*. The process category is then further divided into three stages: *Design, Fabrication, Assembly-Testing and Packaging (ATP)*. If we substitute the 3 stages into the *process* type we get a broader classification of 5 categories: *material_resource, tool_resource, Design, Fabrication and ATP.* There is also a description column which explains each input in more detail but we omitted this column here for clarity but it can be accessed in the repository [19]. Given the limited number of observations in the dataset, we opted to work with this 5 class categorization because the fine-grained categorization has a few drawbacks:

1. Given the sparse population of most classes (avg. instances per class are 5.75), this poses problems for the multi-label classification process [38] [52].

2. Some of these inputs are very specific and even for human annotators would be difficult to classify. Because given the general nature of our predicting text which is mostly general company overview text, we believe that it would be nearly impossible to correctly classify firms into such narrow domains like: *'Tube-based diffusion and deposition tools'* or *'Physical vapor deposition tools'*, just based on the texts we have available. Conversely, recognizing that a company produces mainly tools or specializes in chip design seems feasible.

3. A specific case shows another problem of the fine-grained classification. The company TSMC (Taiwan Semiconductor Manufacturing Company) is one of the most prominent semiconductor companies and

mostly known for chip fabrication focus [14]. In the Georgetown dataset TSMC has three entries: *Advanced Photomasks, ATP* and *Fabrication*. If we were to choose only the fine-grained classification we would get the ground truth that TSMC produces primarily *Advanced Photomasks*, which is hardly what would be predicted, also not by a human expert, since TSMC is best known as one of the leading fabrication firms [14].

For these reasons, the 5 class categorization was chosen. And the following distribution of classes after aggregating it on company level and dropping duplicates is obtained in

class	count
material_resource	105
tool_resource	99
Design	22
Fabrication	13
Assembly, testing, and packaging (ATP)	11

Table 4: Class Distribution of final dataset

We see that there is still a significant class imbalance but since we do not have to train a model with these data, but only need to predict them, this seems to not be a big problem. We further account for this class imbalance in our evaluation process as discussed in section 3.3. Here is a snapshot of the final Georgetown dataset we obtained (only showing essential columns without IDs etc.):

provider_name	class
Cerebras	['Design']
CAMTEK	['tool_resource']
GlobalWafers	['material_resource']
Samsung	['ATP', 'Fabrication', 'material_resource']
Achronix	['Design', 'material_resource']
Renesas	['Fabrication']
Toray	['material_resource']
Kangqiang	['material_resource']
Synopsys	['tool_resource', 'material_resource']
Fujimi	['material_resource']

Table 5: Excerpt of final Georgetown dataset

4.2 The Orbis dataset

In curating the Orbis dataset, the first step, as mentioned above, was to match the Orbis companies to the firms from Georgetown. From these 232 matched companies, we extracted the following columns from the Orbis database. After some experimentation, we found these columns to be the most promising for our task: *main_products_and_services*, *full_overview*, *primary_business_line*, *trade_description_english*. After examining these, we find again some companies do not have any description at all. Thus, we drop all companies that do not have at least one field filled out and are left with 180 (this combination of Orbis columns yielded the highest number of remaining companies, i.e. non-empty descriptions). Afterwards, we concatenated those columns into one *text* column. Here is an abbreviated example of such a concatenated description. The full description can be found in appendix D. This company is classified in Georgetown as *material_resource*.

'main_products_and_services':'Specialty materials for the semiconductor and display industries', 'full_overview':'This company, based in the United States of America, is engaged in the development, manufacture, transportation, and handling of specialty materials for the semiconductor and display industries. It was incorporated in 2016 and has registered headquarters located in Tempe, Arizona. It specializes in chemicals and materials used in semiconductors, as well as specialty gases used in the semiconductor manu-

facturing process, including high purity process materials for deposition, metallization, and chamber cleaning and etching; chemicals mechanical planarization slurries; organosilanes; organometallics and liquid dopants for thin film deposition...

The average length of these descriptions is 1933 characters, the min is 651 and max is 4685. The final Orbis dataset looks like this where the text column would be what was above shown (here only essential columns):

provider_name	text	class_georgetown
Micron	'main_products_an...	['ATP']
Heraeus	'main_products_an...	['material_resource']
Cohu	'main_products_an...	['tool_resource']
Nvidia	'main_products_an...	['Design']
Ceva	'main_products_an...	['material_resource']

Table 6: Excerpt of Orbis dataset

4.3 The Common Crawl dataset

Here the step by step process followed to obtain the Common Crawl dataset is laid out.

4.3.1 Obtaining and analysing the URLs

Since we already linked the Georgetown companies with their website addresses from Orbis, we used those to search the Common Crawl index. The index is a searchable database of metadata from the actual website content. This index search is necessary given that the whole of Common Crawl data size is in the order of petabytes. As mentioned in section 3.1 we use the last crawl of 2021, which has the crawl ID "CC-MAIN-2021-49". As seed URLs we used the company websites we had from Orbis and collected all URLs that start with the seed URLs. For instance the seed URL might be "www.cerebras.net" and then we find urls that start with this like "www.cerebras.net/product-system/" etc.

Following this process, we found 2,143,908 URLs from 206 seeds we start with (a few of the Orbis companies did not have website information). Of those URLs, around 1.3 million where URLs from Google. Thus, we dropped Google and were left with around 800,000 URLs from 168 distinct companies,

for the rest we could not find any entries in the Common Crawl index. In table 4.3.1 are some basic statistics of the URL distribution as number of URLs per company.

Minimum	Maximum	Average	Median	75th percentile
1	138852	4590	66	471

Table 7: Basic statistics of Common Crawl URLs

We further illustrate this distribution in figure 1. There, we plot the complementary cumulative distribution function (CCDF) on a double log scale. We also show that the empirical distribution we observe resembles the theoretical log-normal distribution, except that in the tail the log-normal distribution falls a bit faster than the observed distribution. With the double log scale we also highlight the scale invariability of the distribution. The CDF of the log-normal distribution is given by [37]:

$$F(x;\mu,\sigma) = \frac{1}{2} + \frac{1}{2}\,\mathrm{erf}\left(\frac{\ln x - \mu}{\sigma\sqrt{2}}\right) \tag{1}$$

where:

- x is the value of the random variable,
- μ is the mean of the logarithm of the variable,
- σ is the standard deviation of the logarithm of the variable.
- erf is the error function, defined as [18]:

$$\mathrm{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2}\,dt \tag{2}$$

where:

- t is the variable of integration.
- The integral bounds from 0 to z describe the accumulation of probabilities from the center (mean of the normal distribution) up to z standard deviations away.

The CCDF is then defined as:

$$\bar{F}(x;\mu,\sigma) = 1 - F(x;\mu,\sigma) \tag{3}$$

For our case, the y-axis represents $\bar{F}(x; \mu, \sigma)$, indicating the probability that a randomly sampled company has more than x URLs, where x is the value on the x-axis.

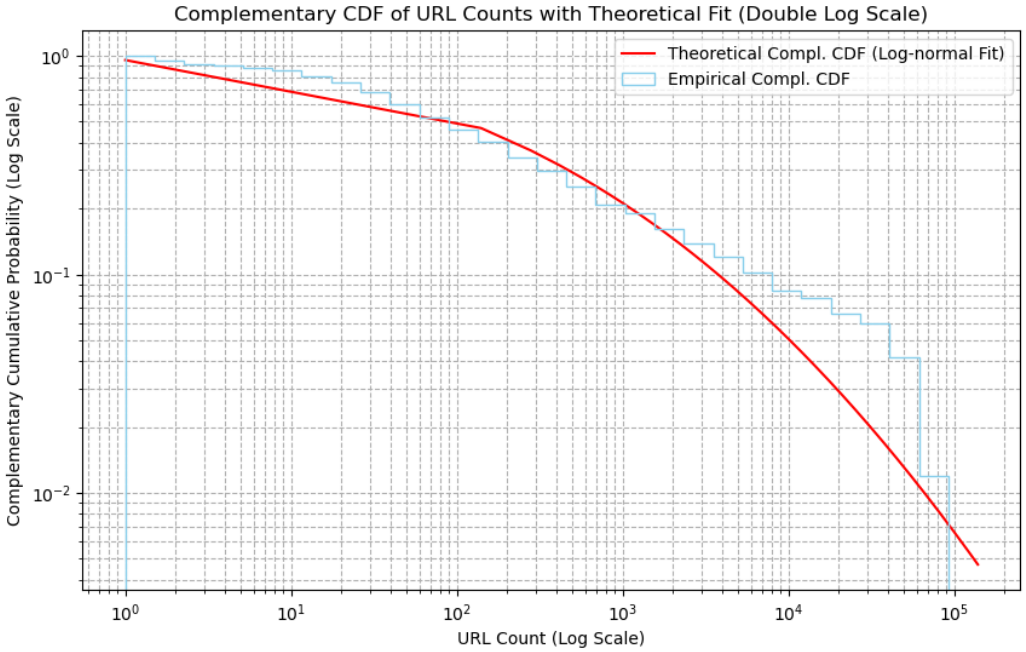

Figure 1: Distribution of Number of URLs per company

Since 800,000 URLs would be too much data to process and also most of it would be irrelevant for our task, we apply some filtering steps.

4.3.2 Filtering the URLs

1. First we deduplicated the URLs. Due to the internal workings of the crawl process it can happen that some URLs are crawled multiple times in the same crawl. Then we are left with 770,000 URLs.

2. Next, we apply a first, rather naive but computationally inexpensive filter. We keep only those URLs that contain one of these keywords that seem promising for the task: 'about', 'service', 'product', 'news', 'semicon', 'technology'. We apply this only for companies that have more than 50 URLs as to not lose smaller companies in this step where filtering is not necessary anyway due to limited number of URLs. After this we are left with 220,000 URLs which is approximately 27.5% of the 800,000 URLs we started with.

3. Next, we check whether the URLs contain the string "/en" which is a convention indicating an English language website. If yes, we only keep those sites that have "/en". We apply this filter only if company has more than 100 URLs and if after the filtering the company has less than 200 URLs left, we revert the "en" filtering. After this step, 200,000 URLs remain.

4. As the next step we conduct a similarity filter. For this we first preprocess the URLs, such as removing special characters and stop words. Then we turn the cleaned URL words into Global Embeddings (GloVe) [46]. Embeddings is a common way of bringing language into a mathematical structure by turning the text into an array of n-dimensional vectors that represent the underlying structure in the language algebraically [46]. Then we create a bag of words (BoW) from the descriptions of the 'inputs' table from the Georgetown dataset, i.e. the descriptions of the semiconductor segments [19]. For this, we conduct similar preprocessing steps as with the URLs before. We assume that this BoW represents the kind of content we want to filter for. Next, we create GloVe embeddings for this BoW as well, so that we can compare the URLs' embeddings with the BoW's. For this we calculate the cosine similarity between the two embeddings, as is common in such NLP cases [67]. The cosine similarity is defined as the cosine of the angle between two vectors, or more rigorously [16]:

$$\text{Cosine similarity}(\mathbf{a}, \mathbf{b}) := \frac{\langle \mathbf{a}, \mathbf{b} \rangle}{\|\mathbf{a}\| \|\mathbf{b}\|} \tag{4}$$

   where:

   - The expression $\langle \mathbf{a}, \mathbf{b} \rangle$ denotes the inner product of vectors $\mathbf{a}$ and $\mathbf{b}$.
   - The vectors $\mathbf{a}$ and $\mathbf{b}$ are elements of an inner product space, which provides the context for defining the inner product and the associated norms.
   - The notation $\|\mathbf{a}\|$ and $\|\mathbf{b}\|$ represents the norms of vectors $\mathbf{a}$ and $\mathbf{b}$ respectively. These norms are derived from the inner product in the same space.

   Because the GloVe embedding gives us an array of vectors, we compute the average of these vectors to represent the text, then we calculate the cosine similarity between these aggregated vectors.

This similarity filter is applied to all companies that have more than 60 URLs and then keep the 60 most similar URLs. After this, there are only 7000 URLs left which is a more manageable number for our resource constraints. It should be noted that the choice of cutoff numbers here (like 60 URLs etc.) is somewhat arbitrary. After some experimentation these numbers just seemed to be reasonable choices for our aims.

4.3.3 Extracting text from the URLs

After slimming down the URLs to a manageable number, we can get the website HTML content from those 7000 URLs from the Common Crawl Web ARChive (WARC) [58] files. To extract the text from the HTML, we use the Python library BeautifulSoup [47]. After this, we filter out those companies that have very short extracted texts as they are usually not useful. Here an example of a dropped text (this is the full concatenated extracted text from all URLs for this firm):

> Unseren Aktionären und Partnern stellen wir nachfolgend alle Geschäfts- und Halbjahresberichte sowie Zwischenmitteilungen der vergangenen Jahre mit allen wichtigen Informationen online zur Verfügung. Dr. Gert Fisahn Zwischenmitteilung zum 30. September 2021 Halbjahresbericht zum 30. Juni 2021 Zwischenmitteilung zum 31. März 2021 zum Archiv Geschäftsbericht 2020 zum Archiv PVA TePla AG Im Westpark 10 - 12 D-35435 Wettenberg Telefon: +49 (0) 641/68690-0 Fax: +49 (0) 641/68690-800 E-Mail: info@ pvatepla.com Web: www.pvatepla.com

Another issue we see here is that some texts are in non-English languages. This aspect is, however, no problem because the LLMs we employ are multi lingual.

Besides the too short texts, there are also very long texts. The longest text we extracted had 118,266,933 characters. However most of this is non usable, here an excerpt:

> 606 0 obj <>/Filter/FlateDecode/ID[<1F51013 0E8824D46BC18 BDC C0A71ED6C>]/Index[586 180] /Info 585 0 R/Length 104/Prev 263573/Root 587 0 R/Size 766/Type/XRef/W[1 2 1]»stream hbbd“b‘$ @H¡I(XAD68$XX/A3j@ Hv(>!$Vf‘bd R8J1Ch endstream endobj startxref 0 %

However, after some manual inspection, we found that those non usable long texts still had usable English text hidden within. For this reason, we applied another round of the similarity filter step on the largest texts to filter out usable and relevant text. In essence, we used the same steps as elaborated on before, merely adjusted for this case. Most notably, we also implemented a chunking step between the cleaning and embedding. After applying this step, the longest extracted text we have is 16,000 characters.

Here is an excerpt of an extracted text:

> mitsui high tec inc products precision tooling precision parts surface grinder motor core company profile company profile history global network corporate sustainability investor relations contact us products development oriented manufacturer working create brighter tomorrow daily lives convenient fulfilling secure ideal situation primarily result rapid technological innovations recent years people feel happiness changes times define evolves well mitsui high tec development oriented manufacturer wields prowess ultra precision machining technology die technology create high quality high precision lead frames motor cores machine tools globally provide value matches needs society create better safer brighter future sincere desire play integral role people rely creating

We see that the text is still in the cleaned format and not in fully fluent English. However, this should not be a big problem, because the Transformer-based LLMs that we use apply these cleaning steps internally in any case. Transformer is an architecture or type of language model [53].

Here is an excerpt of an extracted text that did not go through the similarity search step and thus still has capitalization, stop words etc.:

> ASML is the world's leading provider of lithography systems for the semiconductor industry, manufacturing complex machines critical to the production of integrated circuits or chips. ASML's corporate headquarters is based in the Netherlands with Manufacturing and Development sites located in Asia, Europe and the United States. The Manufacturing site in Wilton Connecticut is responsible for the sourcing, engineering, fabrication, assembly and test of the world's most complex and technologically advanced optical modules and sensors. This position requires access to controlled technology, as defined in the Export Administration Regulations (15 C.F.R. § 730,

et seq.). ...

Here we can observe that this text looks useful for our classification task, and it may be objected that the similarity filter may worsen the quality of the text. This is, however, not the case because the similarity search step functions mostly as a rescue operation for an extracted text that is mostly completely not usable and full of irrelevant text. Therefore, compared to that, the similarity step is still reasonable.

After all these steps, we lost quite a lot of companies because we were not able to extract useful text from their website with our method or they had none. In the end, we are left with 99 companies with a mean of 4000 characters of description text.

Furthermore, want to note that this filtering process is far from ideal and that the extracted text is not of the same quality as the Orbis text. However, after much experimentation this is still the best we could achieve. But as we will elaborate on in the discussion (6), we see this as a major opportunity for improvement for future research but found it to be out of scope of this study to delve even deeper into this problem.

5 Results

In this section the results obtained with the different models and on the two datasets are presented. As is typical in evaluating LLMs, their performance is compared on each dataset separately [12].

5.1 Orbis

The performance evaluation of various models on the Orbis dataset reveals insightful distinctions. The summary of our results on this dataset can be seen in Table 5.1.

To start with, GPT-4 demonstrates superior precision (0.51) and recall (0.74). This performance is particularly noteworthy in the F1 score, where GPT-4 achieves the highest score of the models, 0.61, indicating a balanced trade-off between precision and recall (as the F1 is the harmonic mean between the two). Similarly it achieved the highest Partial Match Ratio of 0.78, indicating that it predicted at least one class correctly in 78% of the cases. The only metric on which GPT-4 is outperformed by Mistral and GPT-3.5 is in the Exact Match Ratio. This is interesting since one would expect that GPT-4 is always better than GPT-3.5, which does not seem to be the case. However, here it may be necessary to apply further statistical tests to get an

understanding of the statistical significance of minor differences, given the non-deterministic behaviour of LLMs [4].

Surprisingly, GoLLIE [48] receives the lowest scores on all measures except the Partial Match Ratio and Recall. However, this may be due to the prompt setup we chose. In order to make it more comparable, we used the same prompt and setup as outlined in section 3.2.2 for all Hugging Face models. While this works, it may lead to suboptimal results for models that require a more specialized workflow. For GoLLIE, for instance, the authors provide a Github repository [26] to access their models in this specialized way. There they have Jupyter notebooks that explain the necessary setup for different IE tasks such as NER or EE. They also provide a template for custom tasks. We tried to use this template and adapt it for our classification problem but did not get superior results compared to the standard approach we used for the other Hugging Face models. As we understand it after this experimentation, GoLLIE is specialized to extract explicit information from text, but not conduct any further reasoning or more abstract analyses. However, this additional step is crucial for our task as the value chain segments are not explicitly mentioned in the text, but rather have to be inferred from the context. We expected GoLLIE to combine the specialization for finding explicit information with an ability to perform inference. We suspect that using a larger GoLLIE model might remedy this and achieve the expected combination because the additional parameters allow the more general reasoning necessary for the task. But because the authors themselves claimed that GoLLIE 7B is not inferior to larger versions and even outperforms them in some cases [48], we chose the 7B model, also for resource constraints.

A similar reason might have affected the lower performance of CatPPT. However, there the authors do not give detailed instructions of any special implementation necessary for the model [2] and since it has the same architecture as Mistral, we opted to use the same setup, also for fair comparison. Nevertheless it achieves comparable measures to GPT-3.5 and Mistral, albeit slightly lower.

Mistral performed rather well, especially compared to GPT-3.5 and GPT-4 considering that those are much larger models. We do not know the exact number of parameters, but GPT-3 had 175B [9] and it is very likely that the further models have considerably more. The Mistral we used, on the other hand, has just 7B. For instance, looking at the Exact Match Ratio it had approximately the same score as GPT-4, 68 and 67, respectively.

Given these observations, our analysis suggests that GPT-3.5, GPT-4 and Mistral performed the best on the Orbis dataset. This is somewhat surprising because we expected the more specialized models CatPPT and GoLLIE to perform better, given the specialized nature of the task at hand. However,

the results seem to suggest that the task required more general reasoning and abstraction capabilities, which larger models like GPT-4 arguably have [10].

Metric	Mistral	CatPPT	GoLLIE	GPT-3.5	GPT-4
Precision	0.43	0.32	0.26	0.46	0.51
Recall	0.65	0.63	0.57	0.44	0.74
F1 Score	0.51	0.43	0.35	0.45	0.61
Exact Matches	68	56	9	74	67
Exact Match Ratio	0.38	0.31	0.05	0.41	0.37
Partial Matches	123	116	104	86	141
Partial Match Ratio	0.68	0.64	0.58	0.48	0.78
Nr. Firms	180	180	180	180	180

Table 8: Model Evaluation Metrics on Orbis

5.2 Common Crawl

The analysis of model performance on the Common Crawl dataset provides another dimension to our understanding and potentially reveals the model's respective strengths and limitations in a broader context by adding another benchmark. The summary Table 5.2 can again be found below. It has to be further noted here, that we did not run the GPT-4 model on this dataset. That is due to resource constraints, since a run with GPT-4 would have been financially intensive given that the Common Crawl dataset has longer input text. Even so, the general trend of how Orbis and Common Crawl differ on this task is still pertinent.

The first observation is that the scores on the metrics are generally lower than on the Orbis datset. Further it has to be mentioned, that there are fewer companies in this dataset, due to the lengthy filtering process the Common Crawl data preparation necessitated. The inspection of the datasets (which was in more detail conducted in section 4) seems to suggest a relatively straight-forward interpretation of the lower scores achieved on the Common Crawl and that is that the text was simply less suitable for our task than the Orbis text.

When looking at the differences in performance among the models we find a striking reordering of what models perform best. While CatPPT was among the least performing models in the Orbis dataset, it achieved the highest scores on Recall (0.48), F1 Score (0.33) and Partial Match Ratio (0.49) on the Common Crawl dataset. GPT-3.5 stands out with the highest

Precision (0.28) and Exact Match Ratio (0.22) among the compared models. However, it falls behind in Recall (0.26), indicating a potential limitation in retrieving pertinent information across the dataset. Furthermore, GPT-3.5 has a remarkably low Partial Match Ratio (0.29), the lowest in fact.

Mistral cannot transfer its strong performance and achieves only mid-tier results on this dataset, with an F1 Score of 0.3. This is still better than GoLLIE which has an F1 Score of 0.28 and a remarkably low Exact Match Ratio of 0.01. In general, on the Common Crawl dataset the gaps between models are not as pronounced as in the Orbis dataset, as all models perform rather badly. This could be due to the overall increased difficulty of this dataset.

Metric	Mistral	CatPPT	GoLLIE	GPT-3.5
Precision	0.25	0.26	0.21	0.28
Recall	0.38	0.48	0.45	0.26
F1 Score	0.30	0.33	0.28	0.27
Exact Matches	11	14	1	22
Exact Match Ratio	0.11	0.14	0.01	0.22
Partial Matches	40	49	47	29
Partial Match Ratio	0.40	0.49	0.47	0.29
Nr. Firms	99	99	99	99

Table 9: Model Evaluation Metrics on Common Crawl

6 Discussion

6.1 Limitations

A significant limitation encountered in our study pertains to the process of extracting the relevant text from Common Crawl. Identifying relevant URLs and effectively extracting text from these sources is critical, as the quality of the model predictions is obviously heavily dependent on the quality of the input text. In some instances, the extracted text was so ambiguous or contextually sparse that even domain experts could face difficulties in classifying the company based on the provided text alone.

Another noteworthy limitation is the methodological approach of classifying companies into discrete value chain steps. This approach introduces a simplification, representing a deviation from the complex and nuanced reality where such categorizations are inherently spectrum-based rather than binary. Moreover, these classifications, devised by human experts from [11],

are not absolute truths but constructed frameworks that aim to approximate real-world phenomena. The subjectivity inherent in these constructs is a known challenge within supervised machine learning and IE [65]. Accurately generating consistent labels is fraught with difficulties, as definitions can be ambiguous and interpretations may vary among experts. This variability in agreement on labels is a concern also reflected in the work of [60], where discrepancies among labelers were notably frequent.

Resource limitations also constituted a constraint on this study. Specifically, our computational resources restricted us to the utilization of 7B models, precluding the employment of more powerful models due to our GPU's limitations. Additionally, the financial implications of leveraging advanced computational technologies like GPT-4 posed another barrier, constraining the extent and depth of our analysis. These limitations undoubtedly influenced the overall robustness and generalizability of our research findings. However, they also highlight opportunities of where future research can achieve some interesting results by simply employing more resources.

6.2 Future Research

From our research, the insights we gained and challenges we faced, we see some promising directions for future endeavours:

Improving the extraction procedure from Common Crawl and enhancing the overall quality of input text represents a promising area of future exploration. Specifically, employing a Retrieval Augmented Generation (RAG) pipeline could offer significant advancements. A RAG pipeline is a setup where a model interacts dynamically with a database of external data and incorporates this data into its responses[30]. What would probably be the most beneficial aspect of this, would be to implement additional feedback loops that check whether the extracted data is useful for the given task or not and potentially repeating the process multiple times iteratively. While we already implemented parts of such a pipeline, namely the similarity search, there is probably still some potential for more sophisticated pipelines.

Beyond enhancing RAG, investigating LLM agents endowed with increased feedback mechanisms presents another intriguing research avenue. LLM agents are a way of chaining multiple LLM prompts and responses together in such a way that a model can prompt itself [64]. Projects like AutoGPT [50] aspire to building an LLM agent that can accomplish a broad variety of tasks autonomously. Regarding the topic of this thesis, we envision a scenario where an LLM agent sifts through Common Crawl text, autonomously determining the utility of the retrieved content. Although this methodology demands enhanced computational resources, it holds po-

tential for groundbreaking insights. Exploring this pathway could unveil how these augmented feedback loops impact LLM's efficacy and adaptability in real-world applications. This approach would also most resemble the way that human experts conduct research.

Advancing to larger-scale models offers another promising research trajectory. While Sainz et al. [48] argue with GoLLIE that larger models do not uniformly translate to superior performance in narrow IE tasks, our observations suggest an addition to this remark. Namely, that for complex tasks necessitating heightened reasoning and abstraction, bigger models seem to retain their superiority. Consequently, reiterating our study with future, more evolved LLMs could provide valuable insights into the progressive capabilities of these models and their utility for practical real-world tasks. This iterative evaluation would also help chart the evolution of LLMs [12].

7 Conclusion

In this thesis we explored how LLMs can extract supply chain information from textual data. For this, we curated two custom datasets. One company related text from the Orbis database and one from firm website text using Common Crawl [15]. As a ground truth we used the classification of semiconductor firms into value chain segments devised by [11], thereby turning the task into a hybrid between Information Extraction (IE) and a multi label classification problem. On these two datasets we tested different models, namely: GPT-3.5 [9], GPT-4 [43], Mistral [29], CatPPT [2] and GoLLIE [48].

What we found, returning to the research question, is that yes LLMs can accurately extract supply chain information from text, but with some caveats. First, it is not perfect. For instance our highest scoring model, GPT-4, achieved an F1 Score of 0.61 on the Orbis dataset. But this may be due to the inherent difficulty of classifying companies with limited context. Regarding the subquestions of how different models perform and of how different input text affects these performances, we find that the input text has a greater impact than the models. While there are differences among the models, there was always a general trend dictated by the datasets, that was more pronounced than the model differences. We found that the Orbis dataset led to better predictions than the Common Crawl dataset. We did not have the case that one model outperformed all models on all metrics. However, among the strong models across datasets were the GPT models from OpenAI and the Mistral model. We found Mistral's strong performance noteworthy given its limited size of 7B compared to the much larger GPT models. Moreover, we found that the more specialized models CatPPT and GoLLIE did not have

a particular advantage. On the contrary, their performance lagged slightly behind the more general models. This suggests that our task, even though rather narrowly defined, still demands a certain level of general reasoning capability which larger and general purpose models typically exhibit [66].

One of the largest roadblocks we faced was to extract the relevant text from the websites with common crawl. This either suggests that the approach is inherently limited and company websites do not lend themselves well to predicting the supply chain segment of a company. On the other hand, it more likely indicates that there is still an opportunity to improve the text extraction method. As discussed in section 6.2, two interesting options could be implementing a more sophisticated RAG pipeline or employing LLM agents.

Overall, our study adds to the vast landscape of LLM applications[66] and we can imagine that such applications of AI will be very disruptive also in practical industry settings. For instance, Versed.ai [1] is a Supply Chain analysis company that uses primarily AI methods similar to those explored in this thesis. The company spun out of the study of Wichman et al. that we mentioned throughout this work [60]. Examples like this highlight the potential and utility of advanced AI applications for industry use today and especially in years to come.

Bibliography

[1] *A true reflection of your supply chain with Versed AI*. URL: https://www.versed.ai/ (visited on 2024-03-11).

[2] Rishiraj Acharya. *CatPPT*. Publication Title: Hugging Face repository. 2023. URL: https://huggingface.co/rishiraj/CatPPT.

[3] Amazon. *Amazon Mechanical Turk*. URL: https://www.mturk.com/ (visited on 2024-03-19).

[4] Blaise Agüera y Arcas. "Do Large Language Models Understand Us?" In: *Daedalus* 151 (2022), pp. 183–197. DOI: 10.1162/daed_a_01909. URL: https://consensus.app/papers/language-models-understand-arcas/f6de6bb2928b537c81f5078017b9f0dc/ (visited on 2024-03-11).

[5] Dhananjay Ashok and Zachary C. Lipton. *PromptNER: Prompting For Named Entity Recognition*. June 20, 2023. DOI: 10.48550/arXiv.2305.15444. arXiv: 2305.15444[cs]. URL: http://arxiv.org/abs/2305.15444 (visited on 2024-03-03).

[6] Andrea Bacilieri and Pablo Austudillo-Estevez. *Reconstructing firm-level input-output networks from partial information*. Mar. 31, 2023. arXiv: 2304.00081[physics,q-fin]. URL: http://arxiv.org/abs/2304.00081 (visited on 2023-09-26).

[7] Nils Barlaug and Jon Atle Gulla. "Neural Networks for Entity Matching: A Survey". In: *ACM Transactions on Knowledge Discovery from Data* 15.3 (June 30, 2021), pp. 1–37. ISSN: 1556-4681, 1556-472X. DOI: 10.1145/3442200. arXiv: 2010.11075[cs]. URL: http://arxiv.org/abs/2010.11075 (visited on 2024-03-19).

[8] Zhen Bi, Jing Chen, Yinuo Jiang, Feiyu Xiong, Wei Guo, Huajun Chen, and Ningyu Zhang. *CodeKGC: Code Language Model for Generative Knowledge Graph Construction*. Jan. 18, 2024. DOI: 10.48550/arXiv.2304.09048. arXiv: 2304.09048[cs]. URL: http://arxiv.org/abs/2304.09048 (visited on 2024-03-03).

[9] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. *Language Models are Few-Shot Learners*. July 22,

2020. DOI: 10.48550/arXiv.2005.14165. arXiv: 2005.14165[cs]. URL: http://arxiv.org/abs/2005.14165 (visited on 2024-03-09).

[10] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, and Yi Zhang. *Sparks of Artificial General Intelligence: Early experiments with GPT-4*. Apr. 13, 2023. arXiv: 2303.12712[cs]. URL: http://arxiv.org/abs/2303.12712 (visited on 2024-03-11).

[11] Center for Security and Emerging Technology, Saif M. Khan, Alexander Mann, and Dahlia Peterson. *The Semiconductor Supply Chain: Assessing National Competitiveness*. Center for Security and Emerging Technology, Jan. 2021. DOI: 10.51593/20190016. URL: https://cset.georgetown.edu/publication/the-semiconductor-supply-chain/ (visited on 2023-09-28).

[12] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, Wei Ye, Yue Zhang, Yi Chang, Philip S. Yu, Qiang Yang, and Xing Xie. "A Survey on Evaluation of Large Language Models". In: *ACM Transactions on Intelligent Systems and Technology* (Jan. 23, 2024). Just Accepted. ISSN: 2157-6904. DOI: 10.1145/3641289. URL: https://dl.acm.org/doi/10.1145/3641289 (visited on 2024-03-10).

[13] Xiang Chen, Lei Li, Shuofei Qiao, Ningyu Zhang, Chuanqi Tan, Yong Jiang, Fei Huang, and Huajun Chen. *One Model for All Domains: Collaborative Domain-Prefix Tuning for Cross-Domain NER*. Sept. 18, 2023. DOI: 10.48550/arXiv.2301.10410. arXiv: 2301.10410[cs]. URL: http://arxiv.org/abs/2301.10410 (visited on 2024-03-03).

[14] Min-Hua Chiang. "Taiwan Semiconductor Manufacturing Company: A Key Chip in the Global Political Economy". In: *East Asian Policy* 15.1 (Jan. 2023). Publisher: World Scientific Publishing Co., pp. 36–46. ISSN: 1793-9305. DOI: 10.1142/S179393052300003X. URL: https://www.worldscientific.com/doi/10.1142/S179393052300003X (visited on 2024-03-10).

[15] *Common Crawl - Open Repository of Web Crawl Data*. URL: https://commoncrawl.org/ (visited on 2024-03-09).

[16] *Cosine similarity*. In: *Wikipedia*. Page Version ID: 1207802784. Feb. 15, 2024. URL: https://en.wikipedia.org/w/index.php?title=Cosine_similarity&oldid=1207802784 (visited on 2024-03-10).

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. May 24, 2019. arXiv: 1810.04805[cs]. URL: http://arxiv.org/abs/1810.04805 (visited on 2024-03-03).

[18] *Error function*. In: *Wikipedia*. Page Version ID: 1191398190. Dec. 23, 2023. URL: https://en.wikipedia.org/w/index.php?title=Error_function&oldid=1191398190#Complementary_error_function (visited on 2024-03-10).

[19] *eto-supply-chain/data at main · georgetown-cset/eto-supply-chain*. GitHub. URL: https://github.com/georgetown-cset/eto-supply-chain/tree/main/data (visited on 2023-09-28).

[20] *EU AI Act: first regulation on artificial intelligence*. Topics | European Parliament. June 8, 2023. URL: https://www.europarl.europa.eu/topics/en/article/20230601STO93804/eu-ai-act-first-regulation-on-artificial-intelligence (visited on 2024-03-11).

[21] *FactSet Supply Chain Relationships | FactSet*. URL: https://go.factset.com/marketplace/catalog/product/factset-supply-chain-relationships (visited on 2023-09-30).

[22] Lizhou Fan, Lingyao Li, Zihui Ma, Sanggyu Lee, Huizi Yu, and Libby Hemphill. *A Bibliometric Review of Large Language Models Research from 2017 to 2023*. Apr. 3, 2023. DOI: 10.48550/arXiv.2304.02020. arXiv: 2304.02020[cs]. URL: http://arxiv.org/abs/2304.02020 (visited on 2024-03-10).

[23] Chengguang Gan, Qinghao Zhang, and Tatsunori Mori. *GIELLM: Japanese General Information Extraction Large Language Model Utilizing Mutual Reinforcement Effect*. Nov. 12, 2023. DOI: 10.48550/arXiv.2311.06838. arXiv: 2311.06838[cs]. URL: http://arxiv.org/abs/2311.06838 (visited on 2024-03-03).

[24] *Global Supply Chain Data*. URL: https://www.bloomberg.com/professional/dataset/global-supply-chain-data/ (visited on 2023-09-30).

[25] Achintya Gopal and Chunho Chang. *Discovering Supply Chain Links with Augmented Intelligence*. Nov. 2, 2021. DOI: 10.48550/arXiv.2111.01878. arXiv: 2111.01878[cs]. URL: http://arxiv.org/abs/2111.01878 (visited on 2023-09-28).

[26] *hitz-zentroa/GoLLIE*. original-date: 2023-10-05T15:38:20Z. Feb. 27, 2024. URL: https://github.com/hitz-zentroa/GoLLIE (visited on 2024-03-01).

[27] *Hugging Face Hub documentation*. URL: https://huggingface.co/docs/hub/index (visited on 2024-03-09).

[28] *Indicator function*. In: *Wikipedia*. Page Version ID: 1187362170. Nov. 28, 2023. URL: https://en.wikipedia.org/w/index.php?title=Indicator_function&oldid=1187362170 (visited on 2024-03-20).

[29] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. *Mistral 7B*. Oct. 10, 2023. DOI: 10.48550/arXiv.2310.06825. arXiv: 2310.06825[cs]. URL: http://arxiv.org/abs/2310.06825 (visited on 2024-02-15).

[30] YuHe Ke, Liyuan Jin, Kabilan Elangovan, Hairil Rizal Abdullah, Nan Liu, Alex Tiong Heng Sia, Chai Rick Soh, Joshua Yi Min Tung, Jasmine Chiat Ling Ong, and Daniel Shu Wei Ting. *Development and Testing of Retrieval Augmented Generation in Large Language Models – A Case Study Report*. Jan. 29, 2024. DOI: 10.48550/arXiv.2402.01733. arXiv: 2402.01733[cs]. URL: http://arxiv.org/abs/2402.01733 (visited on 2024-02-27).

[31] Edward Elson Kosasih and Alexandra Brintrup. "A machine learning approach for predicting hidden links in supply chain with graph neural networks". In: *International Journal of Production Research* 60.17 (Sept. 2, 2022). Publisher: Taylor & Francis, pp. 5380–5393. ISSN: 0020-7543. DOI: 10.1080/00207543.2021.1956697. URL: https://doi.org/10.1080/00207543.2021.1956697 (visited on 2023-09-28).

[32] Mingchen Li and Rui Zhang. *How far is Language Model from 100% Few-shot Named Entity Recognition in Medical Domain*. June 30, 2023. DOI: 10.48550/arXiv.2307.00186. arXiv: 2307.00186[cs]. URL: http://arxiv.org/abs/2307.00186 (visited on 2024-03-03).

[33] Peng Li, Tianxiang Sun, Qiong Tang, Hang Yan, Yuanbin Wu, Xuanjing Huang, and Xipeng Qiu. *CodeIE: Large Code Generation Models are Better Few-Shot Information Extractors*. May 10, 2023. DOI: 10.48550/arXiv.2305.05711. arXiv: 2305.05711[cs]. URL: http://arxiv.org/abs/2305.05711 (visited on 2024-03-03).

[34] Chengyuan Liu, Fubang Zhao, Yangyang Kang, Jingyuan Zhang, Xiang Zhou, Changlong Sun, Kun Kuang, and Fei Wu. "RexUIE: A Recursive Method with Explicit Schema Instructor for Universal Information

Extraction". In: *Findings of the Association for Computational Linguistics: EMNLP 2023*. Findings 2023. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 15342–15359. DOI: 10.18653/v1/2023.findings-emnlp.1024. URL: https://aclanthology.org/2023.findings-emnlp.1024 (visited on 2024-03-03).

[35] Yushan Liu, Bailan He, Marcel Hildebrandt, Maximilian Buchner, Daniela Inzko, Roger Wernert, Emanuel Weigel, Dagmar Beyer, Martin Berbalk, and Volker Tresp. *A Knowledge Graph Perspective on Supply Chain Resilience*. May 15, 2023. arXiv: 2305.08506[cs]. URL: http://arxiv.org/abs/2305.08506 (visited on 2023-09-28).

[36] *LLM-Perf Leaderboard - a Hugging Face Space by optimum*. URL: https://huggingface.co/spaces/optimum/llm-perf-leaderboard (visited on 2024-03-09).

[37] *Log-normal distribution*. In: *Wikipedia*. Page Version ID: 1210518171. Feb. 27, 2024. URL: https://en.wikipedia.org/w/index.php?title=Log-normal_distribution&oldid=1210518171 (visited on 2024-03-10).

[38] Gjorgji Madjarov, Dragi Kocev, Dejan Gjorgjevikj, and Sašo Džeroski. "An extensive experimental comparison of methods for multi-label learning". In: *Pattern Recognition* 45 (Sept. 1, 2012), pp. 3084–3104. DOI: 10.1016/j.patcog.2012.03.004.

[39] Luca Mungo, François Lafond, Pablo Astudillo-Estévez, and J. Doyne Farmer. "Reconstructing production networks using machine learning". In: *Journal of Economic Dynamics and Control* 148 (Mar. 2023), p. 104607. ISSN: 01651889. DOI: 10.1016/j.jedc.2023.104607. URL: https://linkinghub.elsevier.com/retrieve/pii/S0165188923000131 (visited on 2023-09-26).

[40] Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, N. Barnes, and A. Mian. "A Comprehensive Overview of Large Language Models". In: *ArXiv* abs/2307.06435 (2023). DOI: 10.48550/arXiv.2307.06435. URL: https://consensus.app/papers/comprehensive-overview-large-language-models-naveed/fa427c8e20b959f79c1d742f152c4f85/ (visited on 2024-03-10).

[41] *Open LLM Leaderboard - a Hugging Face Space by HuggingFaceH4*. URL: https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard (visited on 2024-03-11).

[42] OpenAI. *Function calling and other API updates*. URL: https://openai.com/blog/function-calling-and-other-api-updates (visited on 2024-03-09).

[43] OpenAI et al. *GPT-4 Technical Report*. Dec. 18, 2023. DOI: 10.48550/arXiv.2303.08774. arXiv: 2303.08774[cs]. URL: http://arxiv.org/abs/2303.08774 (visited on 2024-02-15).

[44] *Orbis - BVD is now Moody's*. URL: https://www.moodys.com/web/en/us/capabilities/company-reference-data/orbis.html (visited on 2024-02-15).

[45] Arka Pal, Deep Karkhanis, Samuel Dooley, Manley Roberts, Siddartha Naidu, and Colin White. "Smaug: Fixing Failure Modes of Preference Optimisation with DPO-Positive". In: *arXiv preprint arXiv:2402.13228* (2024).

[46] Jeffrey Pennington, Richard Socher, and Christopher Manning. "Glove: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). Doha, Qatar: Association for Computational Linguistics, 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: http://aclweb.org/anthology/D14-1162 (visited on 2024-03-10).

[47] Leonard Richardson. *Beautiful Soup*. URL: https://www.crummy.com/software/BeautifulSoup/ (visited on 2024-03-10).

[48] Oscar Sainz, Iker García-Ferrero, Rodrigo Agerri, Oier Lopez de Lacalle, German Rigau, and Eneko Agirre. *GoLLIE: Annotation Guidelines improve Zero-Shot Information-Extraction*. Feb. 21, 2024. arXiv: 2310.03668[cs]. URL: http://arxiv.org/abs/2310.03668 (visited on 2024-03-01).

[49] Henning Schöpper and Wolfgang Kersten. "Using natural language processing for supply chain mapping: a systematic review of current approaches". In: 5th International Conference on Computational Linguistics and Intelligent Systems (COLINS 2021). ISSN: 1613-0073 Issue: 5. RWTH Aachen, May 26, 2021, pp. 71–86. DOI: 10.15480/882.3589. URL: http://hdl.handle.net/11420/9687 (visited on 2023-09-26).

[50] Significant Gravitas. *AutoGPT*. original-date: 2023-03-16T09:21:07Z. Mar. 11, 2024. URL: https://github.com/Significant-Gravitas/AutoGPT (visited on 2024-03-11).

[51] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. July 19, 2023. DOI: 10.48550/arXiv.2307.09288. arXiv: 2307.09288[cs]. URL: http://arxiv.org/abs/2307.09288 (visited on 2024-02-15).

[52] Grigorios Tsoumakas and Ioannis Katakis. "Multi-Label Classification: An Overview". In: *International Journal of Data Warehousing and Mining* 3 (Sept. 1, 2009), pp. 1–13. DOI: 10.4018/jdwm.2007070101.

[53] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c
Abstract.html (visited on 2024-03-10).

[54] Zhen Wan, Fei Cheng, Zhuoyuan Mao, Qianying Liu, Haiyue Song, Jiwei Li, and Sadao Kurohashi. *GPT-RE: In-context Learning for Relation Extraction using Large Language Models*. Dec. 8, 2023. DOI: 10.48550/arXiv.2305.02105. arXiv: 2305.02105[cs]. URL: http://arxiv.org/abs/2305.02105 (visited on 2024-03-03).

[55] Chenguang Wang, Xiao Liu, Zui Chen, Haoyun Hong, Jie Tang, and Dawn Song. "DeepStruct: Pretraining of Language Models for Structure Prediction". In: *Findings of the Association for Computational Linguistics: ACL 2022*. Findings of the Association for Computational Linguistics: ACL 2022. Dublin, Ireland: Association for Computational Linguistics, 2022, pp. 803–823. DOI: 10.18653/v1/2022.findings-

acl.67. URL: https://aclanthology.org/2022.findings-acl.67 (visited on 2024-03-03).

[56] Shuhe Wang, Xiaofei Sun, Xiaoya Li, Rongbin Ouyang, Fei Wu, Tianwei Zhang, Jiwei Li, and Guoyin Wang. *GPT-NER: Named Entity Recognition via Large Language Models*. Oct. 7, 2023. DOI: 10.48550/arXiv.2304.10428. arXiv: 2304.10428[cs]. URL: http://arxiv.org/abs/2304.10428 (visited on 2024-03-03).

[57] Xiao Wang, Weikang Zhou, Can Zu, Han Xia, Tianze Chen, Yuansen Zhang, Rui Zheng, Junjie Ye, Qi Zhang, Tao Gui, Jihua Kang, Jingsheng Yang, Siyuan Li, and Chunsai Du. *InstructUIE: Multi-task Instruction Tuning for Unified Information Extraction*. Apr. 17, 2023. DOI: 10.48550/arXiv.2304.08085. arXiv: 2304.08085[cs]. URL: http://arxiv.org/abs/2304.08085 (visited on 2024-03-03).

[58] *Web ARChive*. In: *Wikipedia*. Page Version ID: 235764211. July 24, 2023. URL: https://de.wikipedia.org/w/index.php?title=Web_ARChive&oldid=235764211 (visited on 2024-03-10).

[59] Xiang Wei, Xingyu Cui, Ning Cheng, Xiaobin Wang, Xin Zhang, Shen Huang, Pengjun Xie, Jinan Xu, Yufeng Chen, Meishan Zhang, Yong Jiang, and Wenjuan Han. *Zero-Shot Information Extraction via Chatting with ChatGPT*. Feb. 20, 2023. DOI: 10.48550/arXiv.2302.10205. arXiv: 2302.10205[cs]. URL: http://arxiv.org/abs/2302.10205 (visited on 2024-03-01).

[60] Pascal Wichmann, Alexandra Brintrup, Simon Baker, Philip Woodall, and Duncan McFarlane. "Extracting supply chain maps from news articles using deep neural networks". In: *International Journal of Production Research* 58.17 (Sept. 1, 2020). Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/00207543.2020.1720925, pp. 5320–5336. ISSN: 0020-7543. DOI: 10.1080/00207543.2020.1720925. URL: https://doi.org/10.1080/00207543.2020.1720925 (visited on 2023-09-26).

[61] *WU (Wirtschaftsuniversität Wien)*. URL: https://www.wu.ac.at/bibliothek/recherche/datenbanken/info/orbis/ (visited on 2024-03-09).

[62] Derong Xu, Wei Chen, Wenjun Peng, Chao Zhang, Tong Xu, Xiangyu Zhao, Xian Wu, Yefeng Zheng, and Enhong Chen. *Large Language Models for Generative Information Extraction: A Survey*. Dec. 29, 2023. DOI: 10.48550/arXiv.2312.17617. arXiv: 2312.17617[cs]. URL: http://arxiv.org/abs/2312.17617 (visited on 2024-02-28).

[63] Hongshen Xu, Lu Chen, Zihan Zhao, Da Ma, Ruisheng Cao, Zichen Zhu, and Kai Yu. *Hierarchical Multimodal Pre-training for Visually Rich Webpage Understanding*. Feb. 28, 2024. arXiv: 2402.18262[cs]. URL: http://arxiv.org/abs/2402.18262 (visited on 2024-03-03).

[64] Hui Yang, Sifu Yue, and Yunzhong He. *Auto-GPT for Online Decision Making: Benchmarks and Additional Opinions*. June 3, 2023. arXiv: 2306.02224[cs]. URL: http://arxiv.org/abs/2306.02224 (visited on 2024-03-11).

[65] Min-Ling Zhang and Zhi-Hua Zhou. "A Review on Multi-Label Learning Algorithms". In: *IEEE Transactions on Knowledge and Data Engineering* 26.8 (Aug. 2014), pp. 1819–1837. ISSN: 1041-4347. DOI: 10.1109/TKDE.2013.39. URL: http://ieeexplore.ieee.org/document/6471714/ (visited on 2024-03-11).

[66] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. *A Survey of Large Language Models*. Nov. 24, 2023. DOI: 10.48550/arXiv.2303.18223. arXiv: 2303.18223[cs]. URL: http://arxiv.org/abs/2303.18223 (visited on 2024-03-10).

[67] Kaitlyn Zhou, Kawin Ethayarajh, Dallas Card, and Dan Jurafsky. *Problems with Cosine as a Measure of Embedding Similarity for High Frequency Words*. May 10, 2022. DOI: 10.48550/arXiv.2205.05092. arXiv: 2205.05092[cs]. URL: http://arxiv.org/abs/2205.05092 (visited on 2024-03-10).

[68] Wenxuan Zhou, Sheng Zhang, Yu Gu, Muhao Chen, and Hoifung Poon. *UniversalNER: Targeted Distillation from Large Language Models for Open Named Entity Recognition*. Jan. 18, 2024. DOI: 10.48550/arXiv.2308.03279. arXiv: 2308.03279[cs]. URL: http://arxiv.org/abs/2308.03279 (visited on 2024-03-03).

A Implementation for Hugging Face models

```python
import torch
from transformers import AutoModelForCausalLM, AutoTokenizer
import pickle
import pandas as pd
import argparse
import logging
import os
from dotenv import load_dotenv

### Load environment variables from the .env file ###
def load_environment_variables():
    load_dotenv()

### Setup logging ###
def setup_logging():
    logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')

### Command line arguments ###
def parse_arguments():
    parser = argparse.ArgumentParser(description="Run the model with custom input file and output filename.")
    parser.add_argument('-input_filepath', type=str, required=True, help="Input file path, needs to be a pickle dataframe with columns 'text' for prediction and 'class' for the true class of the firm.")
    parser.add_argument('-output_filename', type=str, required=True, help="Output filename as it will be saved to OUTPUT_DIR.")
    parser.add_argument('-model', type=str, required=True, help="Exact Hugging Face model name, e.g., 'mistralai/Mistral-7B-Instruct-v0.2'.")
    return parser.parse_args()

### Model and tokenizer setup ###
def initialize_model_and_tokenizer(model_name):
    device = "cuda:0" if torch.cuda.is_available() else "cpu"
    model = AutoModelForCausalLM.from_pretrained(model_name)
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    model.to(device)
    return model, tokenizer, device

### Call the model ###
def call_model(model, tokenizer, device, description, categories):
    """
    Generate a response from the model based on the provided
```

```
    description and categories.
38
39      This function constructs a prompt using the given
    description and categories, sends it to the model,
40      and processes the model's output to extract and return
    the relevant response.
41
42      Args:
43      - model (PreTrainedModel): The loaded model object from
    Hugging Face's transformers.
44      - tokenizer (PreTrainedTokenizer): The tokenizer
    corresponding to the model, used for encoding the input
    text.
45      - device (str): The device type (e.g., 'cuda:0', 'cpu')
    indicating where the model is loaded.
46      - description (str): The text description from a company
    to be analyzed by the model.
47      - categories (list): The list of categories which should
    be predicted.
48
49      Returns:
50      - list: A list of strings, each representing a category
    identified from the model's response to the prompt.
51      """
52
53      prompt = f"""Here is a text from a semiconductor company
    website. Can you infer from this text what the company
    does? Classify it into these {len(categories)} classes
    which should be self-explanatory: {categories}.
54      A firm can be part of 1 or many classes. For instance,
    big firms could engage in multiple activities while
    smaller firms tend to specialize in one class.
55      Return your predicted class(es) as a python list with
    strings, e.g., ['Design'] or ['tool_resource', '
    Fabrication']. Return only this list, nothing else.
56      Here the website text:
57      {description}
58
59      REMINDER: only return your predicted class out of these {
    categories}, DO NOT use any other categories, and DO NOT
    return anything except the predicted list.
60      """
61      messages = [{"role": "user", "content": prompt}]
62      encodeds = tokenizer.apply_chat_template(messages,
    return_tensors="pt")
63      model_inputs = encodeds.to(device)
64      generated_ids = model.generate(model_inputs,
    max_new_tokens=1000, do_sample=True)
65      decoded = tokenizer.batch_decode(generated_ids)
```

```
    response = decoded[0].split("[/INST]")[-1].strip()
    if response.endswith("</s>"):
        response = response[:-4].strip()
    return response


### Find categories within the returned text ###
def find_categories_in_text(categories, text):
    return [category for category in categories if category in text]

### Main execution function ###
def main():
    setup_logging()
    args = parse_arguments()
    load_environment_variables()

    # Check if OUTPUT_DIR is set in the environment and valid before proceeding
    output_dir = os.getenv('OUTPUT_DIR')
    if not output_dir:
        logging.error("OUTPUT_DIR environment variable is not set. Falling back to the current working directory.")
        output_dir = os.getcwd()

    output_path = os.path.join(output_dir, f'{args.output_filename}.pickle')

    logging.info("Loading data...")
    df = pd.read_pickle(args.input_filepath)

    logging.info(f"Preparing model and tokenizer for {args.model}...")
    model, tokenizer, device = initialize_model_and_tokenizer(args.model)

    exploded_class = df['class'].explode()
    categories = list(exploded_class.unique())
    df['predicted'] = None

    logging.info("Classifying descriptions...")
    for i, row in df.iterrows():
        if i % 10 == 0:
            logging.info(f"{i} companies processed...")
        result = call_model(model, tokenizer, device, row['text'], categories)
        found_classes = find_categories_in_text(categories, result)
        df.at[i, 'predicted'] = found_classes
```

```
107
108     logging.info(f"Saving results to {output_path}...")
109     with open(output_path, 'wb') as f:
110         pickle.dump(df, f)
111
112     logging.info("Processing completed successfully.")
113
114 if __name__ == "__main__":
115     main()
```

Listing 4: Implementation for Huggingface models

B Implementation for GPT models

```
import pandas as pd
import pickle
import argparse
from langchain.chains.openai_functions import
    create_structured_output_runnable
from langchain.prompts import ChatPromptTemplate
from dotenv import load_dotenv
from langchain_openai import ChatOpenAI
import logging
import os

# Setup logging configuration
def setup_logging():
    logging.basicConfig(level=logging.INFO, format='%(asctime
    )s - %(levelname)s - %(message)s')

# Load environment variables from the .env file
def load_environment_variables():
    load_dotenv()

# Parse command-line arguments
def parse_arguments():
    parser = argparse.ArgumentParser(description="Run the
    model with custom inputfile (full path) and output
    filename (only name).")
    parser.add_argument('-input_filepath', type=str, required
    =True, help="Input file path, needs to be pickle dataframe
     with columns 'text' that will be used for prediction and
    'class' that is the true class of the firm.")
    parser.add_argument('-output_filename', type=str,
    required=True, help="Output filename as it will be saved
    to OUTPUT_DIR")
    parser.add_argument('-model', type=str, required=True,
    help="Exact OpenAI model name, for instance 'gpt-3.5-turbo
    ' ")
    return parser.parse_args()

# Load data from a given file path
def load_data(file_path):
    return pd.read_pickle(file_path)

# Prepare categories from the dataframe
def prepare_categories(df):
    exploded_class = df['class'].explode()
    unique_classes = exploded_class.unique()
    return {i: category for i, category in enumerate(
    unique_classes)}
```

```
36
37 # Create a model instance for structured output generation
38 def create_model(model_name, categories_dict):
39     """
40     Create a model instance using LangChain and OpenAI with a
       specific structure defined for JSON schema.
41
42     This function initializes a structured output runnable
       with a specific prompt template and JSON schema,
43     tailored for classifying company activities into specific
       categories. The categories_dict is used to
44     dynamically insert the possible categories into the JSON
       schema description, ensuring the model's output
45     is interpretable and aligned with the expected
       classification categories.
46
47     Args:
48     - model_name (str): The name of the model to be used from
       OpenAI.
49     - categories_dict (dict): A dictionary mapping category
       indices to their descriptive names.
50
51     Returns:
52     - A runnable instance configured to generate structured
       outputs based on the given model and schema.
53     """
54     json_schema = {
55         "title": "Company",
56         "description": "Identifying information about
       activities of a company based on its description.",
57         "type": "object",
58         "properties": {
59             "category": {
60                 "title": "Company's Value Chain Category",
61                 "description": f"The company's predicted step
       in the Value Chain should be one of the following values:
       {categories_dict}. Please return only the class numbers."
       ,
62                 "type": "string"
63             },
64         },
65         "required": ["category"],
66     }
67
68     prompt = ChatPromptTemplate.from_messages([
69         ("system", "You are an supply chain expert that
       classifies semiconductor companies based on their
       activities."),
70         ("human", "Here is the description of a semiconductor
```

```
        company. Classify the firm based on its activity into one
        of these steps in the value chain: {input}"),
71          ("human", "Tip: Make sure to answer in the correct
        format, only returning the numbers of your predicted class
        . Remember, a firm can be classified into multiple
        categories.")
72      ])
73
74      llm = ChatOpenAI(model=model_name, temperature=0)
75      return create_structured_output_runnable(json_schema, llm
        , prompt)
76
77  # Translate numerical categories to their descriptive names
78  def translate_to_categories(input_data, categories_dict):
79      numbers = [int(s) for s in str(input_data).replace('{', '
        ').replace('}', '').split(',') if s.isdigit()]
80      return [categories_dict.get(num) for num in numbers if
        num in categories_dict]
81
82  # The main execution function
83  def main():
84      setup_logging()
85      load_environment_variables()
86      args = parse_arguments()
87
88      # Check if OUTPUT_DIR is set in the environment and valid
        before proceeding
89      output_dir = os.getenv('OUTPUT_DIR')
90      if not output_dir:
91          logging.error("OUTPUT_DIR environment variable is not
        set. Falling back to the current working directory.")
92          output_dir = os.getcwd()
93
94      output_path = os.path.join(output_dir, f'{args.
        output_filename}.pickle')
95
96      logging.info("Loading data...")
97      df = load_data(args.input_filepath)
98      categories_dict = prepare_categories(df)
99
100     logging.info("Setting up model...")
101     model = create_model(args.model, categories_dict)
102
103     logging.info("Start analysis...")
104     for i, row in df.iterrows():
105         if i % 10 == 0:
106             logging.info(f"{i} companies processed...")
107         result = model.invoke({"input": row['text']}).get('
        category', 'Unknown')
```

```
108             df.at[i, 'predicted'] = translate_to_categories(
    result, categories_dict)
109
110         with open(output_path, 'wb') as f:
111             pickle.dump(df, f)
112
113         logging.info("Processing completed successfully.")
114
115 if __name__ == "__main__":
116     main()
```

Listing 5: Implementation for GPT models

C Evaluation Function

```python
import pandas as pd
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.metrics import precision_score, recall_score, f1_score

def evaluate(outcome_file):
    df = pd.read_pickle(outcome_file)
    df['class'] = df['class'].apply(lambda x: x if isinstance(x, list) else [])
    df['predicted'] = df['predicted'].apply(lambda x: x if isinstance(x, list) else [])

    mlb = MultiLabelBinarizer()
    mlb.fit(df['class'])
    y_true = mlb.transform(df['class'])
    y_pred = mlb.transform(df['predicted'])

    precision = precision_score(y_true, y_pred, average='micro')
    recall = recall_score(y_true, y_pred, average='micro')
    f1 = f1_score(y_true, y_pred, average='micro')

    exact_matches = sum([set(actual) == set(predicted) for actual, predicted in zip(df['class'], df['predicted'])])
    exact_match_ratio = exact_matches / len(df)

    partial_matches = sum([len(set(actual) & set(predicted)) > 0 for actual, predicted in zip(df['class'], df['predicted'])])
    partial_match_ratio = partial_matches / len(df)

    # Creating DataFrame with object type to allow mixed types
    results_df = pd.DataFrame({
        'Metric': ['Precision', 'Recall', 'F1 Score', 'Exact Matches', 'Exact Match Ratio', 'Partial Matches', 'Partial Match Ratio', 'Nr Firms'],
        'Value': [
            round(precision, 2),  # float
            round(recall, 2),     # float
            round(f1, 2),         # float
            exact_matches,        # int, stored as object
            round(exact_match_ratio, 2),  # float
            partial_matches,      # int, stored as object
            round(partial_match_ratio, 2), # float
            len(df)               # int, stored as object
        ]
```

```
    }, dtype=object)  # Setting dtype as object

    return results_df
```

Listing 6: Full evaluation function

D Example Orbis text

```
'main_products_and_services':'Specialty materials for the
   semiconductor and display industries', 'full_overview':'
   This company, based in the United States of America, is
   engaged in the development, manufacture, transportation,
   and handle of specialty materials for the semiconductor
   and display industries. It was incorporated in 2016 and
   has registered headquarters located in Tempe, Arizona. It
   specialty chemicals and materials used in semiconductors,
   as well as specialty gases used in the semiconductor
   manufacturing process, including high purity process
   materials for deposition, metallization, and chamber
   cleaning and etching; chemicals mechanical planarization
   slurries; organosilanes; organometallics and liquid
   dopants for thin film deposition; and formulated chemical
   products for post-etch cleaning primarily for the
   manufacture of silicon and compound semiconductors, and
   thin film transistor liquid crystal displays. It develops,
    designs, manufactures, and sells bulk gas, specialty gas,
    and specialty chemical cabinets and systems, which are
   used to manage the delivery of key materials into the
   semiconductor manufacturing process; and flow and
   temperature control systems and analytical systems to
   capture data. In addition, it offers on-site services to
   assist customers in managing the inventory of gases and
   chemicals comprising ordering, product changes and
   monitoring, quality assurance, operation of delivery
   systems, and managing the bulk gas and specialty gas
   operations.', 'primary_business_line':'Engaged in the
   development, manufacture, transportation, and handle of
   specialty materials for the semiconductor and display
   industries', 'trade_description_english':'Versum Materials
   , Inc. is a provider of solutions to the semiconductor and
    display industries. The Company is engaged in the
   development, manufacturing, transportation and handling of
    specialty materials. Its segments include Materials;
   Delivery Systems and Services (DS&S), and Corporate. The
   Materials segment is an integrated provider of specialty
   materials for the electronics industry, focusing on the
   integrated circuit and flat-panel display markets. The DS&
   S segment designs, manufactures, installs, operates, and
   maintains chemical and gas delivery and distribution
   systems for specialty gases and chemicals delivered
   directly to its customers' manufacturing tools. The
   Company is engaged in molecular design and synthesis,
   purification, advanced analytics, formulation development
   and containers and delivery systems for the handling of
```

```
high purity materials.'
```

Listing 7: Example full company description

E Acknowledgement of Artifical Intellligence Tools Used

For purposes of writing revision and grammar checks, the following tool was used: Chat-GPT v.4 (https://chat.openai.com/) The tool was prompted with the following text:

> "You are tasked with revising the text, focusing on correcting grammar, punctuation, adding any missing articles, and enhancing the style. While making these revisions, you must preserve the original meaning and manner of writing. You may suggest alternative words to better convey the intended message, but any substantial changes should be avoided to retain as much of the original text as possible, don't change anything if the change is not necessary."

The output was used to help revise writing in all segments of the thesis.