High Quality NLP Data Pipelines

Master of Science (Tech) Thesis
University of Turku
Department of Computing
Biomedical Engineering and
Health Technology
2025
Juuso Torikka
Examiners:
Prof. Filip Ginter
Dr. Antti Rahtu
MSc. Johannes Miettinen

The originality of this thesis has been checked in accordance with the University of Turku quality assurance system using Turnitin Originality Check service.

UNIVERSITY OF TURKU
Department of Computing

Torikka, Juuso High Quality NLP Data Pipelines

Master of Science (Tech) Thesis, 58 pp.
Biomedical Engineering and Health Technology
01, 2025

In an era where companies utilize Natural Language Processing to guide business processes, the importance of high quality data pipelines cannot be overstated. Customer interactions can produce vast amounts of data containing crucial information about the customers. This thesis explores the factors of high quality NLP data pipelines, their challenges and best practices for building a robust applications delivering reliable and actionable insights.

Producing valuable insights through NLP requires an automated, scalable and dependable data pipeline, which is able to produce inference results for various types of interaction texts while maintaining data quality. This thesis explores through literature the requirements and steps to produce these pipelines in an industrial scale while uncovering seven key factors which the system should aim to fulfill. This thesis considers factors from ways of working to data governance and creates an end-to-end suitable list of factors to consider applying when building an NLP data pipeline. These factors suggest that the system must be scalable, dependable, automated, monitored, well documented, its data quality validated and evaluated, while being developed in an effective iterative way.

This thesis emphasizes the importance of data quality. Data quality is assessed through various validation techniques and data quality dimensions which can identify deficiencies or issues within the resulting data. As this thesis aims for industrial scale results, it also explores the aspects of data governance and documentation by means of data lineage and metadata.

A literature review provides a frame for the implementation. At the end of this thesis, a high quality NLP data pipeline is developed according to the high quality factors identified from the literature. The pipeline makes use of pre-built models to extract topics and sentiments from interaction texts.

Ultimately, this thesis sets out to provide a comprehensive guide for practitioners and researchers to design and implement high quality NLP data pipelines.

Keywords: NLP, Data Pipeline, Data Quality, Scalability

Contents

Preface

I would like to thank my thesis supervisor Professor Filip Ginter of University of Turku for his great guidance and motivating way of supervising this thesis throughout the writing process. I would also like to thank my other supervisors of this thesis, MSc. Johannes Miettinen and Dr. Antti Rahtu, for their everlasting patience and will to guide me in my pursuit of this thesis with their irreplaceable knowledge in their fields. Thank you.

Turku, December 16th 2024

Juuso Torikka

1 Introduction

Natural Language Processing (NLP) is continuously increasing its status in many different industries. Interaction between customers and businesses is constantly under development from simple chat bots with predefined questions and answers to fully Generative Pre-Trained Transformers (GPT). Besides direct customer interaction, companies can utilize various NLP methods to gather structured insight from conversations and other types of data, and follow-up with data-driven decision making.

Results of bad customer interactions are estimated in the trillions of dollars annually [1]. Analyzing individual conversations between customer and the business can be extremely valuable to maximize the satisfaction for both parties. Analyzing texts is very time consuming, but modern NLP models can simplify the process by creating summaries and analyzing the topics and sentiments of each individual conversation incredibly fast.

However, scaling a language processing task to industrial size workloads, and a fully automated process with reliable outcomes, can be extremely demanding. This thesis sets out to tackle the issues of creating a large scale, industrial level DevOps, MLOps and DataOps compliant NLP pipeline to gain qualitative data about customer interactions.

In this thesis I choose to focus on answering what aspects enable high quality data pipelines and how to measure the data asset's quality, how human interactions can be transformed to actual business use cases, as well as implementing a large-scale NLP data pipeline utilizing already existing Machine Learning (ML) models.

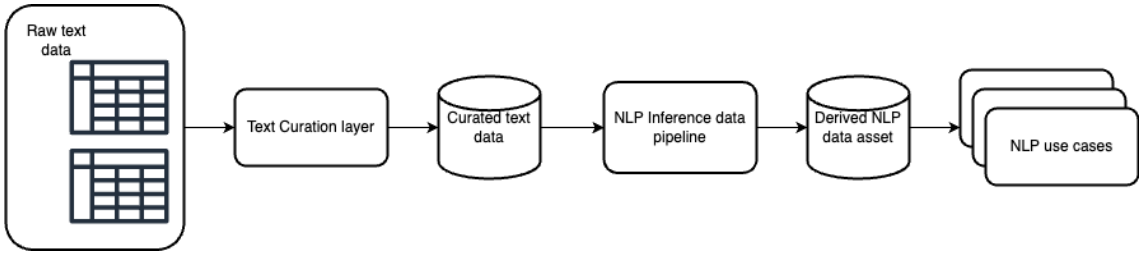

Figure 1. High level NLP data pipeline

1.1 Objectives and Research Questions

The objective of this thesis is to uncover requirements and steps needed to produce an industrial level MLOps and DataOps compliant NLP data pipeline. More specifically, this thesis focuses on finding a set of factors which enable producing these high quality data pipelines. The literature review is done with NLP use cases in mind, thus including relevant fields of technology required with it. The NLP data pipeline illustrated in Figure 1 describes the full NLP process on a high level where texts are refined to a standard required by the models and compliance, processed by the model and stored in a manner suitable for usage or further analytics. The implementation part of this thesis is focused on the inference and use case steps of the pipeline. The research questions are formulated based on the objective of deriving a check list of requirements which, when fulfilled, can constitute a high-quality NLP data asset.

The research questions of this thesis are hereafter referred to by their identifiers RQi and are formed as follows.

RQ1: How can raw texts be transformed into actual NLP use cases?

RQ2: What are the key factors of a High Quality NLP Data Pipeline?

RQ3: How to validate and measure data quality in an NLP data pipeline?

Abbreviations and acronyms

NLP	Natural Language Processing
NL	Natural Language
IR	Information Retrieval
ML	Machine Learning
ASR	Automatic Speech Recognition
STT	Speech-To-Text
AM	Acoustic Model
LM	Language Model
BLEU	BiLingual Evaluation Understudy
WER	Word Error Rate
MLOps	Machine Learning Operations
DevOps	Development Operations
DataOps	Data Operations
KPI	Key Performance Indicator
CSP	Cloud Service Provider
SaaS	Software as a Service
PaaS	Platform as a Service
IaaS	Infrastructure as as Service
CICD	Continuous integration and continuous delivery
ETL	Extract, Transform and Load
API	Application Programming Interface
LLM	Large Language Model
AWS	Amazon Web Services
ACID	Atomicity, Consistency, Isolation, Durability
PII	Personally Identifiable Information
UDF	User Defined Function
SCM	Source Control Management
GX	Great Expectations

Part I

Literature review

Natural Languages (NL) are a way for humans to communicate verbally or in writing. Natural languages consist of structured words which are represented by symbols such as letters, special characters and numbers, and a various sets of rules, which state how words and structured sentences can be manipulated or structured [2]. In 1950s, NLP was introduced as the intersection of artificial intelligence and linguistics. At the time, information retrieval (IR) was distinct from Natural Language Processing which then utilized highly scalable statistics to process large amounts of text, which Manning *et al* [4] give excellent introduction to. NLP started out with simple approaches to translations between languages, rather unsuccessfully, but in time NLP and IR have both developed to similar direction, and thus converged nearly completely together [3]. This literature review introduces some of the most common NLP methods, such as topic and sentiment analysis, as well as others within the scope. As Large Language Models (LLM) are currently developing with substantial pace, the issues and benefits with them are not covered thoroughly or in depth, as they are likely to have changed to a new set of issues and features in near future. This thesis does not dive into the development or functionalities of these NLP methods, but introduces them on a high level.

To utilize modern NLP methods at industrial scale with large quantities of texts, a data pipeline of high quality must be created to handle the vast amount of texts while being maintainable, reliable and trusted to produce high quality data. Within this thesis, the high quality factors of these pipelines are investigated and reproduced in the implementation.

The scope of this thesis is limited to NLP data pipelines, and more precisely to application of inference, and thus the deep-dive into NLP itself is left out. Literature review of factors of modern NLP pipelines is carried out in this chapter. Relevant aspects of forming a industrial level NLP pipeline are introduced and unpacked.

2 NLP use cases

Among the main goals of modern NLP is to extract various types of information from texts, whether it's transcribed phone calls or plain text, generating more text based on given instructions, and eventually provide human-like language processing for various tasks and thus enable human interaction with machines using natural language [2].

Information extraction from texts can be divided into several sub-segments, of which the most prominent and most used are sentiment analysis and topic analysis [12], which are also the main applications implemented in this thesis. In addition to researching the basics of Information Extraction, this thesis also introduces Text summarization. Each of these NLP procedures have a use case in various fields of industry, and they are the applications covered in the scope of this thesis.

NLP has various use cases in many different industries. This introduction to the use cases focuses on healthcare industry. For example, sentiment analysis of customer feedback, patient sentiment in mental health monitoring of text-based therapy applications or overall public sentiment during public health outbreaks. Sentiment analysis tries to understand the human emotions. Topic analysis can be used to gather insights from vasts amounts of literature to find trends in treatment or development. Topic analysis can also be beneficial in extracting trends within patients and their conditions or concerns about health by applying it to electronic health

records or support forums. The purpose of topic analysis is to find key themes in large sets of data. Other use case examples for NLP applications in healthcare are summarization, which as the name suggests, summarizes large quantities of texts, Questions and Answer assistants for patients or clinical decision support for doctors, machine translation, or LLMs, which can be used to produce documentation, simulation or finding patterns in vast selection of data. These use cases are examples, and the selection of other use cases is nearly endless and can vary between industries.

In the following sections, the previously mentioned applications of NLP, that are part of this thesis' scope, topic, sentiment and summary extraction are introduced.

2.1 NLP methods

Information Extraction (IE) is a technology enabling relevant content extraction from textual information electronically powered by natural language processing [5] [6]. Topic and Sentiment modeling are both part of information extraction, and the key modeling techniques used in the implementation part of this thesis. IE also covers tasks like Named Entity Recognition, a task of classifying predefined entities such as names and places, Relation Extraction, the task of detecting relations between the discovered entities as well as Event Extraction which is the task of identifying events and their details from text [6].

The purpose of Topic modeling is to analyze and extract semantic pieces of information from texts. A topic is derived from the usage of a particular terms in the same text or document [8]. These models can be probabilistic, which try to automatically extract the topics from text, or keyword-assisted, where the domain knowledge of the creators enhances the model's ability to extract more relevant topics. Probabilistic models offer a great scalability to various use cases, but they are

often lacking the ability to measure specific concepts from texts, whereas keyword-assisted models lack the similar scalability, but are often better in extracting the most relevant topics [9].

The purpose of Sentiment analysis is to identify the sentiment expressed in the text. Sentiment analysis often produces either binary negative or positive result, or with an additional neutral result. Analyzing the sentiment in text can be more complex when the text contains different propositions that conflict with each other, but utilizing a similar keyword-method as with topic modeling, this can be simplified [8]. More advanced sentiment analysis, a beyond-polarity, can dive deeper into the polarity of the text by distinguishing different emotional states [10].

Summarization is a method of NLP which attempts to condense documents or text to its essential content. Summarization is a crucial element of accessing the growing amounts of information available. Automatic text summarization is often based on a sentence-extraction paradigm, which utilizes a list of features and keywords, believed to indicate the relevance of a sentence to interpret the text [11].

3 MLOps

In 2015, the problems with technical debt and the increasing usage of ML applications were highlighted in the paper by Sculley *et al* indicating that a shift in team's development culture is required to provide long term and maintainable ML systems [15]. DevOps and DataOps have established themselves as methodologies applied across industries to develop high-quality and fast time-to-market software and data engineering products [13]. Common objective with these paradigms is to reduce the gap between development and operations and to create a consistent framework that enables fast, frequent and reliable releases to production. A large part of ML

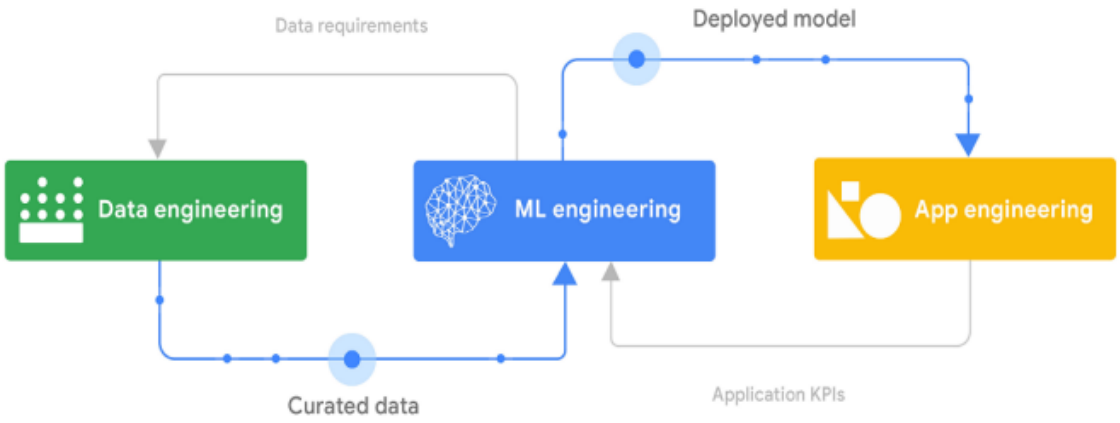

Figure 2. Relationship of data engineering, ML engineering and app engineering [13]

projects have remained in the proof of concept stage and never reached production, as the ML community is heavily focused on producing ML models and not building production-ready ML systems [14].

MLOps (Machine Learning Operations) is introduced here and the specific methodology applied in this thesis is followed up in the upcoming section to provide context for High Quality MLOps factors.

3.1 Google MLOps

In this section, Google's MLOps is introduced and summarized as a process. This section is fully referencing the original Whitepaper by *Salama, Kazmierczak, and Schut* [13]. As the focus of this thesis lies in the data and app engineering practices of MLOps, the model engineering is ignored in the practical implementation due it not fitting the scope. The introduction to Google's MLOps framework is done to give insight on how the implemented NLP data pipeline integrates with MLOps compliant model development by enabling the various use cases the model is required for.

Google's Practitioner's Guide to MLOps Whitepaper was published in 2021 to give

insight on the themes of scaling and automating ML systems. These themes cover the issues with scaling in cloud environments with reduced operational overhead and automating the process of deploying and executing, as well as ability to operate the technologies for data processing and ML pipelines in production environments efficiently, frequently and reliably. The motivation behind the Whitepaper, is to enable production ready ML systems for companies as a large part of models making it into production break in usage because they fail to adapt to changes.

By applying the MLOps process as standard development procedure the benefits can be seen in shorter development cycles, collaboration, reliability, scalability and security as well as in operational and governance processes. Building an ML system requires combinations of engineering capabilities as shown in Figure 2. For MLOps to work, the workflow requires robust data engineering practices in data ingesting, integrating, curating and refining. Models deployed to production should utilize the curated data created by the data engineering and work as components of or support processes in various applications. Deploying the model as part of any system requires the monitoring of relevant Key Performance Indicators (KPI) including the model output quality, effective usage and other business oriented KPIs.

The MLOps life cycle is determined in a way that each part of the process is integrated and iterative, thus enabling and encouraging continuous development and improvement of the process. The end-to-end workflow is designed in an Agile iterative format instead of the very common waterfall workflow. Each individual step can be skipped if needed and returned to when necessary to maximize the capabilities of each sub process. MLOps capabilities are key to implementing high functioning MLOps process into the model development workflow. These capabilities aim for a continuously iterated and scalable development process of ML models and thus the

utilization of these models in the data pipeline should follow similar standards.

4 High Quality NLP data pipeline

The goal for this section of this thesis is to provide insight into factors, that enable producing a High Quality NLP data product in terms of *RQ2* as well as finding solutions for the *RQ1*. The outcome of this section is a detailed description of factors that should be considered when building NLP data pipelines and products. The High Quality in question factors are derived from generally used MLOps frameworks and more thoroughly defined within Google's MLOps framework [13]. The resulting list of factors provided in this section will be used in the implementation part of this thesis, when the NLP Inference application is created.

4.1 Evaluation and validation

This section focuses on providing answers to *RQ3*. It will cover the validation and evaluation methods for input data in NLP use cases, as well as introduce the methodology for validating and evaluating the output data using dimensions of quality to validate that each aspect of quality is being fulfilled in the output dataset.

4.1.1 Source data

NLP applications' validation and evaluation can be divided into multiple categories depending on the step of the full process of transforming raw texts into the actual business use case. Language processing models require written text to be able to extract information. As most of human interaction is done using spoken languages, the audio requires computing to get the input as text, or in other words, transcribed. Automatic Speech Recognition (ASR) has rapidly developed in the recent past to enable various services like personal assistants and speech-to-text (STT) translators.

Speech-to-text applications are based on Acoustic Models (AM), which produce a probability over each letter in the alphabet for the audio signal, and Language Models, that estimate a logical and coherent wording over these probabilities provided by the AM. This means the ASR's effectivity is highly dependable on the AM's and the Language Model's (LM) training data and how well it it relates to the testing conditions [16].

Transcriptions can be created manually by listening to audio and writing it, or automatically as the end result of an STT process. Evaluating automatic transcriptions requires manually transcribing audio to achieve gold standard of the transcription, which then is compared to automatically transcribed texts. Standard procedure of evaluating the performance of these services is to use Word Error Rate (WER). When aligned with the manual test transcription, the number of errors is computed as the sum of substitutions (S), deletions (D) and insertions (I) which is divided by the total count of words in the word sequence [16]. WER can thus be calculated as follows:

$$WER = \frac{I + D + S}{N}$$

Substitutions, deletions and inserts are referring to model's tendency to either delete or change an unrecognizable word or to insert completely new words into the sequence of words and characters.

Transcriptions are often evaluated using other metrics such as the BiLingual Evaluation Understudy-score (BLEU) [7], which computes the word-based overlap between the machine transcript and the gold standard transcript. As the name suggests, BLEU is often used for machine translations, but is also very common in evaluation of transcriptions. The computation checks if the transcript contains the same, or similar, words as the gold standard. However, this assumes that higher overlap di-

rectly means better output and this might lead to over simplification of evaluation as errors or missing words can change the meaning of the transcription. Overall, it is recommended that BLEU-scores are accompanied with human judgment, as the scores are required to correlate with the direct evaluation [17].

Advanced methodologies to evaluate and validate input texts created by transcripts have surfaced to provide metrics beyond the traditional ways such as WER or BLEU scores, giving insight through deeper analysis of the given text. When comparing the effectiveness of transcription services, evaluating the semantic textual similarity (STS) between transcripts and the golden sample can provide deeper knowledge about the semantics of each text. Major goal of STS is to provide semantic correlation and enable comparison which is less sensitive to variations in phrasing, but more focused on the meaning of the text unlike statistical approaches like the BLEU-scores which are focused on the similarity of the words themselves [30].

Rising methodologies to assess the task of STS are utilizing BERT modeling (Bidirectional Encoder Representations from Transformers), and its variations and it's stated to provide state-of-the-art results in sentence-pair regressions such as STS. Challenges with STS using BERT based models stated in the research by Hyuinjin *et al* [31] were that while these models are effective in word-level tasks, the effectiveness can be questioned in sentence similarity tasks. It was concluded that BERT based models such as BERT, A Lite BERT (ALBERT) and Sentence-BERT (SBERT) heavily require task specific fine-tuning to be able to perform sentence level similarity tasks, but the SBERT model outperforms each of its predecessors due to its sentence-pair training structure. SBERT, proposed by Reimers & Gurevych [32] alters a pre-trained BERT with siamese and triplet network structure which utilizes only cosine similarity unlike standard BERT models. As well as providing better

similarity on sentence level, the SBERT model is highly more efficient compared to BERT model due to its siamese architecture. This thesis does not go into full detail about the functionality and differences between these methods, as the issues and features are in constant development and are likely to be changed in the near future.

Common between all previously noted evaluation methods is the fact that they rely on reference such as golden samples. With the emergence and continuous increase of LLMs, the possibility of deriving accurate evaluation metrics without a reference text is increasing. However, evaluating using LLMs contains issues as all evaluation strategies do. Gao, Hu and Ruan [33] have listed the issues with evaluating texts without reference using LLMs and have stated that LLMs contain issues with robustness, efficiency and fairness. Efficiency of evaluation is crucial when dealing with a multitude of texts. Current LLMs struggle with memory issues as the key-value cache for each batch can dynamically switch from very large to back to small and can lead to memory waste, fragmentation and duplication and thus lead to limited batch sizes. These efficiency issues can be tackled using vLLMs introduced in [34], which utilize virtual memory that can achieve near-zero waste. While there are various LLM-derived methods, the implementation of these in any environment is near-impossible due to those being closed-source and do not provide their parameters, representations or logits available to the public. Besides previously mentioned issues, the LLM approach has trouble with biases. As stated by *Gao, Hu and Ruan*, LLMs often prefer longer and more verbose texts with factual flaws to factually correct, short texts with grammatical errors [33]. Where LLM evaluation stands out, is that the user can provide evaluation criteria and methods in natural language, which again leads to great flexibility. Prompting the LLM to explain itself while assessing texts can make this approach more interpretable. Utilizing LLMs in text comparison or STS can already be done without supervision, but often human

supervision is stated to provide better results [33].

4.1.2 High-quality data

Data quality is a metric that should be measured in data pipelines. Data can be considered high-quality when the data looks and acts like it is expected to, complete and accurate, as well as available when it is expected to be. High-quality data can thus be defined by its expectations, as quality must be measured against a common standard or a business related aspect, which varies between the intended use of the data, what the data is supposed to represent and why it exists in the first place. Thus data quality is directly related to the purpose of the data [37].

A data quality dimension represents a measurable aspect, or a metric, of data which indicates a degree of quality. To assess data quality, it is essential to examine the unique characteristics of the data and determine how well it meets specified expectations and requirements [37].

As human interaction comes in different languages and diverse properties, the data can be difficult to generalize. In terms of NLP, we can utilize many common data quality dimensions relevant for the NLP pipeline. Inspired by the common data quality dimensions, we can tailor the dimensions to be more relevant to NLP. Dimensions used here are derived from conference paper *Data Quality in NLP: Metrics and a Comprehensive Taxonomy* by *Dang & Rakesh* [39] as well as extended from common data quality dimensions with respect to both data expectations and restrictions in the following way:

1. **Completeness**

   All the necessary information is available to achieve the expectation that the data holds [39]. Completeness acts as a prerequisite to other dimensions as data must exist before its quality or other aspects can be measured and thus is the first condition of completeness. For a dataset to be considered complete, it must fulfill at least three conditions [37]:

   Width

   The dataset must be defined such that it contains all the desired attributes.

   Depth

   The dataset must contain desired amount of data.

   Density

   The attributes must be populated to the desired extent.

   When tailored to the use case of NLP, this could mean that all outputs contain valid inference results or any metadata the model produces. In NLP pipelines the completeness can be argued to not be fulfilled in cases where key topics visible in source data are not represented in the outcome of the topic inference.

2. **Consistency**

   Within limits of constraints, all data entries should adhere to specified format, standard or rule without exceptions, or more directly, consistency represents the absence of variation in data entries [39]. Data should also remain intact between records [40]. Consistency is often measured in data over time, and how it conforms to an equivalent set of data, such a set produced by the same process over time. Consistent inference data should provide similar results between similar inputs both in sentiment and topic analysis. If the results of similar inputs vary in data over time, the data can be considered inconsistent.

3. **Representativeness**

With respect to the data expectations, the data must provide a broad enough range of entries to produce a holistic view of the subjects. The comprehensiveness required is mandated by the expectations [39]. If the inference consistently favors certain topics over less frequent labels, the output might lack representativeness. When put to the domain of health care, the less frequent topics might give crucial insights to patient care or treatment adjustments.

4. Appropriateness and Reasonableness

The appropriateness of data can be implied through its direct relevance and suitability towards the expectations [39]. In an NLP context, the input data should align with the specific use case scenario such as health care chat service topic modeling. For instance, if the goal is to model doctor-patient conversations, the input data must consist of healthcare related chat-messages. The output data must also conform to domain standards and be understandable. Appropriateness and Reasonableness are often similar to each other. Reasonableness is more often referred to when numerical data is expected to be between set thresholds.

5. Accuracy

Accuracy represents the data's correctness or truth, and data can be defined accurate when it's true and correct. Measuring accuracy requires comparing data to the actual object or entity it represents, or to a reliable, validated version of that representation [37].

6. Validity

Validity, not to mistake with accuracy or correctness, compares data to domain specific business rules or standards defined within the domain. Unlike accuracy, validity is limited to measure against surrogates for real-world objects or entities, allowing it to be measured directly from withing the dataset [37]. Validity expects output data to conform to expected formats and values. For example

with valid data, the sentiment score should fall between -1 and 1 and topic output should contain only predefined relevant labels.

7. Timeliness

Timeliness, in its most general definition, refers to the appropriateness of when an event happens. This can be interpreted in the context of data quality as the degree to which data represents reality at the required point in time [37]. This means that the data must be recent enough to meet the expectation defined for the data.

8. Uniqueness

Uniqueness as a dimension refers to the fact that there should not be duplicate records for the same entity, or the entire table. Uniqueness should be checked at least for the identifying primary keys for each data entity [38].

9. Integrity

Integrity refers to state of the data being whole. Integrity represents the internal consistency and completeness and thus reveals to which extent the data conforms to the data model's internal rules [37].

In terms of *RQ3*, the data quality dimensions listed above exist to provide both validation and evaluation for the output data. Evaluating the data before the inference phase of the pipeline is extremely beneficial to enable accurate performance for the model. Besides the evaluation and validation of the input data, the output of the model and the final output dataset should be evaluated and validated as well.

4.2 Scalability

MLOps compliance demands the possibility of expanding and incrementing the requirements of the process. This is why scalability becomes a key factor when designing a data pipeline for an NLP product. In the MLOps whitepaper, scalability

is included as a key property of Data Processing as the systems needs to be able to support scalable batch or streaming processes for both ML training and serving workflows as well as be modular in terms of collaboration [13].

4.2.1 Cloud computing

Cloud computing is referred by *Sehgal & Bhatt* in the book *Cloud Computing: Concepts and Practices* [18] as IT services, data or applications using dynamically scaling pools, but the definition of NIST (National Institute of Standards and Technology) is still evolving [19]. Essential characteristics of cloud computing according to NIST are On-demand network access to shared pools of computing resources. Computing resources can in this case include storage, networks, servers, applications, and other services. These resources need to be rapidly provisionable in self-service manner without the need for management efforts or interaction with the Cloud Service Provider (CSP). Cloud computing comes in various forms of services and providers often categories the services in three tiers:

1. **Software as a Service (SaaS)**

   Often referred to as Application Service Provider model, it can be seen as a multi-tenant cloud-platform which does not require local installation and supports multiple customers simultaneously [20]. Examples of SaaS products include Microsoft Office 365 and Google's Gmail and Docs.

2. **Platform as a Service (PaaS)**

   PaaS provides developers with a platform equipped with necessary tools required in each stage of development [18]. The resources are scalable with minimal administrative overhead and produce an end-to-end development environment in the cloud [20]. Examples of PaaS products include Amazon Web Services and Microsoft Azure.

3. **Infrastructure as a Service (IaaS)**

   IaaS can be described as delivery of resources such as processing power, networking and storage [20]. From the three tiers the IaaS is the most flexible as each unique service can be handpicked on-demand and users often benefit from the usage-based pricing where up-scaling is possible without pre-planning. Example of IaaS is Amazon EC2.

As ML systems grow, the possibility of scaling up resources becomes a critical task. With efficient resource allocation, workload scheduling and parallel computing, the systems can mitigate bottlenecks and latency issues which can substantially affect the performance of these systems. In a machine learning data pipeline, and more specifically, in the inference step, the workload may vary significantly, which creates a need for dynamically scaling the resources required. The ability of cloud computing to scale up or down the resources on-demand based on the needs of the system is crucial for optimal performance and cost-effectiveness [23]. In such data pipeline, cloud computing mitigates many of the issues that often come-up in any software development such as dependency issues, resource allocation and security concerns.

4.2.2 Computation

With high amounts of data, the computational power to perform inference tasks becomes a crucial aspect to consider. Many modern big data applications utilize distributed and parallel computing strategies to achieve highly fault-tolerant and resilient processing. Apache Spark is an open-source distributed processing model that allows utilizing both parallel and distributed computing by applying a collection of objects partitioned across a set of machines and utilize these machines' multiple cores in parallel to process the data [21]. Spark allows the user to write massively parallelized operators and not to have to worry about the distribution or fault-tolerance of the data while utilizing the in-memory caching and optimized

query execution [22]. By utilizing cloud computing, these features of parallelism and distributed computing can be utilized effectively and resiliently.

4.3 Dependability

Many systems can lack in quality when the development has narrowly focused on meeting other quality measures such as performance of the system and the dependability of the system has been neglected. Dependability as a software quality measure can be defined as the characteristics of the system that ensures it can consistently and correctly perform its intended functionality enabling users to justifiably rely on the service it provides.

Dependability engineering consists of several attributes, as displayed in Figure 3 (after Laprie *et al*), which in addition to the dependability attributes, shows the means and impairments to achieve dependability. The attributes include, but are not limited to availability, reliability, security, integrity, confidentiality and maintainability. As some of these quality attributes overlap with each other, and attributes like security, integrity, as well as confidentiality fall in the category of information security. Safety in turn could be categorized as an extension of reliability, but the attributes are often grouped like in the article by Song & Sohn [26] to form more compact list of attributes. For the scope of this thesis, the dependability attributes availability, reliability, and maintainability are chosen for closer inspection and are covered more thoroughly in the following sections. In addition to these attributes, dependability engineering includes system impairments, threats or factors, such as faults, errors and failures, and the means or methods to mitigate the factors, like fault prevention, tolerance, removal and forecasting [24].

Quality of software can be divided by its attributes into two groups, indirectly

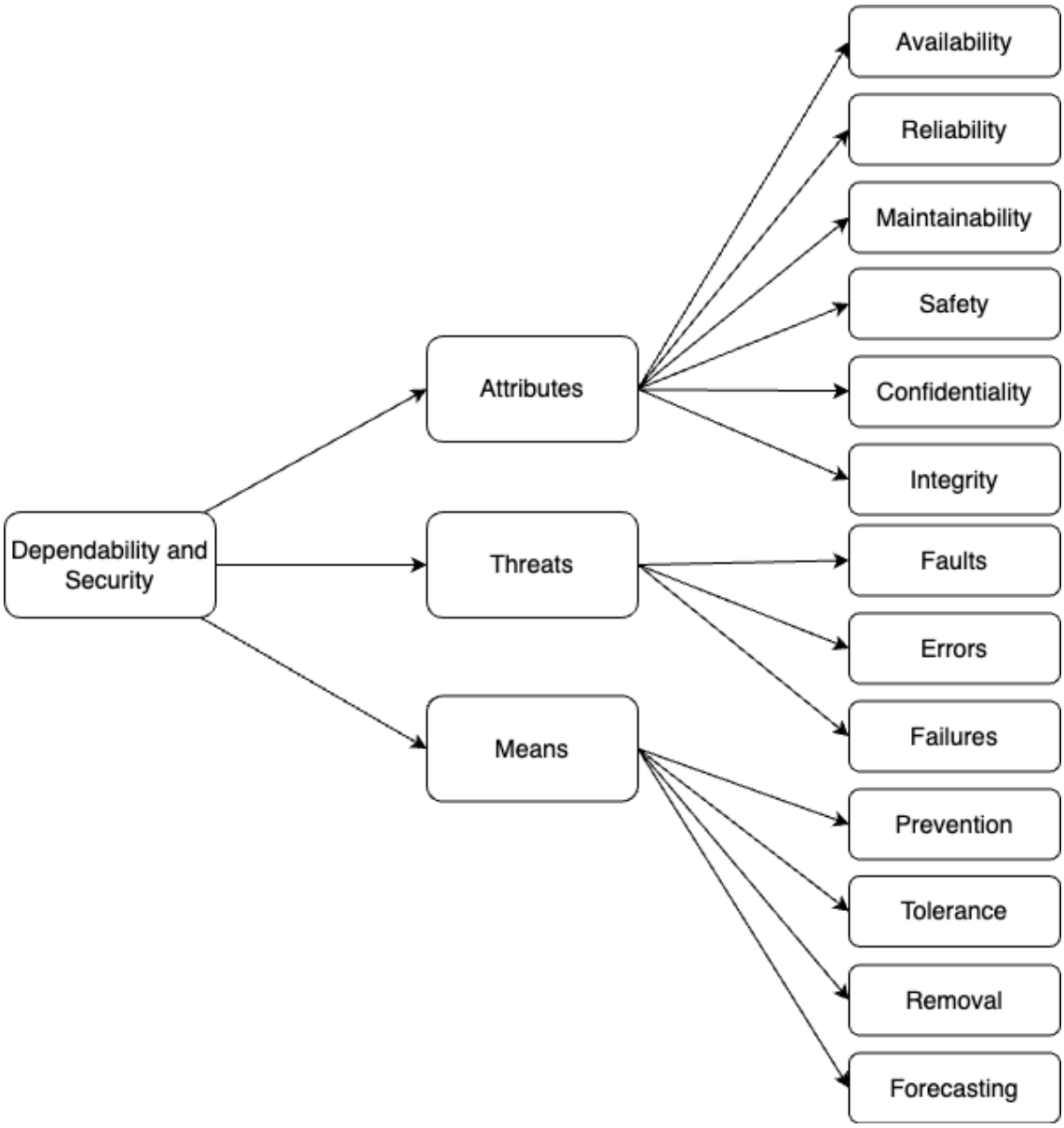

Figure 3. Taxonomy of dependability and security attributes after Laprie *et al* [28]

measurable external attributes and directly measurable internal attributes [25]. Dependability can be perceived by a user, in the form of how well they trust the system being able to provide relevant services in terms of maintainability, but reliability and availability can be directly measured over time using statistical approaches. Reliability, availability, and maintainability (RAM) -analysis is a common statistical approach to analyze the system's dependability [27]. Often RAM analysis is performed after the system is already finished, and the analysis is treated as an after-the-fact documentation exercise, and thus if the assessment indicates that changes are required, they ofter remain un-implemented. In Model Based System Engineering (MBSE) the RAM analysis can be – but often is not – integrated in the process, which allows it to affect the design choices [29].

4.3.1 Reliability

Reliability can be considered as a measure of continuity of service. The measure is the ability of a system to stay in operating condition over time. The measure varies between systems depending on the purpose and requirements of the system. Reliability of a system is often measured as its mean time to failure (MTTF) [24], or mean time between failures (MTBF) [29]. These can thus be considered as the expected operational life of the system itself [24].

The goals of software testing are both immediate and long-term. Testing of software attempts to help discover and prevent bugs in immediate effect and increase the quality, thus the reliability of the software in the long-term. Confidence in the software's reliability increases the quality of it [41].

4.3.2 Maintainability

Maintainability, as an attribute of system dependability can be defined as system's ability to be repaired and evolved [24]. In cloud computing, the CSPs have mitigated the burden of hardware and on-premise system maintenance for the users [26], but the maintenance of the developed software still remains with the developers.

As all software, ML products and data pipelines require thorough testing to mitigate issues across different aspects of system dependability [13]. Software maintenance has long been an expensive and resource-consuming process in software systems and especially in component-based systems. Software component refers to a part of the software that is nearly independent and replaceable which fulfills a certain objective [42]. As mentioned before, testing increases the reliability but also the maintainability of the software and each of its components. A thorough testing process also enhances the future testing and development [41]. Testing each component of the software, or unit testing, may simplify the future refactoring process by enabling sanity checks to verify the process. Unit testing can be complimented with integration testing which refers to the testing how each unit works together in the actual process.

4.3.3 Availability

While reliability as an attribute indicates the over-time continuity of a service, availability of a system is a measure of its readiness for usage. Availability can be measured as the limit of the probability that the system is functioning as it should at time t, as t approaches infinity [24]. Availability of a system can be calculated in following ways:

$$\alpha = \frac{MTTF}{MTTF + MTTR}$$

where MTTF is the reliability measure Mean Time To Failure, and maintainability measure MTTR is the Mean Time to Repair [24]. Or:

$$\alpha = \frac{MTBF}{MTBF + MTBR}$$

where MTBF is the reliability measure Mean Time Between Failures, and maintainability measure MTBR is the Mean Time Between Repairs [29].

4.4 Way of working

The design of each system is highly dependent on the way-of-working methodology implemented by the developing team and is often defined in the business organizational level [43]. Software development follows a cycle known as Software Development Life Cycle (SDLC), which in simple terms means the process of planning, developing, testing and deploying software in a sequence of phases while referring to the results of the previous phase [44]. The waterfall method of development life cycle has been covered thoroughly in past years and is thus not covered here as it has been surpassed by more iterative and faster time-to-market approaches such as Agile.

Agile development methodology bases its effectiveness on customer satisfaction through fast and early releases of useful software. In Agile, the goal is to constantly deliver new software for use in small batches. Through this iterative process the quality of the software can be monitored with the user feedback and continuous integration and continuous delivery (CICD) pipelines while the working software acts as the measure of progress. In terms of pro-activity, Agile methodology brings business people and developers closer in all required phases of the development and maintainability to achieve best possible outcome [44]. Often complementing the chosen methodology with development operations, or DevOps, the process of development becomes more stable and future proof in terms of the ability to maintain and upgrade the software.

4.5 Automation

To further increase the quality of the software, continuous integration and continuous delivery are often implemented to automate the process of productizing the software supply chain. Automating the process leads to faster feedback, better quality and faster time-to-market. Continuous integration is based on the code base being stored in a Source Control Management system (SCM), where each change triggers the build pipeline to produce a build artifact identically each time. Continuous delivery is based on the fact that the pipeline produces a stable binary that can be deployed to production at any time. Each build is automatically tested in the integration phase which again leads to better dependability of the system [43].

4.6 Monitoring

In previous sections the data quality and its dimensions as well as testing have been introduced. To achieve high quality in data pipeline, these must be monitored alongside system availability, performance and logs. Data quality measurement is a fundamental aspect of data-driven decision making. To increase trust in data and its driving capabilities in decision making, it is necessary to be able to measure and monitor it [35]. Data quality monitoring should be a cyclic process that is carried over continuously and automatically [37]. As discussed in this thesis before, data quality is a multi-dimensional concept and should be monitored as a quantified measures of these dimensions [35]. Data quality monitoring should be accompanied by alerts, which are triggered when the data fails to meet the quality expectations.

To ensure reliability and timeliness, and to minimize the downtime of a data pipeline, its operational health needs to be monitored. Besides alerts in data quality issues, the pipeline must alert when it fails completely. Monitoring the process should be facilitated by logging even when the process does not fail. By logging the events to a

centralized location, monitoring of the continuous process can be easier to maintain and debug and thus lead to less downtime [36].

4.7 Documentation

In an industrial scale, functioning data pipelines should be accompanied by sufficient data management. To create valuable assets from data, it should come with proper data governance and data lineage. Data lineage is used to trace the life cycle of the data through phases and transformations until it reaches the end user. The lineage helps the organization to understand how the data is sourced and what it is used for. Data governance provides the principles and framework of how the data should be managed in terms of security and regulation through roles, responsibilities and rules for the data. A high quality data pipeline must also conform to regulatory compliance, and data interoperability and transparency must be up to standards [45].

To meet these needs, the application and the data it produces must be documented properly. To help with data lineage and governance, data profiling can be used as a complementary element. The data itself can be documented using metadata explaining the schema in the dataset [45]. The schema should contain data characteristics such as column names, data types and descriptions explaining the attributes as well as indications of possible privacy issues.

Part II

Implementation

This section covers thoroughly the implementation of a high quality NLP data pipeline, or more simply, an inference application. The pipeline's primary purpose is to deliver various inferences from curated interaction texts between parties. 'Curated texts' in this instance indicates that the texts passed through the pipeline have been processed with necessary sanitation and language detection in a preceding pipeline to be available for processing further. The pipeline follows the high quality NLP data pipeline factors identified and listed in the literature review to maximize the pipeline's dependability, usability and efficiency.

Workflow of the pipeline is the following: Create various configurations for each individual use case for easy maintenance and up-scaling, gather the curated texts based on the use case configurations and perform the selected inference for each required text. At this point, the pipeline should produce sentiment and topic inference where it is required, while fulfilling the high quality factors identified before and reintroduced as follows:

1. **MLOps**

   The pipeline should be developed according to and follow an MLOps framework to it's needs.

2. **Evaluation and Validation**

   Data pipeline should provide evaluated and validated data with sufficient quality to meet the business needs.

3. Scalability

The pipeline must be scalable. The processing requirements should be adjustable and adding new use cases, models and texts should be possible and as simple as it can be.

4. Dependability

The system must be dependable. This means that the system should fulfill all the aspects of dependability deemed relevant in the literature review. The system must be available when it is needed, maintainable for repair and future development, and reliable.

5. Way of Working

The system must be developed in a way that it supports pro-activity and is not person-dependent. The system is built in a modern, incremental way to be able to provide fast results to business.

6. Automation

The system must be automatic with the timely specifications. The pipeline should be running automatically and reliably as specified.

7. Monitoring

The system should be monitored. The pipeline itself should have a way to be monitored for failures and performance issues. The output of the pipeline should be monitored for data quality errors and the data should be evaluated against given expectations. Errors in performance and data quality to should produce alarms.

8. Documentation

The system should be documented properly to increase the pro-activity and understandability of the system. Besides the application documentation, the

data should be documented with metadata. All data documentation should be stored in a centralized documentation space.

Each of these high quality NLP data pipeline factors set the foundation for multiple steps of the process and thus are extremely beneficial when trying to produce an industrial scale NLP data pipeline. The following chapters will explain the requirements and steps of producing such a system. Often within companies, the choice of Cloud Service Providers and other available services and their service type is limited to previously made choices by the company where the system is developed.

5 Requirements for a High Quality NLP pipeline

In this chapter, the utilized services and platforms are identified and explained. Focus here lies in the actual services that are used in this implementation, so other variants of each platform or service are left untouched.

5.1 Cloud Services

Modern data applications are highly dependent on cloud computing and other cloud services as they provide highly scalable infrastructure for development and production runtime. For this application, cloud services are utilized to their maximum capacity to produce a long-lasting solution that can fulfill the requirements of this application for the foreseeable future. As discussed in the literature review, cloud services include various useful resources that can be utilized when producing a high quality data pipeline.

As the system must be scalable, it should be built to be run on a cloud-based computing platform or service such as Databricks. With Databricks, the system can utilize open-source analytics engines like Apache Spark, which relies on paral-

lelism and distributed computing to allow more efficient large-scale data processing. Databricks allows the system processing units to be scaled to its needs while utilizing the cluster-computing feature of Spark. As model inference tasks on large data sets of texts can be substantially high load, it is important to be able to scale the processing needs accordingly. Utilizing parallelism leads to faster and more fault-tolerant processing of the data with the possibility of scaling the processing to the system's needs while distributing the data across the cluster nodes. In this implementation PySpark, the Python API for Spark, is used with the at the time latest Python version 3.10. Spark also offers an API for languages like Scala as well. As many other Cloud Computing platforms, Databricks offers an API to allow for local development of the pipeline while the actual software can be run in cloud and monitored through a radiator view of the workflow.

Another main Cloud Service required to produce this application is a storage service. This implementation is run in the AWS platform augmented with a Databricks environment. AWS platform provides an extensive selection of storage options from which the selection should be made according to the purpose of the system and its requirements. Implementation of this pipeline will utilize AWS S3 due to its highly scalable nature of object storage, multi-format support and cost efficiency.

For data cataloging, the S3 works seamlessly with AWS Glue, which is an ETL (Extract, Transform and Load) service that allows the data to be cataloged and transformed into suitable formats for analytics without needing further provisioning or server management. Both of these services are serverless, meaning the infrastructure is managed by the service, and automatically scalable according to the load, which reduces the operational overhead of the storage process. Correctly data cataloging the output data creates a sort of metadata repository of the data asset.

Cataloging the data with metadata and detailed descriptions improves the asset's discoverability, understandability, governance and reliability. In cataloging, the data can be assigned with a given schema, that indicates its structure including data types and attributes. This enforces the reliability of the asset.

The pipeline will need the option to apply ACID transactions to the data, which suits S3 as the storage service for its multi-format support which includes using Delta format. Utilizing these services from AWS platform leads to efficient querying of the output data using AWS services when the data is needed for analytics purposes.

As the system is based on AWS, it also utilizes several other resources from the platform. As listed in the high quality factors, the pipeline performance must be monitored. To achieve this factor, AWS CloudWatch is utilized to gather run-time logs in a centralized place for monitoring and debugging purposes.

As MLOps compliance is pursued, the system should comply with DevOps features such as an efficient CICD pipeline. A CICD pipeline enables a fast and controlled process of introducing features and fixes to the system. The code base should be version controlled and stored in a SCM system which integrates with required tools to build the new binary package of the application, the CI part of the pipeline. The system implemented will utilize Git source control in Bitbucket which is integrated with AWS services such as CodeBuild and CodePipeline, where automatic builds of the application are created upon updating the code base. Code building steps should always run required testing automatically to promote the reliability of the system. The CD, or continuous delivery part of the pipeline should allow easy releases to production environment after successful builds in previous phases. To

further automate and make the process more agile, the system should be enabled with an orchestration tool to manage automatic scheduled runs for the system, like Apache Airflow.

To utilize each individual NLP model, this system will utilize MLflow as its model management system. MLflow is an open-source platform which manages machine learning lifecycle comprehensively, including model deployment and model registry, where each model's lifecycle can be managed. MLflow allows this application to utilize any version of a registered model through its version control.

5.2 Data Quality and Documenting

Data quality should be monitored and documented to achieve better understanding, confidence and transparency of the data and system quality. In this implementation, Great Expectations (GX) is used to validate and evaluate the data. GX provides an end-to-end framework through expectation based validations of data. The expectations should be set so that validation errors terminate the run and alert the developing team, while evaluation is monitored in long-term to view how the data develops through time. While AWS Glue provides cataloging for the data stored in S3, we can utilize a more accessible centralized place for the data quality and metadata documentation like Datahub. Datahub is a unified metadata platform, that provides a centralized location for data assets for increased governance, discoverability and monitoring. Results of the data validation will be displayed in the data asset entry inside Datahub. Storing the asset metadata and descriptions in an accessible location allows transparency to business users as well. Data evaluation results are not required to be transparent to business, but to the developers themselves. For data quality monitoring through data quality dimensions the software observability platform Dynatrace is used.

6 Implementation

This chapter explains how the implementation is built for a High Quality NLP Data Pipeline within the scope of the high quality factors identified in this thesis. Figure 1 describes the workflow of the application on a high level but the primary purpose of the application is to provide inference results in terms of sentiment and topic analysis, with a high standard of data quality and validation. The application should provide suitable inference for each use case according to a configuration. The specific workflow and utilization will be explained in this section. The application utilizes services mentioned in section 5.

The development is performed in an Agile way, by iterating each step to maximize the efficiency of each development process. This section does not explain each iteration of the application, but the latest and released version of it and the development should be continued in a pro-active manner after the scope here is fulfilled.

The application is developed locally while utilizing the cloud computing of Databricks and its integrations with AWS. The system is intended to be task-based, and run each individual task concurrently or sequentially as visualized in Figure 4. Each task is defined by its *entrypoint*, which is the main function of each task and orchestrates the task's workflow. As described in Figure 4, the application contains six tasks, each of which is run dependently on the previous' succession, excluding topic and sentiment inference, which are run in parallel.

To create a dependable data product, the application code is thoroughly unit tested. Testing is focused on transformations and is expected to have at least 80% of testing coverage throughout the application code. Each unit test is run in the CICD pipeline alongside other code quality checks, which integrates the source code with

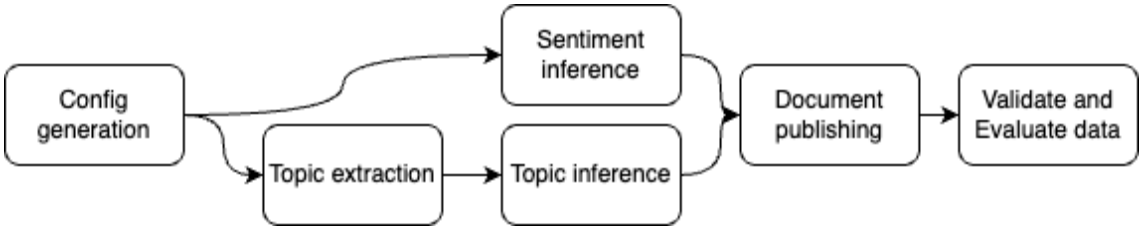

Figure 4. Task entrypoints

necessary services in the AWS platform to produce an automated and scheduled data pipeline. Each task in the application provides monitoring logs for each crucial action. This way the debugging process and monitoring of the performance is made simpler and more transparent, as the logs are published to AWS CloudWatch.

6.1 Source data

To produce relevant and reliable inference results, the source data must be curated to match data quality standards. Curating individual text sources is not in the scope of this thesis' implementation, but the process of curation is covered to support the data quality aspect of the thesis. The curated source data of this application is gathered from multiple sources and each source can contain texts of different types. Source texts can be chat messages, phone calls, web messages, feedback texts or multiple other types of interaction texts. Interaction can happen in many forms and in various systems, but all the source data must be in text format to be used for inference. In this case, all source data is already in text format, but phone calls are transcribed, and the quality of the transcriptions is validated to evaluate the quality of the service.

Each data source goes through a process of language detection and sanitation. Language detection is performed to maximize the effectiveness of the models, as they are tuned for specific languages. Sanitation removes all Personally identifiable Information (PII), and vulgar or otherwise inappropriate language from the texts. All

sources are processed in a single application utilizing similar services as the NLP pipeline implemented in this thesis. After processing, the texts are gathered into a single data asset with identifiers for each text and its source accompanied by metadata. The source data quality is validated to meet the required expectations.

6.2 Application configuration

The first step to manage an application required to perform inference using multiple models and to be able to scale up as new models and use cases occur, the system is managed through an inference configuration. The purpose of the configuration, displayed in Table I and Table II, is to easily manage what the pipeline does without having to affect the actual code, thus increasing the agility of the software by allowing fast feature releases. Adding new use cases and updating models should require as little workload as possible and the application code should react to changes in configurations accordingly. The inference configuration contains the use case name for traceability – to give a logical identifier for what the inference is about. The use case name acts as an identifying part in the output, i.e. what the pipeline has done to that specific text. Each use case must have a version identifier, which is used to update the data to tell the user which of the inference results are the latest and most relevant. The use case version should then be increased when something changes in the workflow. For example if a new model is used for the same texts, the use case version is incremented to indicate that these results have a distinct difference to previous versions. Version control of the use cases allow for better data management, as old versions can be deleted or archived in the long-run. The task of archiving and deleting inference data based on use cases identifiers is described further in the following sections.

The configuration holds information about the model that should be used in each

Use case name	String
Use case version	String
Model name	String
Model version	String | None
Inference type	Literal["sentiment", "topic"]
Filter	pyspark.sql.Column

Table I. Inference configuration data types

use case. As models are fetched from MLflow model registry, the configuration takes in the model's name and the version. The configuration allows the model version to be optional. This is done to support easy model management, as the software should use the latest production model if version is not specified explicitly. By allowing optionality to versioning, the application can automatically use the latest version if it is intended to do so. The option to specify the version adds robustness to the system e.g. for only using versions that are proven efficient. The full output data set contains both use case identifiers and model identifiers to achieve better transparency for the inference results.

Additionally, the configuration takes in the inference type, a literal of 'topic' or 'sentiment'. This guides the inference pipeline to perform the desired tasks. The inference pipeline will be further elaborated later on in this section. Filter attribute in the configuration allows the system to select the correct texts from the curated combined source of various types of text data. As the system is developed using Python and PySpark, the filter is given as a PySpark DataFrame Column filter condition to collect the correct texts based on their category. A Column condition was chosen to allow better freedom for the developers to get as specific texts as required. The Column condition is also scalable, and can accept extremely complicated filtering conditions.

The configuration is included in the application as a Python *TypedDict* namespace for easy integration to the software itself. The configuration is stored in its own module inside the application code and utilized similarly to any Python module. The application's configuration parser task reads the given set of configurations and validates them to being properly defined and that they do not contain duplicate configurations. Duplicate configurations may lead to issues within the inference output. Each configuration is then turned into tabular format and upserted (update and insert) into Delta table in S3 and cataloged with metadata and schema as an AWS Glue table entry. The output dataset is assigned with a predefined schema, which helps to validate the output. PySpark DataFrames can be assigned with a StructType schema, that contains the attributes, their types and optional metadata. In this application, all output tables use a predefined schema in JSON format, which can be converted to a StructType and assigned to each table. Using ACID transactions, new versions of use cases are inserted into the Delta table with new timestamps. Simultaneously, older versions of the same use case are updated to archived status. By having the boolean attribute 'archived', the application can easily manage which use cases are currently relevant and which are not for the application of inference. These configurations are later used by the application to gather instructions for the pipeline as well as to trace back the inference results to the correct use case.

6.3 Topic extraction

The second task of the pipeline is extracting topics from key-word topic models. Each of these models contain a set of key-words divided into categories: main category, sub category and topic. Each of these are inserted into a Delta table in S3 along with the model's name and the version. Topics are extracted from the model

Use case name	chat-sentiment-classification	phone-topic-classification
Use case version	1	2
Model name	chat-sentiment-model	phone-topic-model
Model version	1	2
Inference type	sentiment	topic
Filter	col("text_type")==("chat")	col("text_type")==("phone")

Table II. Inference configuration example

Name	**Id**	**Parent id**	**Level**
Sickness	1-1-1-1	null	mainCategory
Non-infectious	2-2-2-2	1-1-1-1	subCategory
Allergies	3-3-3-3	2-2-2-2	topic

Table III. Topic extraction from keyword topic models (Mock data)

by fetching each configured topc model separately from MLflow. The models are loaded as Python models and unwrapped using the `pyfunc` module of MLflow. The topics are then de-duplicated and inserted as individual rows into the table with identifiers to their parent category as described in Table III, where the data is hypothetical and adjusted to the domain of health care. This way each topic and sub category can be traced to its main category while creating a sustainable format to store each topic. These topics are later on used to match topic hits from the inference step. The purpose of this is to be able to validate topic inference hits based on the topics available to the model.

6.4 Inference pipeline

The third task is to perform inference on the desired texts. This task utilizes all previously introduced parts of the pipeline by utilizing the inference configuration to collect relevant texts, model, use case and the inference type for each inference task.

The user splits the configurations to topic and sentiment inference tasks which are then run concurrently to maximize the efficiency of the system. Both tasks utilize the same application code, but are using different configuration variables to achieve separate behavior. After the topic and the sentiment tasks have been initiated, the pipeline begins to run inference on each text by filtering the source dataset according to the inference configuration. The filter attribute in the configuration is used to gather the use case specific texts from the source. The input is read in as a batch, but can be upgraded to be a streaming service in the future of development.

The pipeline must be able to perform the task on only texts it has not seen before and on the full dataset in case a new use case has been added with new model configuration. By default, the application expects to process new texts and the distinction is done by comparing the existing results from the same use case and their timestamps. This leads the application to process all texts when the use case has not existed before. At this point, the pipeline is ready to perform the first use cases for topic and sentiment inference. In each task, the source data is filtered according to the inference configuration after the application has identified the texts to process and validated that there is new source data available. The entrypoint to these tasks initializes a *Config* object (detailed in the Figure 5 for the inference with all required attributes including the model name and version, the inference type, and details about where the results are to be stored. This type of dataclass helps with passing the correct parameters and allows for better attribute management inside the main class, as this way the attributes are type annotated and the complexity of the class remains stable through code quality checks in the CICD pipeline. As visualized in Figure 5, the Pipeline class acts as the main class, which is inherited by the two subclasses handling the specific tasks relevant for the type of inference that is being run. The main class does the required processing needed by both pipelines.

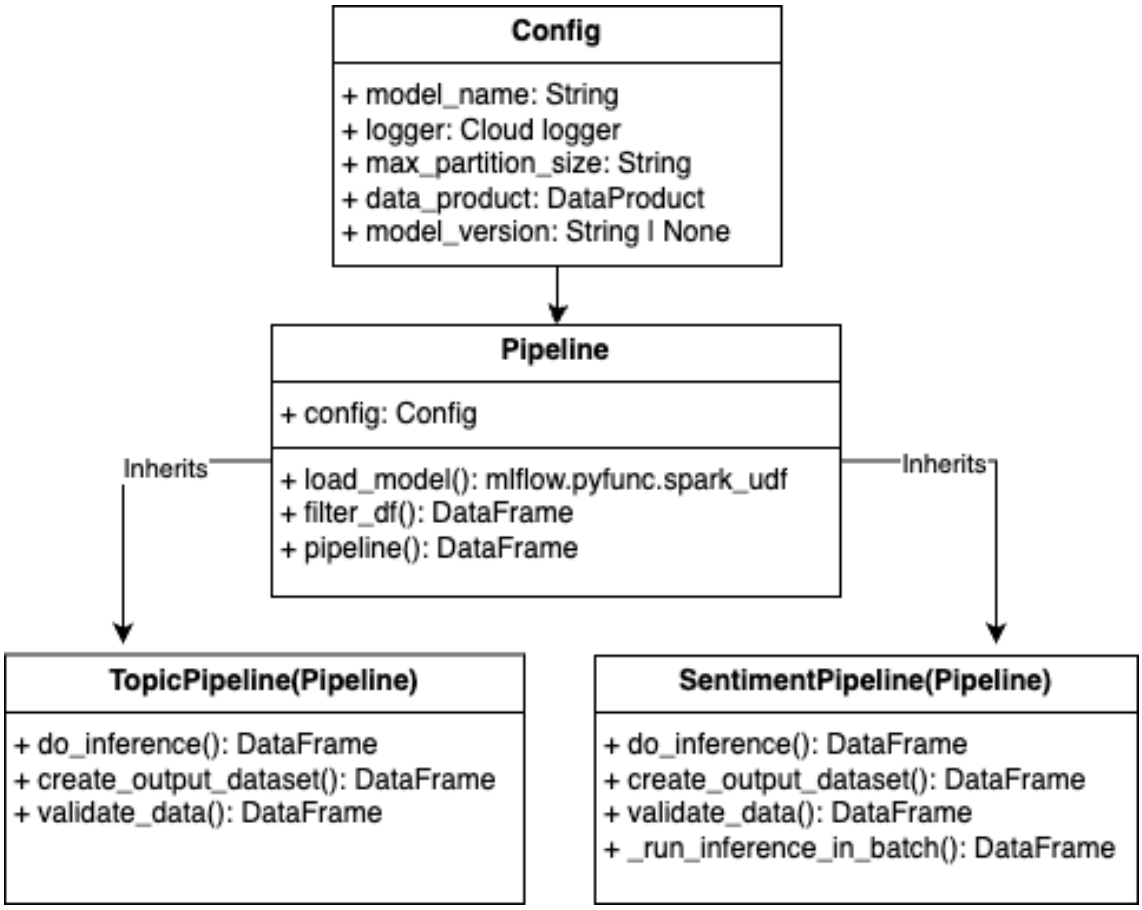

Figure 5. Inference pipeline

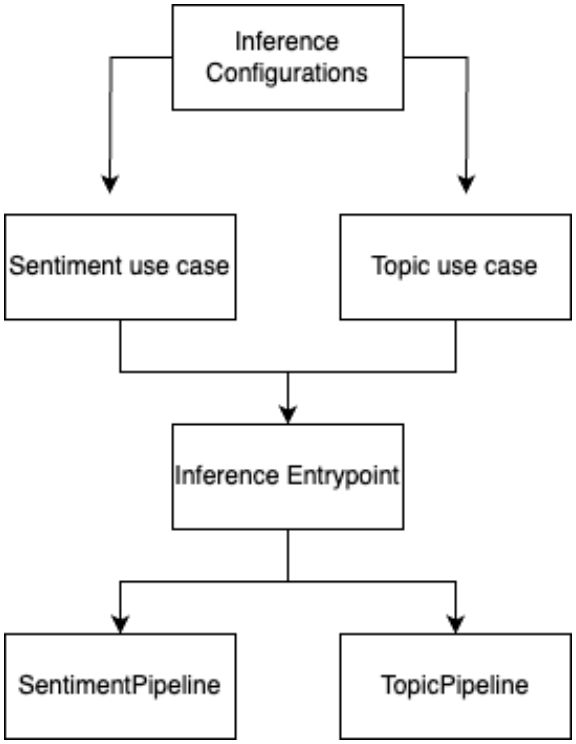

Figure 6. Pipeline initialization

By creating an object oriented flow, the maintainability and reliability of the system is increased as new pipelines can be included without breaking the existing system. The desired subclass is initialized in the entrypoint with the Config object based on the inference type. In the Figure 6, the initialization of each pipeline is visualized. Based on the configurations, the entrypoint initializes the correct pipeline which is used to perform the inference.

When the correct subclass is initialized, the application begins to perform common tasks inside the main class. As pictured in the Figure 5, the main class *Pipeline* holds three methods to perform these, as each method is designed for a single purpose. When the class is initialized, the application automatically loads the required model from MLflow's model registry using its method `load_model()`. The application loads the model specified in the inference configuration for the type of texts, or chooses the latest production model if the version is not explicitly specified. Sync-

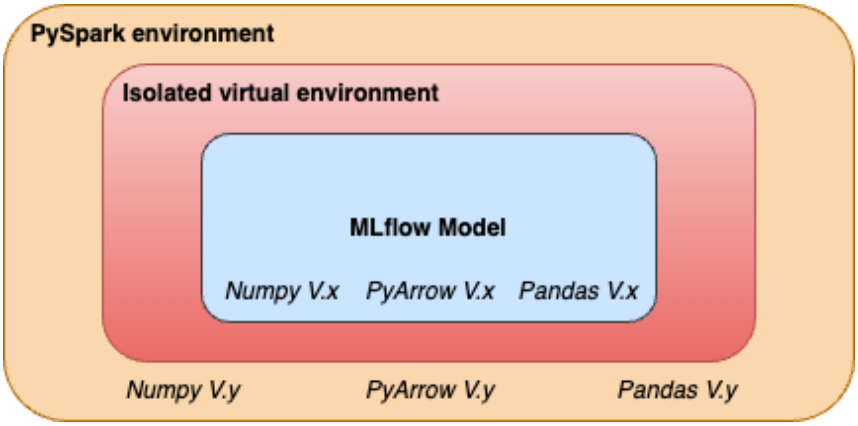

Figure 7. Isolated MLflow model with Spark UDF

ing dependencies with model training, serving, and the application of the model can often be an issue with inference applications. To mitigate dependency syncing black-holes, the models are run in isolation, thus mitigating the dependencies from each. As this application runs on Spark, the Spark's User Defined Function (UDF) can be utilized to produce inference on a large scale DataFrame. MLflow's Spark UDF, as viewed in Figure 7, creates a virtual environment that is completely isolated from its containing environments, thus allowing the dependencies to be model specific.

The *pipeline* -method of the main class orchestrates the rest of the process by applying the required filtering to each DataFrame with the method `filter_df()` and applying the correct inference pipeline from the sub classes. The texts are cleaned of non-parseable elements by the models, such as special symbols, special characters and empty strings. The DataFrame is repartitioned according to the size of the data to maximize the effectiveness of distributed computing provided by Databricks and Spark while performing inference. At this point, the texts are ready to be processed. The application then continues to call the rest of the methods from the relevant subclass to conclude the process. The workflow of each inference type is explained in the following parts.

6.4.1 Topic inference

Topic inference is applied to desired texts by applying the Spark UDF that has been loaded in its own isolated environment. Executing the UDF creates an array column to the original texts DataFrame, containing the topics and metrics for each text. To create an immutable table schema, each topic match is assigned to their own row with the text's id. While this increases the size of the table vertically, the schema, or the structure of the table remains the same even though different models contain different topics. Each topic hit is collected and matched with the Topic table created in the topic extraction phase described in section 6.3.

Each row is populated with metadata related to the inference, such as archival status and inference timestamp. As partitioning only positively affects the inference workload, which is distributed to available nodes and their cores, the result data is repartitioned to a reasonable partition sizes which suit the reading process better. As Spark practices Lazy evaluation, an execution model of only performing tasks when actually needing to, so far the application has only created a an execution plan to the driver node. At this point, the execution plan must be applied to make debugging and later stages easier by limiting the steps where issues could come up. The DataFrame is evaluated so that it does not contain missing inference results or rows. By applying these operations, Spark must evaluate the data and thus execute the queued operations. After the validation step, the DataFrame is returned to the entrypoint from which the inference process was initiated.

6.4.2 Sentiment inference

Similarly to topic inference, the sentiment inference pipeline uses the same method of applying inference on the full set of data. As the sentiment models used by this application are more complex than the key-word based topic models, applying the

sentiment requires more thorough processing of the input data. Some Bert based language models limit the input size to a certain amount of tokens. In this case, our limit is 512 tokens. Put simply, a token is similar to a word, depending on the language, but the actual tokens can be calculated using a tokenizer. To be able to get consistent results, the input texts must be tokenized and split to lengths that are less than the limit of these Bert based models. Each row of the input data is tokenized and split to even sizes under the limit and assigned to their own rows with common identifiers to match them after the inference. The limit is reduced to 500 from 512 to mitigate possible errors in tokenization.

As the sentiment models in this case are more complex, they may run into issues with large quantities of texts. As the input for the model can be a few thousand texts, or up to tens of millions of texts, the process must contain fail safes to succeed consistently. To be as efficient as possible, the application begins the inference process for the whole input dataset containing the specific type of texts. Often with too large inputs, or inputs containing difficult-to-parse texts, the process might fail and thus the whole task fails and must be restarted. The application is set up with error handling that prevents the system from failing in case this happens by catching such errors. In case the system catches an error in this phase of the process, it retries the inference run in batch mode. To achieve batch mode, the input data is split to even sizes, and the UDF is run separately on each batch. As mentioned, Spark is lazy evaluated, and the process is not run until the output is required. In batch mode, each batch is written to a sink in S3 for temporary storing and thus evaluating the inference execution plan and ensuring each batch has succeeded in the process. This fail safe is concluded with the deletion of the sink after the full set of batches is finished, and all the texts are available for further processing, or in case the system fails.

After each split text has been processed and given the model output and metrics, the chunks are combined using the IDs. As each chunk contains separate results from the inference, the weighted average, where the bias is on negative sentiment, is assigned as the sentiment of the text. The output dataset populated with metadata, validated for integrity and repartitioned similarly to topics. The resulting evaluated data is returned back to the inference entrypoint, where the process is continued.

6.5 Storing the results

After each type of text has been processed by the relevant inference pipeline, the results are stored in AWS S3 and cataloged to Glue. To keep the data consistent, each dataset is assigned with a predefined schema, which mandates the structure and attributes in terms of data types and metadata. Schemas are defined in JSON format and define each attribute of the dataset as a field, as shown in Table IV. A field is comprised of the attribute name, its null-ability, data type and metadata with a non-optional description and optional terms of use or indication of privacy, such as PII. By defining the schema, output can be validated to be in the correct format and to contain all data it is required to contain. Each of the output tables created by the application is assigned with a predefined schema to validate all outputs.

As mentioned, the write operations utilize Delta format's ability to be changed. Each time a table is saved, the function performing the operation is given conditions about when to update and the unique keys it uses to match the data that should be updated. The application checks that the database and table exist in Glue, and creates them in case they do not exist. Databases in Glue must be explicitly created before they can be used as a catalog layer for data stored in S3. The unique keys, the identifying attribute or attributes from the dataset, are matched

Field	
"name"	"sentiment"
"nullable"	false
"type"	"string"
"metadata"	{ "comment": "Sentiment of the text" }

Table IV. Schema format

with an existing Delta table in AWS. When a text is matched between the incoming and the existing data, the application checks the use case of each text. If the use case is different in the incoming data, the row's attribute *Archived* in the existing table is updated with boolean True value and given a timestamp of the execution to indicate when the row has been archived. The incoming row is then inserted into the table as a new row with False value in the Archived attribute. This is done to keep track of the use cases and to indicate which text is processed with what model and which is the latest inference result. The latest rows are stated as not archived while old rows are in archived state. Each table is stored in a similar form using Delta's ability to be modified. The ability to do this enables it to always have clear indication of which data is relevant and simultaneously to have access to old data for comparison and other analytics.

After the table in question has been stored in AWS, its optimized and vacuumed. Optimizing the delta table is done using z-ordering. Z-ordering is a method of re-organizing data to speed up queries in the future by skipping specific files in the file system and thus reducing the time of storage to memory transfer. To further improve the storing, the table is vacuumed, or cleaned of unnecessary checkpoints from the past. In large datasets like these, the optimization of the queries for later analytics can improve the downstream user's experience immensely. The storing is done in batch mode, as the application does not currently receive streaming data.

6.6 Documentation

The application code is accompanied by a thorough description of the application. This documentation is created to maximize discoverability, governance and transparency of the data asset in the company. In addition to the written documentation about the application, the data assets it produces are documented as well and published into a centralized data asset location called Datahub. The data assets this application produces are delivered with the metadata from the predefined schemas which are used to catalog the data in AWS Glue. After each data asset of this application is documented, the assets are given a lineage illustration. The lineage describes the life cycle of the data through its transformations. When the source data has been documented with its lineage, the documentation will help with understanding where the data has come from and where it is used. Adding lineage in a centralized cataloging location enables the possibility of viewing the downstream access to the produced datasets as well as to further improve the data governance and transparency. As the data asset entries in Datahub contain their lineage, data descriptions and privacy terms provided by the schema, they can also be enriched with the data quality of each asset. How data quality is evaluated in this application, is explained in the following section.

6.7 Data quality validation and evaluation

Data quality is a key part of creating any high quality data pipeline. In this application, data quality is ensured using the Great Expectations framework, which allows for validating and evaluating each attribute of the dataset separately based on given expectations. Data quality is divided into two for this application: Data quality validation and data quality evaluation. These are distinguished by the severity of a failure, as a validation failure terminates the run and alerts the maintaining personnel that something needs to be done immediately, whereas evaluation acts as

Completeness	expect_column_values_to_not_be_null
Consistency	expect_column_values_to_be_between
Validity	expect_column_values_to_be_of_type
Timeliness	expect_column_values_to_be_between
Integrity	expect_table_columns_to_match_set
Uniqueness	expect_column_values_to_be_unique

Table V. Examples of data quality dimensions with expectations

a long-term observation tool which should be monitored to keep track of how the data quality evolves.

Each produced data asset is both validated and evaluated by predefined expectations about the data. As GX is expectation based, each dataset is given heuristic and data specific expectations, which they need to meet to pass the validation. Most of the expectations used can be linked with the data quality dimensions introduced earlier in this thesis. Quality expectations and the data quality dimension are linked for this application as described by examples in Table V, where each expectation represents a part of a dimension, and is used to enforce the quality requirements of that dimension.

All assets are passed through a schema validation, where the table's predefined JSON schema is transformed into the PySpark supported StructType format, and the table is validated based on attribute names, their type and their nullable status. These validations address the validity, completeness and integrity dimensions, and validates that no non-nullable columns contain null values, all data types are as expected and that attributes are correctly defined and present. To further address the completeness dimension, each table is validated based on their row count. Each table is given a row count range based on previous data, which it needs to satisfy.

The consistency dimension is fulfilled by validating each table's distinct entities to be in a set, or between a set of values. For example, the topic dataset should contain the level of each topic and they should be in the set of mainCategory, subCategory or topic. Inference output metrics are validated to be between the model's validation limits and not to produce results where the reliability is under 50%, as this should alert the maintainers to investigate the issue. Model metrics are also validated using the mean, standard deviation and median values to maximize the consistency of the model output. Like consistency, timeliness of the data can be validated by setting limits for timestamps.

All output dataset rows contain identifying primary keys, as well as foreign keys to relevant tables. As the row identifying keys must be unique, the column is validated to only contain unique values. If any of these validations fail, the process is terminated and alerts are raised.

Validation results are published to Datahub alongside each data asset to represent the asset's data quality. By publishing the results to a centralized location, it increases trust in the data by visibly indicating its quality.

Alongside validation, the data is also evaluated. The evaluation is produced similarly to validation through expectations. With evaluation, the purpose is necessarily not to pass the check, but to set the expectations to measure quality against the highest conceivable standard. The evaluation uses similar ways of validating, but the thresholds of each are tightened. Doing this allows the monitoring of changes in model performance, data quality and quantity of data processed. The evaluation metrics are published to Dynatrace on an asset level for the maintainers to monitor.

Each asset contains a dashboard with each data quality dimension metric illustrating how well each dimension is fulfilled. With data evaluation, it is crucial to set the standards high to be able to distinctly discover changes in quality.

Part III

Conclusions

This thesis sought to provide an overview of existing research and information relevant to high quality NLP data pipelines as well as implementing one based on the factors identified in the literature review. By examining each aspect of the literature review through the research questions declared at the opening section of this thesis, the goal was to identify the steps required to produce an NLP use case from raw texts, identify the key factors which enable producing a high quality NLP data product, and how the data can be evaluated and validated.

The literature review identified key aspects of building a high quality NLP data pipeline with a focus on the quality of data. A total of seven key factors were identified and introduced to fulfill the needs of these pipelines as required by *RQ2* (*What are the key factors of a High Quality NLP Data Pipeline?*). The literature review thus produced the quality factors which were to be satisfied in the implementation of the pipeline. The identified factors were scalability, dependability, way of working, automation, documentation, data quality evaluation and validation, and monitoring. Each of these factors was researched and provided with context on how each can affect the quality of an NLP data pipeline. These factors were constructed to support an end-to-end capable pipeline while taking a stance on all aspects of the development from the ways of working to managing good data governance.

The literature review introduced common NLP applications such as topic modeling, sentiment analysis and summarization, but also gave a touch point to Large Language Models. While the review gave insight to the key factors of the NLP pipelines, it also provided an explanation on how these applications can be utilized

to produce an NLP use case in terms of *RQ1* (*How raw texts can be transformed into actual NLP use cases?*). In the implementation section of this thesis, *RQ1* is more thoroughly explained through the actual implementation of the application and given concrete ways of creating an NLP use case from texts.

Data quality and its evaluation and validation were highly relevant in the literature review. Alongside how data should be evaluated and validated, a total of nine data quality dimensions were extracted from various literature sources, which can be used to validate and evaluate NLP data pipelines' output. Alongside the data quality dimensions, ways of validating the input data for these pipelines were thoroughly researched, thus concluding *RQ3* (*How to validate and measure data quality in an NLP data pipeline?*).

Based on the research, a High Quality NLP data pipeline performing sentiment and topic inference on multiple types of texts was implemented. The implemented application followed the factors extracted from literature to maximize its quality. As these pipelines should be developed iteratively, there will be room left for improvement. Development should be continued on the application's data validation and evaluation by further filling each implemented data quality dimension with new expectations and performing the validation in an earlier stage of the process. While the dependability of the system is verified with thorough unit testing and monitoring, it could be improved with quality integration testing and thorough dependability analysis. The application currently works in batch mode with scheduled orchestration, but should at some point be supporting streaming data when input data is available as a stream.

Overall, this thesis has provided a framework for building high quality NLP pipelines

on an industrial scale. Due to the limitations of the scope of this thesis, some issues were left open to more research or better implementation.

References

[1] Sri, M. (2021). NLP in customer service. In Practical natural language processing with Python. Berkeley, CA: Apress.

[2] Iroju, O. G., & Janet, O. (2015). A systematic review of natural language processing in healthcare. International Journal of Information Technology and Computer Science, 8(8).

[3] Nadkarni, P. M., Ohno-Machado, L., & Chapman, W. W. (2011). Natural language processing: An introduction. Journal of the American Medical Informatics Association, 18(5), 544–551.

[4] Manning, C., Raghavan, P., & Schuetze, H. (2008). Introduction to information retrieval. Cambridge, UK: Cambridge University Press.

[5] SCIE-99, & SCIE-99. (1999). Information extraction: towards scalable, adaptable systems (1st ed. 1999.). Springer. https://doi.org/10.1007/3-540-48089-7

[6] Poibeau, T. (2013). Multi-source, multilingual information extraction and summarization. Springer.

[7] Le, N.-T., Servan, C., Lecouteux, B., & Besacier, L. (n.d.). Better Evaluation of ASR in Speech Translation Context Using Word Embeddings. Interspeech 2016 Proceedings.

[8] Bonaccorso Giuseppe. (2018). Topic Modeling and Sentiment Analysis in NLP. In Machine Learning Algorithms (pp. 1–1). Packt Publishing.

[9] Eshima, S., Imai, K., & Sasaki, T. (2024). Keyword-Assisted Topic Models. American Journal of Political Science, 68(2), 730–750. https://doi.org/10.1111/ajps.12779

[10] Ho, V. A., Nguyen, D. H.-C., Nguyen, D. H., Pham, L. T.-V., Nguyen, D.-V., Van Nguyen, K., & Nguyen, N. L.-T. (2019). Emotion Recognition for Vietnamese Social Media Text.

[11] More, A., Dalal, V., Tuba, M., Joshi, A., Akashe, S., Akashe, S., Joshi, A., & Tuba, M. (2021). Graph-Based Multi-document Text Summarization Using NLP. In Advances in Intelligent Systems and Computing (Vol. 1270, pp. 177–184). Springer Singapore Pte. Limited. https://doi.org/10.1007/978-981-15-8289-9_17

[12] Jusoh, S. (2018). A study on NLP applications and ambiguity problems. Journal of Theoretical & Applied Information Technology, 96(6).

[13] Salama, K., Kazmierczak, J., & Schut, D. (2021). Practitioner's guide to MLOps: A framework for continuous delivery and automation of machine learning (White paper).

[14] Kreuzberger, D., Kühl, N., & Hirschl, S. (2023). Machine learning operations (MLOps): Overview, definition, and architecture. IEEE Access, 11, 31866–31879.

[15] Sculley, D., Holt, G., Golovin, D., Davydov, E., & Phillips, T. (2015). Hidden technical debt in machine learning systems. In Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2 (NIPS'15) (pp. 2503–2511). Cambridge, MA: MIT Press.

[16] Ali, R. & Renals, S. (2018). Word Error Rate Estimation for Speech Recognition: e-WER.

[17] Reiter, E. (2018). A structured review of the validity of BLEU. Computational Linguistics, 44(3), 393–401. doi: https://doi.org/10.1162/coli_a_00322

[18] Sehgal, N. K., & Bhatt, P. C. P. (2018). Cloud computing: Concepts and practices. Cham, Switzerland: Springer. URL: https://utuvolter.fi/permalink/358FIN_UTUR/1cgjm0n/alma9920622405405971

[19] Mell, P., & Grance, T. (2011). The NIST definition of cloud computing. Special Publication 800-145.

[20] Antonopoulos, N., & Gillam, L. (2017). Cloud computing (2nd ed.). Cham, Switzerland: Springer. URL: https://utu-finna-fi.ezproxy.utu.fi/Record/volter.1882040

[21] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: Cluster computing with working sets. In 2nd USENIX workshop on hot topics in cloud computing (HotCloud 10).

[22] Amazon AWS Documentation(13.12.2024). https://aws.amazon.com/what-is/apache-spark/#:~:text=Apache%20Spark%20is%20an%20open,against%20data%20of%20any%20size.

[23] Priyadarshini, S., Sawant, T. N., & Yadav, Y. G. B. (2024). Enhancing security and scalability by AI/ML workload optimization in the cloud. Cluster Computing, 27, 13455–13469. https://doi-org.ezproxy.utu.fi/10.1007/s10586-024-04641-x

[24] Barbacci, M., Klein, M. H., Longstaff, T. A., & Weinstock, C. B. (1995). Quality attributes.

[25] Alsolai, H., & Roper, M. (2020). A systematic literature review of machine learning techniques for software maintainability prediction. Information and Software Technology, 119, 106214. https://doi.org/10.1016/j.infsof.2019.106214

[26] Song, C., & Sohn, Y. (2022). The influence of dependability in cloud computing adoption. Journal of Supercomputing, 78, 12159–12201. https://doi.org/10.1007/s11227-022-04346-1

[27] Tsarouhas, P. (2017). Statistical techniques of reliability, availability, and maintainability (RAM) analysis in industrial engineering. In Diagnostic techniques in industrial engineering (pp. 207–231). Switzerland: Springer International Publishing AG. https://doi.org/10.1007/978-3-319-65497-3_8

[28] Avizienis, A., Laprie, J. C., Randell, B., & Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing, 1(1), 11–33. doi: 10.1109/TDSC.2004.2

[29] Diatte, K., O'Halloran, B., & Van Bossuyt, D. L. (2022). The integration of reliability, availability, and maintainability into model-based systems engineering. Systems, 10(4), 101. https://doi.org/10.3390/systems10040101

[30] Majumder, G., Pakray, P., Gelbukh, A., & Pinto, D. (2016). Semantic textual similarity methods, tools, and applications: A survey. Computación y Sistemas, 20(4), 647–665.

[31] Hyunjin, C., Judong, K., Seongho, J., & Youngjune, G. (2021). Evaluation of BERT and ALBERT sentence embedding performance on downstream NLP tasks.

[32] Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP).

[33] Gao, M., Hu, X., Ruan, J., Pu, X., & Wan, X. (2024). LLM-based NLG evaluation: Current status and challenges. arXiv preprint arXiv:2402.01383.

[34] Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., & Stoica, I. (2023, October). Efficient memory management for large language model serving with paged attention. In Proceedings of the 29th Symposium on Operating Systems Principles (pp. 611–626).

[35] Ehrlinger, L., & Wöß, W. (2022). A Survey of Data Quality Measurement and Monitoring Tools. Frontiers in Big Data, 5, 850611–850611. https://doi.org/10.3389/fdata.2022.850611

[36] Narayanan, S., Maheswari, S., & Zephan, P. (2024). Real-Time Monitoring of Data Pipelines: Exploring and Experimentally Proving that the Continuous Monitoring in Data Pipelines Reduces Cost and Elevates Quality. EAI Endorsed Transactions on Scalable Information Systems, 11(4).

[37] Sebastian-Coleman, L. (2013). Measuring data quality for ongoing improvement: A data quality assessment framework (1st ed.). Waltham, MA: Elsevier.

[38] Mahanti, R. (2018). Data quality: dimensions, measurement, strategy, management, and governance (1st ed.). Milwaukee, Wisconsin: ASQ Quality Press.

[39] Dang, V. M. H., & Verma, R. M. (n.d.). Data quality in NLP: Metrics and a comprehensive taxonomy. In Advances in Intelligent Data Analysis XXII (pp. 217–229). Cham: Springer Nature Switzerland. doi:10.1007/978-3-031-58547-0_18.

[40] Wang, J., Liu, Y., Li, P., Lin, Z., Sindakis, S., & Aggarwal, S. (2024). Overview of data quality: Examining the dimensions, antecedents, and impacts of data quality. Journal of the Knowledge Economy, 15(1), 1159–1178. doi:10.1007/s13132-022-01096-6.

[41] Chauhan, N. (2010). Software testing: Principles and practices. Oxford University Press.

[42] Matinlassi, M., & Niemelä, E. (2003). The impact of maintainability on component-based software systems. In 2003 Proceedings 29th Euromicro Conference (pp. 25–32). IEEE. https://doi.org/10.1109/EURMIC.2003.1231563.

[43] Van Merode, H. (2023). Continuous integration (CI) and continuous delivery (CD): A practical guide to designing and developing pipelines (1st ed.). Apress. https://doi.org/10.1007/978-1-4842-9228-0.

[44] Pathania, N. (2016). Learning continuous integration with Jenkins: A beginner's guide to implementing continuous integration and continuous delivery using Jenkins. Birmingham: Packt Publishing.

[45] Verma, R., Shrivastava, P., & MERLA, N. (2024). Tracing the Path: Data Lineage and Its Impact on Data Governance. International Journal of Global Innovations and Solutions (IJGIS). https://doi.org/10.21428/e90189c8.4c497b37