Sentiment Analysis For Predicting Cost Management Actions Related To Changes In Infrastructure-as-Code (IaC) Artifacts

Bachelor's Thesis

To fulfill the requirements for the degree of
Bachelor of Science in Computing Science
under the supervision of:

Prof. Dr. Vasilios Andrikopoulos
&
Dr. Daniel Feitosa

Mohammad Al Shakoush (s4274865)
`m.al.shakoush.1@student.rug.nl`

3rd August 2023

Abstract

Infrastructure-as-Code (IaC) is a commonly implemented approach for the administration of IT infrastructure. It involves the use of artifacts to automate the processes of provisioning, deployment, configuration, and maintenance of infrastructure resources, all of which come with a certain cost. As the artifacts define the infrastructure, any modifications to them will have an impact on cost management. Thus, when developers make changes to the artifacts, they document the changes and the rationale behind them, typically using a version control system.

The changes and their associated description, along with the accompanying rationale, express an opinion or sentiment that could be analyzed using sentiment analysis. Sentiment analysis is a natural language processing technique that has several applications in diverse domains, but its potential for cost management in IaC remains under-explored.

The existing models for sentiment analysis have been largely unsuccessful, either being general-purpose models with poor performance or software engineering specialized models that have also been inadequate. In light of this, research has been conducted to create a custom sentiment analysis model, with the use of the Generative Pre-trained Transformer (GPT) model proving to be highly effective.

Acknowledgments

This work would not have been possible without the generous guidance and support of Dr. Daniel Feitosa and Prof. Dr. Vasilios Andrikopoulos. Furthermore, I am greatly appreciative of the assistance provided by Brian Setz. Finally, I am grateful to the University of Groningen and the Faculty of Science and Engineering for providing the necessary resources, such as Hábrók [1], Product Licenses, and others.

[1]https://www.rug.nl/society-business/centre-for-information-technology/news/verbeterd-rekencluster-habrok-geeft-onderzoeker-meer-rekenkracht?lang=en

Contents

List of Figures

1 Introduction And Motivation

Infrastructure as code (IaC) is an approach that has gained widespread acceptance for the management of IT infrastructure. It allows organizations to manage and provision their infrastructure through machine-readable definition files rather than manual configuration, automate deployments, and apply version control techniques [2, 3].

Cloud computing has revolutionized the way organizations deploy their infrastructure, allowing access to a wide range of computing services over the Internet. The main advantage of cloud computing is its cost-effectiveness since the pricing structure is linked to the level of usage. Thus, it is important to select the services that best suit the organization's needs and budget in order to maximize value and minimize expenses [4].

As infrastructure grows in complexity, managing changes to IaC artifacts becomes more challenging, and it is crucial to understand how changes impact the cost of infrastructure management in the cloud[5]. According to Gartner [6], by 2025, more than 80% organizations will adopt IaC as a fundamental practice to manage IT infrastructure. While IaC has several benefits, cost management can be a significant concern for organizations. The lack of insights into the cost implications of changes to IaC artifacts can lead to unexpected expenses [5, 7].

Sentiment analysis identifies opinions, emotions, and attitudes in text data. It has several applications in diverse domains [8]. However, the potential of sentiment analysis for cost management in IaC remains under-explored [9]. Therefore, this proposal seeks to explore the potential of sentiment analysis for predicting cost management actions in response to changes in IaC artifacts.

A worthwhile endeavor would be to investigate whether sentiment analysis can be utilized to enable organizations to effectively manage their infrastructure costs. Sentiment analysis on commit messages related to IaC artifacts can provide insights into how developers and engineers feel about changes in the infrastructure code. These changes could be related to cost optimization, performance improvement, or other factors. By analyzing the sentiment of these messages, we hope to identify patterns and trends that can help us predict cost management actions related to these changes.

For example, positive sentiment in commit messages could indicate that the changes made are likely to result in cost savings or improved performance, while negative sentiment could suggest the opposite. This information can be used to inform decision-making around cost management actions, such as whether to proceed with a particular change or explore alternative options.

The goal of this work is therefore to explore the potential of sentiment analysis on commit messages related to IaC artifacts as a tool for predicting cost management actions. By identifying patterns and trends in the sentiment of these messages, we aim to provide insights that can inform decision-making around changes to IaC artifacts, allowing organizations to optimize their infrastructure performance and reduce costs. Our ultimate objective is to provide a framework that enables organizations to make informed decisions about their infrastructure, contributing to improved business outcomes and innovation.

1.1 Research Questions

To summarize, this thesis focuses on examining the relationship between sentiment and cost management actions in IaC.

Consequently, our research questions are:

Q1. To what extent is it possible to use existing sentiment analysis models from the literature to establish this relation?

Q2. How efficient is it to create a specialized sentiment analysis model for the same purpose?

1.2 Research Outline

In Chapter 2, we provide an overview of the background information and relevant scientific literature related to the main concepts of this research.

Chapter 3 presents the study design and execution of the research conducted for this thesis regarding the non-customized models, as well as the results obtained and their discussion. As the number of results is extensive, they are included in the relevant sections within this chapter rather than a separate chapter.

Chapter 4 focuses on the process of making the customized sentiment analysis model, its challenges and results.

Finally, in Chapter 5, the conclusions and potential future works that can result from this research are outlined.

2 Background

This chapter provides an overview of the fundamental concepts central to this research. In Section 2.1 and Section 2.2 fundamental concepts are explained in order to understand Section 2.3. In which relevant background work is outlined showcasing the relevance of the sentiment in the software engineering realm and how its analysis can provide useful analysis about the project. Additionally, Section 2.4 highlights the two main components this thesis has utilized in order to achieve the results. Section 2.5, Section 2.6, Section 2.7, and Section 2.8 explain essential tools utilized during this research. Finally, Section 2.9 mentions the importance of this project's results.

2.1 Git & GitHub

In the ever-changing landscape of software development, version control, and collaborative tools are essential for effective and dependable project administration. Git, a decentralized version control system, and GitHub, a broadly utilized web-based hosting platform, have become key technologies that have revolutionized how developers manage, monitor changes, and collaborate on codebases.

2.1.1 Git

Git [1] is a distributed version control system (VCS) that has revolutionized the way in which software developers manage and track changes in their projects. Git is a widely-used VCSs within the software development community. The most relevant aspects of Git for this research are :

Commits and Snapshots : Git tracks changes to the codebase through "commits". Each commit captures a snapshot of the project at a specific point in time, making it easy to navigate through the project's history.

Version Control : Git provides a comprehensive history (the snapshots with the commits) of the codebase, allowing developers to track changes and revert to previous versions.

Consequently, having a history of snapshots of the codebase enables the developers to view every change made to every file and crucially the associated commit messages provide details of the changes.

2.1.2 GitHub

GitHub[2] is a web-based platform that facilitates software development using the Git version control system, providing an environment in which developers may collaborate on and manage code repositories. As a hub of both open-source and private development projects, it enables users to share and work together on software projects, thus promoting collaboration and innovation in the development process. Which is crucial to this research and the previous thesis outline in Section 2.4.1.

2.2 Cloud Orchestrators & IaCs

Cloud computing has revolutionized the way organizations manage and deploy their IT infrastructures. Cloud orchestrators and IaC have become essential components of modern cloud-native envir-

[1]https://git-scm.com/about
[2]https://github.com/

onments, helping to streamline cloud operations, improve scalability, and enable effective resource management. The importance of cloud orchestrators and IaC lies in their capability to automate the provisioning, configuration, and management of cloud resources, as well as to provide a platform for continuous integration and deployment. By providing a reliable, consistent, and repeatable infrastructure, these technologies have enabled organizations to quickly and efficiently respond to changing demands.

2.2.1 | Cloud Orchestrators

Cloud orchestration is a powerful tool that automates and manages the provisioning, configuration, and deployment of cloud resources and services. It serves as a critical backbone for cloud infrastructure management by abstracting the complexities of underlying cloud platforms and providing a unified interface for defining and controlling the infrastructure. Popular cloud orchestration solutions include Kubernetes [3], Docker Swarm [4], and Amazon ECS [5].

2.2.2 | Infrastructure as Code

IaC is an approach to cloud infrastructure configuration that treats it as software code. This paradigm involves representing cloud infrastructure in a declarative fashion using code, which can be version-controlled, tested, and deployed programmatically. Popular IaC tools include Terraform [6], AWS CloudFormation [7], and Azure Resource Manager [8].

2.2.3 | Versioned IaC artifacts

The combination of Git, GitHub, and IaC artifacts provides versioned IaC artifacts that describe the infrastructure in the cloud. As these files have direct influence on the cost of the cloud deployments, any changes made to them also affects the cost of the project's deployment. All of the changes are documented using Git and are accessible by others if hosted on GitHub.

2.3 | Sentiment Analysis

Sentiment analysis, also known as opinion mining, is a Natural Language Processing (NLP) technique for determining and analyzing sentiment or emotion expressed in a text, such as reviews, social media posts, or customer feedback. The purpose of sentiment analysis is to classify text into categories of positive, negative, or neutral in order to determine the overall sentiment of the author towards a particular topic. This process usually involves utilizing machine learning algorithms or lexicon-based approaches in order to detect and measure words or phrases that communicate emotion, sentiment, or polarity.

Sentiment analysis has been applied in marketing, customer experience, and social media analysis [8]. Abo et al [10] conducted a systematic mapping study dealing with sentiment analysis for Arabic texts in social media.

[3]https://kubernetes.io/
[4]https://docs.docker.com/engine/swarm/
[5]https://aws.amazon.com/ecs/
[6]https://www.terraform.io/
[7]https://aws.amazon.com/cloudformation/
[8]https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/overview

In the context of software engineering, Ahasanuzzaman et al [11] used sentiment analysis to provide API designers an insight into the issues of their API by analyzing the sentiment of questions and answers on Stack Overflow [9].

Guzman et al [12] applied sentiment analysis on commit messages from open-source projects on GitHub [10]. Then used that to analyze the sentiment relationship with different factors such as used programming language, time and day of the week in which the commit was made, team distribution, and project approval.

However, researchers have found that sentiment analysis using regular sentiment analysis models (the models used for customer reviews for example) is not accurate enough in the realm of software engineering. Since messages include technical jargons, word contractions, emoticons, URLs, and code snippets.

This prompted scientists like Biswas et al [13] to look into the effects of two possible ways to enhance the training for sentiment analysis of SE artifacts when using neural networks that were specifically designed using the Stack Overflow data. They studied the effects of software domain-specific word embeddings on the performance of the sentiment analysis tool, as compared to generic word embeddings learned from Google News, and tailored the sentiment analysis process to the software domain using software domain-specific word embeddings learned from Stack Overflow posts.

Additionally, Ahmed et al [14] made SentiCR, a Customized sentiment analysis tool for code review interactions, and compared their own made model to already existing models. The outcome of this was clear that already existing models cannot be used out of the box in software engineering.

Finally, Ding et al [15] used SentiSW, an entity-level sentiment analysis tool specific to the SE domain. Consisting of sentiment classification and entity recognition to extract the sentiment from issue comments. After building a $3,000$ labeled sentiment dataset, which is the largest software engineering dataset in sentiment analysis.

In the context of IaC, sentiment analysis has the potential to help organizations predict the cost implications of changes to IaC artifacts. By analyzing the sentiment of comments related to IaC changes, organizations can make informed decisions about the changes they make, reduce costs, and optimize infrastructure performance.[16]

In conclusion, sentiment analysis has been successfully applied in software engineering. And plays a role, in the productivity and quality of code the developers provide [17]. However, its potential for cost management in IaC stays uncharted.

2.4 Building blocks

This thesis is composed of several key components which are essential for its successful completion. These components have been instrumental in inspiring the concept of the research and enabling the successful implementation of it.

[9] https://stackoverflow.com/
[10] https://github.com/

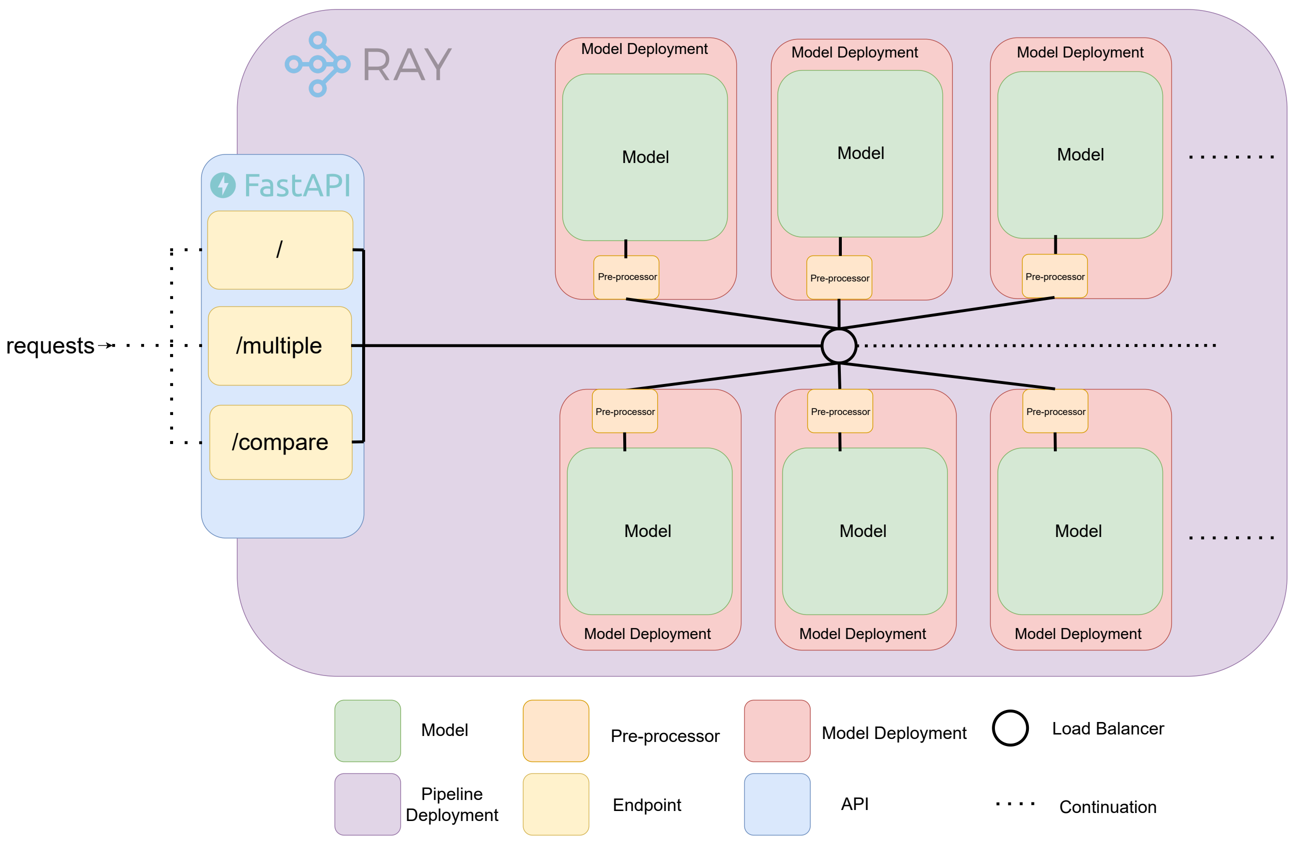


Figure 2.1: Ray cluster overview [1]

2.4.1 Previous Bachelors Project

In 2022, a thesis was created under the same supervisors and titled "Mining and Analysis of Cost-related Decisions in Cloud Infrastructures". This thesis aimed to collect relevant data pertaining to Infrastructure as Code (IaC) files. The study conducted a mining operation of public GitHub repositories to investigate the prevalence of Infrastructure as Code (IaC) artifacts. Specifically, the mining process was aimed at identifying repositories with a codebase that exceeded a specified percentage of IaC artifacts. Two datasets[11]were collected, namely commits messages and issues, both related to IaC files. As a start, the primary dataset to be explored and used in this thesis is the commits.

2.4.2 Sentiment Analysis Pipeline

In anticipation of its use in this thesis, under the same supervisors, I created a horizontally-scalable pipeline, the structure of which can be seen in Figure 2.1, to host multiple Natural Language Processing (NLP) models. This pipeline was designed such that any NLP model can be added to the pipeline after implementing the `model` interface, and the newly added model would be accessible via an Application Programming Interface (API) that exposes all the hosted models on the Ray cluster [12]. This pipeline provides a great utility in this thesis, as it allows for the easy addition of any model of interest and allows for sentiment analysis requests from that model using the exposed FastAPI[13] API. This work was done under the course name Short Programming Project (SPP).

The workflow consisted of adding all of the tested models to the pipeline and utilizing the `/multiple` endpoint to obtain sentiment analysis results from multiple models simultaneously.

[11]https://github.com/feitosa-daniel/cloud-cost-awareness/blob/main/dataset.json
[12]https://docs.ray.io/en/latest/cluster/getting-started.html
[13]https://fastapi.tiangolo.com/

Horizontally-scalable Pipeline

A horizontally-scalable pipeline is a system architecture designed to facilitate the handling of increasing workloads and data volumes by employing a distributed approach to processing tasks across multiple nodes or machines. This design strategy differs from that of traditional vertically-scalable systems, which focus on augmenting resources on a single machine, and instead, allows for the expansion of processing capabilities without any detrimental effect on performance through the addition of additional machines to the processing pool.

Micro-services Architecture

Micro-services architecture is an architectural software style that organizes an application as an aggregation of small, loosely coupled, and independently deployable services. Each service concentrates on a particular business capability and works as an independent unit with its own database, logic, and capabilities. These services communicate with each other via clearly defined Application Programming Interfaces (APIs), usually across a network.

Ray Serve

Ray Serve [14] is a library designed to facilitate the development and deployment of machine learning models as high-performance microservices. It allows developers to create production-ready serving systems with minimal latency and maximum throughput. This library provides a way for developers to create and manage machine learning models in a scalable and reliable manner.

2.5 Agreement Metrics

In this thesis, multiple models will be employed in order to label data entries, referred to as raters. Human annotations, known as ground truth, will be used to evaluate the correctness of the models. Two metrics will be utilized in order to determine the accuracy of the models, namely Cohen's kappa and Krippendorff's alpha. These metrics will allow for comparison of model output to the ground truth, in order to assess the accuracy of the models. This is also known as inter-rater reliability.

2.5.1 Cohen's kappa

The determination of interrater reliability is a critical factor in the validity of research results. The Kappa statistic is a commonly used measure to evaluate the extent to which two or more raters agree in their assessment of a particular variable. By assessing the consistency of ratings across different raters, researchers can gain confidence in the accuracy of the data collected [18].

[14]https://docs.ray.io/en/latest/serve/index.html

$$\kappa = \frac{P_o - P_e}{1 - P_e}$$
$$P_o = \frac{\sum_{i=1}^{k} n_{ii}}{N}$$
$$P_e = \frac{\sum_{i=1}^{k} (n_{i\cdot} \cdot n_{\cdot i})}{N^2}$$

where:

- κ is Cohen's kappa,
- P_o is the observed agreement, which is the proportion of observed agreement among raters,
- P_e is the agreement expected by chance, which is the proportion of agreement expected by chance,
- k is the number of categories (codes or labels),
- n_{ii} is the number of items both raters have assigned to category i,
- $n_{i\cdot}$ is the total number of items assigned to category i by all raters,
- $n_{\cdot i}$ is the total number of items assigned to category i by all raters, and
- N is the total number of ratings (the sum of all n_{ij} values).

If the raters are in complete agreement, then Cohen's Kappa coefficient, $\kappa = 1$. If there is no agreement among the raters other than that which would be expected by chance, $\kappa = 0$. It is also possible for the statistic to be negative [19], which can occur by chance if there is no correlation between the ratings of the two raters, or it may indicate a real tendency of the raters to give contrasting ratings.

2.5.2 | Krippendorff's alpha

Krippendorff's alpha is a reliability measure used to assess the agreement or reliability of data annotations or codings across multiple raters or coders. It is particularly applicable when dealing with nominal or ordinal categorical data, where raters classify items into discrete categories or assign ordinal values [20].

$$\alpha = 1 - \frac{D_o}{D_e}$$

$$D_o = \sum_{j=1}^{k} \sum_{i=1}^{n} \frac{m_{ij}(m_{ij}-1)}{N(N-1)}$$

$$D_e = \sum_{j=1}^{k} \frac{o_j(o_j-1)}{N(N-1)}$$

where:

- α is Krippendorff's alpha,
- D_o is the observed disagreement, which is the sum of squared disagreements among raters,
- D_e is the disagreement expected by chance, which is the sum of squared disagreements expected by chance,
- k is the number of categories (codes or labels),
- n is the number of items being rated, and
- N is the total number of ratings (the product of n and the number of raters).

Negative α values indicate significant disagreement or systematic bias, while an α value of 0 suggests chance-level agreement. Positive values of α reflect higher levels of agreement and indicate a more reliable and consistent coding process. An α of 1 indicates perfect agreement, which is rarely achievable in real-world data coding tasks.

2.6 Huggingface

Hugging Face[15], a well-known NLP company and open-source community, has made significant contributions to the fields of NLP and machine learning. By developing and maintaining state-of-the-art NLP libraries and models, Hugging Face makes it possible that researchers, developers, and the general public to access and use the most advanced tools in the field. The Huggingface library will be employed to deploy the models created and to access the base models necessary for this thesis.

2.7 Hábrók cluster

Hábrók cluster [16] is a computer cluster provided by the University Of Groningen. This cluster is a collection of computers that can be used for doing calculations that exceed the capacity of an average desktop or laptop. This cluster will be used when training the models and generating them. All of our experiments were run on the cluster with at least two $A100$ NVIDIA GPUs and at least $6GB$ of memory per GPU.

2.8 Alpaca-LoRA

Alpaca-LoRA [17] is a low-power variant of the Stanford Alpaca framework designed to enable the training of large models on low-end devices such as the Raspberry Pi. This variant leverages Low-Rank Adaptation (LoRA) to reduce memory consumption and accelerate training time [21].

LoRA freezes the pre-trained model weights and injects trainable rank decomposition matrices into each layer of the Transformer architecture, greatly reducing the number of trainable parameters for

[15]https://huggingface.co/
[16]https://wiki.hpc.rug.nl/habrok/introduction/
[17]https://github.com/tloen/alpaca-lora

downstream tasks. This has several key advantages, sharing and building the model becomes increasingly easier, more efficient and lowers the hardware barrier, and merge the trainable matrices with the frozen weights [21].

2.9 Paper Write-up

This research is in the process of being compiled into a comprehensive research paper by esteemed supervisors. The paper is undergoing extensive review and evaluation.

The review process is ongoing and entails a critical evaluation of the research methodology, data analysis, and interpretation of results, with a steadfast dedication to upholding scholarly standards and scientific rigor. The paper aims to communicate the insights gained, the methods used, and the implications of the findings in a clear, concise, and academically sound manner.

3 Study design and execution

This section delves into the design choices that have been systematically employed throughout the course of this research endeavor. Additionally, it elaborates on the pragmatic aspects of their realization and implementation.

Developers use version control systems such as Git to manage their Infrastructure as Code (IaC) artifacts, which are often hosted on GitHub and are accompanied by commit messages that describe the changes made to them. This allows for an understanding of the reasoning behind the changes to the IaC, and thus can provide insight into the associated costs of hosting IT infrastructure in the cloud. This has driven research efforts to extract these changes and commit messages, leading to datasets such as those discussed in Section 2.4.1.

In order to be able to answer the research question we need to run sentiment analysis on the datasets mentioned. Extracting the sentiment of every entry in the dataset could be achieved using different methods.

In this section, every experiment done in this thesis is outlined in addition to the associated results. In Section 3.2 the utilization of the General NLP Models is described, and in Section 3.3 the attempt to use the Software Engineering NLP Models is brought into the spotlight along with the issues encountered.

This study will only analyze the commit messages dataset to gain insight. Additionally, the issues dataset will be processed through the models, as the only difference between the two datasets is the text sent to the API. The reason for this is having the issues dataset labeled could be beneficial for further research, but due to time constraints analyzing the results will not take place.

3.1 Overall Design

The primary goal of this study is to address the research questions (RQ) defined in the introduction and evaluate the effectiveness of sentiment analysis in software engineering using the Alpaca model. To achieve this, we have designed a comprehensive methodology that leverages the existing components and knowledge discussed in the previous chapter.

Our approach involves the following key steps:

Tripolar Tagging of Sentiments: In contrast to the traditional binary or polarity-based sentiment analysis, we adopt a tripolar tagging scheme to capture nuances in sentiment expression. Our sentiment labels are categorized into three classes: positive, neutral, and negative. This approach allows us to provide more nuanced and context-aware sentiment predictions.

Model Selection: For the sentiment analysis task, we carefully choose a set of language models from the literature, encompassing both transformer-based and traditional machine learning approaches. We will start with general-use models, move on to more software engineering models, and finally make a customized model as it is necessary.

Establishing the ground truth: To ensure the accuracy and reliability of the tripolar tagging, we perform a rigorous process of establishing the ground truth. The author and GPT-3.5 annotate independently the commit messages, and inter-annotator agreement measures, such as Cohen's kappa, are computed to assess the consistency of annotations.

Evaluation Metrics: To assess the models' effectiveness, we employ Cohen's kappa and Krippendorff's alpha as evaluation metrics. These metrics provide measures of inter-rater agreement, allowing us to assess the consistency and reliability of sentiment predictions.

Comparison with Baselines: We establish baseline models using conventional sentiment analysis techniques and compare their performance with the fine-tuned Alpaca models. This comparison enables us to gauge the improvement achieved using Alpaca's capabilities.

Tripolar Tagging of Sentiments

The tripolar tagging of sentiments serves as a fundamental design decision in this study. By moving beyond the binary sentiment classification, we aim to capture nuanced sentiments expressed in commit messages. For instance, a developer's sentiment towards a code change could be positive due to efficiency improvements, neutral for minor bug fixes, or negative for complex refactoring. The tripolar tagging approach enhances the granularity of sentiment analysis, making it more contextually relevant to software engineering tasks.

Model Choices

In selecting the language models for sentiment analysis, we strike a balance between the state-of-the-art transformer-based models, known for their contextual understanding, and traditional machine learning models, renowned for their interpretability.

Experimental Setup

We conduct extensive experiments on the prepared dataset using the Alpaca models. Each model undergoes fine-tuning. The evaluation metrics provide quantitative measures of each model's ability to accurately predict sentiments.

3.2 General-use Sentiment Analysis Models

The first method to label all the entries in the dataset is to use pre-existing models used to achieve this specific goal. Therefore, three general-use models are selected for use: **Stanza NLP**[1], **Vader**[2], and **Textblob**[3]. These models are commonly employed in sentiment analysis and are thus chosen as the go-to models for the purpose of this study.

By using the pipeline outlined in Section 2.4.2, all three models were easily implemented (as illustrated in Figure 3.1). Subsequently, the exposed endpoints yielded the results mentioned in Section 3.2.1.

[1]https://stanfordnlp.github.io/stanza/
[2]https://github.com/cjhutto/vaderSentiment
[3]https://textblob.readthedocs.io/en/dev/

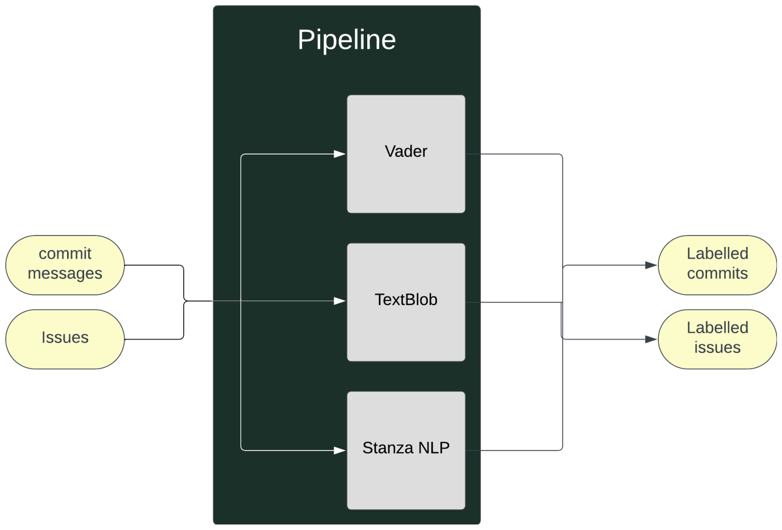


Figure 3.1: General-use sentiment analysis models in the pipeline

Naturally, every model requires a specific implementation to extract the sentiment of a given piece of text. However, implementing the `Classifier` interface guides us toward what methods to implement. Therefore, **TextBlob**'s implementation in the pipeline is as follows:

```python
@serve.deployment
class TextBlobClassifier(Classifier):
    def __init__(self):
        self.model = TextBlob
    '''
    TextBlob takes as input one single sentence at a time
        - we classify the sentence by calling the TextBlob class
          with the given sentence
        - If the polarity is 0 then neutral, > 0 positive else negative
    '''
    def classify(self, text : str):
        polarity = self.model(text).sentiment.polarity
        if polarity > 0:
            response = "Positive"
        elif polarity == 0:
            response = "Neutral"
        else:
            response = "Negative"

        return response
```

The **Stanza NLP** model implementation was not as straightforward due to serialization issues [4]. As

[4] https://stackoverflow.com/questions/68787955/cant-pickle-thread-rlock-objects-when-using-huggingface-trainer-with-ray-tun

each model had to be serialized to be inserted into the pipeline, Ray employed the use of the `pickle` library [5] Unfortunately, it was unable to serialize the Stanza NLP model, so an alternate library, `dill` [6] was used instead.

Consequently, a manual serialization and deserialization of the model prior to it being added to the pipeline needed to take place. The serialization had already been done once by another process, therefore only the deserialization needed to be done before the model could be added, which was done using a `startup_event` method. The implementation is as follows:

```
STANZA_NAME = "stanza_model.pkl"
def startup_event():
    if os.path.exists(STANZA_NAME) == False:
        stz.download('en')
        # Load the Stanza model
        nlp = stz.Pipeline('en')

        # Serialize the Stanza model using dill
        with open(STANZA_NAME, 'wb') as f:
            dill.dump(nlp, f)
@serve.deployment
class StanzaClassifier(Classifier):
    def __init__(self):
        # run the startup event
        startup_event()
        # if the file does not exist throw an error
        if os.path.exists(STANZA_NAME) == False:
            raise Exception("Stanza model not found")

        file = open(STANZA_NAME, 'rb')
        nlp = dill.load(file)
        self.model = nlp

    def classify(self, text : str):
        doc = self.model(text)
        sentiment = 0
        size = len(doc.sentences)
        for i, sentence in enumerate(doc.sentences):
            sentiment += sentence.sentiment
        if (size == 0):
            return "Neutral"
        sentiment = sentiment / size
```

[5]https://docs.ray.io/en/latest/ray-core/objects/serialization.html#serialization-notes
[6]https://pypi.org/project/dill/

```
        polarity = round(sentiment, 0)
        response = {}
        if polarity == 0:
            response = "Negative"
        elif polarity == 1:
            response = "Neutral"
        else:
            response = "Positive"
        return response
```

The **Vader** model addition to the pipeline was as straightforward as Textblob without any issues. The implementation in the pipeline:

```
@serve.deployment
class VaderClassifier(Classifier):
    def __init__(self):
        self.model = SentimentIntensityAnalyzer()
    '''
    Vader takes as input one single sentence at a time
        - we classifiy the sentence by calling the Vader class
          with the given sentence
    '''
    def classify(self, text : str):
        polarity = self.model.polarity_scores(text)


        if polarity['compound'] >= 0.05 :
            response = "Positive"
        elif polarity['compound'] <= - 0.05 :
            response = "Negative"
        else :
            response = "Neutral"

        return response
```

3.2.1 Results

For the results, the JSON markup language was used due to the pipeline using it as its API's response. As mentioned before the `/multiple` endpoint was used with all three mentioned models as the parameters. This results in the following response for a single commit message:

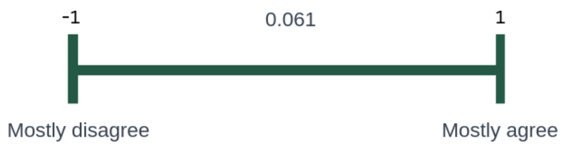
```
{
    "type": "commit",
    "url": "https://github.com/blinkist/terraform-aws-airship-ecs-
    cluster/commit/d7aa659971bee1be873d3dda92e30443556f52df",
    "content": {
        "message": "Removed the default use of detailed monitoring. (#17)
        * Reduces CloudWatch costs for metrics by 80%"
    },
    "codes": [
        "saving"
    ],
    "sentiment": {
        "text_blob": "Positive",
        "vader": "Neutral",
        "stanza": "Negative",
    }
},
```

The pipeline was used to analyze each commit in the dataset. It was able to accept an array of commits, making it simple to apply to the entire dataset, as shown in Figure 3.1.

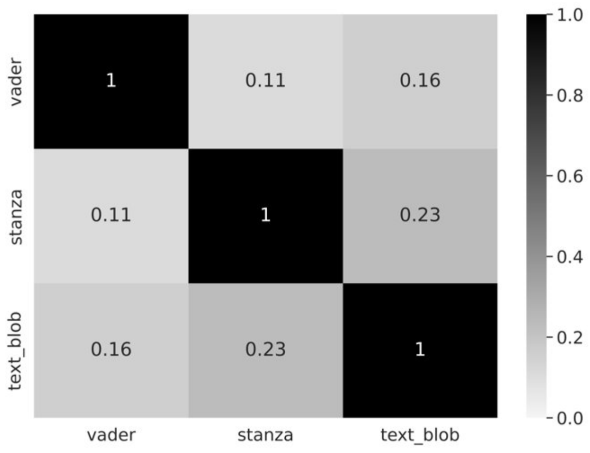
The degree of agreement between the three models applied to the labeled dataset was found to be low, resulting in a 57% percentage of conflicts between the models. To evaluate this agreement, Krippendorff's alpha (see Figure 3.2a) and Cohen's kappa (see Figure 3.2b) were utilized as measures.

Krippendorff's alpha score of 0.061 is indicative of low agreement among raters or coders. This score is considered relatively low, indicating that the observed agreement in the data is not significantly different from what would be expected by chance alone. As a reliability coefficient, higher values are generally desired to indicate better agreement between the models.

Cohen's Kappa score of $0.11 - 0.23$ is regarded as relatively low, indicating a lack of agreement between the models. Again suggesting that the observed agreement in the data is not significantly different from what would be expected by chance alone.



(a) Krippendorff's alpha



(b) Cohen's kappa

Figure 3.2: General-use Models Results

Furthermore, the models were found to be unable to handle typos correctly, resulting in disagreements between the models. For example:

```
{
    "type": "commit",
    "url": "https://github.com/stealthHat/k8s-
    terraform/commit/681a3f8b4942be495b3f2528fb9ee40d7a4eb08a",
    "content": {
        "message": "nat gateway is verry expensive"
    },
    "sentiment": {
        "text_blob": "negative",
        "vader": "neutral",
        "stanza": "positive"
    }
}
```

We can see that all 3 models are giving different sentiments for the same sentence.

Lastly, the models were unable to understand software jargon in the commit messages, which hindered their ability to accurately analyze the data.

```
{
    "type": "commit",
    "url": "https://github.com/thomastodon/jabujabu/commit/
    02210a3d3ba4a770c29623825b7f54f3ff33f3c7",
    "content": {
        "message": "Make the concourse cluster cheaper
        - no longer uses a load balancer\n- no longer uses more expensive VMs"
    },
    "sentiment": {
        "text_blob" : "neutral",
        "vader": "negative",
        "stanza": "negative",
    }
}
```

3.2.2 Conclusion

From the observations, we can safely conclude that we cannot use these models' results as the ground truth for our analysis. From the results discussed in Section 3.2.1 that are also depicted in Figure 3.2 we know that such results and such scores have the following implications:

- Limited Reliability: Such kappa and Krippendorff scores suggest only a slight agreement between the raters, indicating inconsistent assessments of the data.
- Subjectivity and Variability: The observed scores may be indicative of subjectivity and variability in the coding process. It suggests that the models hold different interpretations of the categories or criteria, resulting in in-congruent outcomes.
- Challenges in Data Interpretation: Interpreting the data becomes more arduous with such scores. It is difficult to draw dependable conclusions or generalize findings due to the limited agreement between raters.
- Necessity for Improvement: The diminished scores underscore the necessity for procedural enhancement in the coding process. Additional training or clearer guidelines for models are essential to augment their agreement and heighten the data's overall reliability.

It is not possible to use these models to accurately determine the true sentiment (i.e. ground truth) of commit messages. Consequently, alternative models must be explored in order to develop a reliable method for gauging sentiment. Although further training may help to improve the agreement between models, this would require significant time and effort. Therefore, it is necessary to identify models that have already been trained in a context similar to that of the current research.

3.3 Software Engineering Sentiment Analysis Models

In Section 3.2.2 it was established that sentiment analysis models that are aware of software-specific jargons are in need. Accordingly, Senti-SE [7] and Senti-CR[8], models that were initially discussed in Section 2.3, were selected for use in the context of software engineering. Both of these models were trained with the consideration of software-jargons, and were successfully applied in extracting the sentiment of sentences.

3.3.1 Results

The results of both models were not satisfactory. When analyzing Senti-CR, it was observed that it did not perform adequately. Similarly, Senti-SE yielded poor results. Both results will be discussed below.

Senti-SE

When attempting to utilize this model, we encountered a few issues that set us back and made it not possible for us to run this model locally. The model was built using JAVA with Ant as the building tool.

The repository includes a fat jar [9] of the model which should be executable; however, this jar does not function due to missing classes. It was discovered that the structure of the code for the model needed to be fixed in order to build it locally, rather than relying on the included jar. Upon attempting to build the model locally, it became apparent that some crucial files were missing which were required for training the model. Consequently, the author was contacted to provide the missing files or further instructions on how to build the model; however, no files or instructions were received.

Senti-CR

Senti-CR, built using Python, proved to be a difficult task to work with. The repository lacked clear indications of the version of Python used, as well as the dependencies and their exact versions. This made getting started with the model very difficult. After using the latest version of Python and fixing all syntax errors, the necessary libraries were updated. The fixes included:

- The `workbook` library for python is not supported for the current LTS version `3.11` and needed to be swapped for `openpyxl`. Consequently, all snippets of code invoking specific methods of `workbook` needed to be swapped with those of `openpyxl`.
- The `SMOTE` does not accept as many parameters as before. Therefore the function calls needed to be updated.
- The method called `smote_model.fit_sample` has been renamed to `smote_model.fit_resample`.
- The `3.11` Python does not allow many old syntax but the most interesting one is

  ```
          for i, j in dic.iteritems():
  ```

  Which needed to be changed to

[7]https://github.com/amiangshu/SentiSE
[8]https://github.com/senticr/SentiCR
[9]https://stackoverflow.com/questions/19150811/what-is-a-fat-jar

```
        for i, j in dic.items():
```

- The `string` in `expand_contractions` is now a byte string, so a change to `string.decode("utf-8")` was necessary.

Subsequently, the model was trained, taking several hours. After which the model was added to the pipeline as depicted in Figure 3.3 to be tested with the datasets.

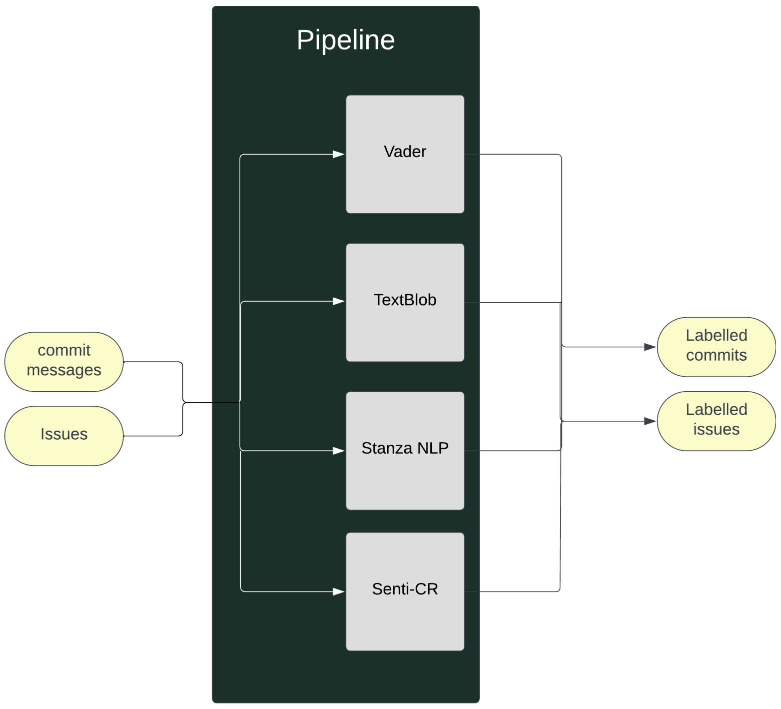


Figure 3.3: Senti-CR in the pipeline

The specific implementation as added in the pipeline is as follows :

```
SENTI_CR_NAME = "classifier_model_GBT.pkl"

'''
called before the API is started, check if the model exists
'''
def startup_event():
    if os.path.exists(SENTI_CR_NAME) == False:
        Exception("SentI_CR model not found")


@serve.deployment
class SentI_CRClassifier(Classifier):
    def __init__(self):
        # run the startup event
        startup_event()
        file = open(SENTI_CR_NAME, 'rb')
        nlp = dill.load(file)
        self.model = nlp

    def classify(self, text : str):
        doc = self.model(text)
        sentiment = 0
        size = len(doc.sentences)
        # Same concept as in the Issues notebook. We average out the sentiment over all
        for i, sentence in enumerate(doc.sentences):
            sentiment += sentence.sentiment

        if (size == 0):
            return "Neutral"

        sentiment = sentiment / size
        polarity = round(sentiment, 0)
        response = {}
        if polarity == 0:
            response = "Negative"
        elif polarity == 1:
            response = "Neutral"
        else:
            response = "Positive"
        return response
```

Testing with the datasets showed that the model utilized polarities that did not align with the -1 for negative, 0 for neutral, and 1 for positive which was mentioned in the paper. Instead, it used only 0 or 1. For the purpose of this research, we do need all three polarities.

3.3.2 Conclusion

The results presented in Section 3.3.1 indicate that Senti-SE cannot be used for this research, as the necessary files were not provided by the author and could not be obtained locally. Which lead to not having a functioning version of the model.

Additionally, Senti-CR is incompatible with this research due to its use of different polarities output. This is regrettable, as comparison with these models would have been beneficial to this research.

3.4 Conclusion

In Section 3.2.1 and Section 3.3.1, our results demonstrated that the general-use models and the attempted software engineering models were inadequate for labeling our dataset of commit messages.

In Section 3.2.2 we concluded that general-use models demonstrated difficulty in correctly labeling the dataset due to typos and software engineering jargon, resulting in a low agreement between the models. This suggests that a better solution is required. Which prompted us to use software engineering models as described in Section 3.3.

In Section 3.3.2 we concluded that attempted software engineering models are unusable for either an inadequate project setup or for a conflict between the model result and our tri-polarity.

This reveals the need for a model that encompasses all of our requirements. Learning from the errors encountered when attempting to use the previous models, we can develop a customized sentiment analysis model as outlined in Chapter 4.

4 Custom Sentiment Analysis Model

After examining the two options of utilizing general-use Sentiment Analysis models, as outlined in Section 3.2, and software engineering Sentiment Analysis models, as outlined in Section 3.3, we now move towards applying our customized Sentiment Analysis model that fulfills the requirements of:

- **Handling typos correctly**: in Section 3.2.1 it was demonstrated that Sentiment Analysis models must be specifically trained in order to effectively handle typos in sentences.
- **Being aware of software-specific jargons**: as outlined in Section 3.2.1 Sentiment Analysis models that are not aware of the context are not able to provide correct sentiment.
- **High accuracy**: our model must achieve high accuracy in predicting sentiment, which can be determined by comparing its results to the manually labeled data. The manual label will be used as the ground truth.

Therefore, the following next steps could be identified:

1. Labeling the data.
2. A suitable base model must be chosen to meet most of the requirements.
3. Train the model with the labeled dataset.
4. Study the results.

4.1 Manual Data labeling

Recent developments in the field of artificial intelligence have led to the emergence of large-scale language models, which are constructed using artificial neural networks. These models are pre-trained using self-supervised and semi-supervised learning approaches and typically contain tens of millions to billions of weights. In order to efficiently train these models, specialized AI accelerator hardware is used to parallel process large volumes of textual data, typically obtained from the Internet. As language models, they are able to take an input text and generate a prediction of the subsequent token or word [22].

Data labeling in the context of this research is to label all the commit messages in the commits dataset as one of the three polarities positive, neutral, and negative. Additionally, we decided to take an extra step and include the rationale behind the sentiment when labeling. The annotators of this data will be the author and **GPT-3.5 3.5** specifically `text-davinci 003`. The reason we chose to include another agent is that the use of two agents to label data, instead of relying on a single agent, offers several advantages that contribute to the overall quality and reliability of the dataset. These advantages include :

- **Increased inter-rater agreement and reliability**: it allows for the assessment of inter-rater agreement. Comparing the labels generated by the two agents using metrics as done in Section 3.2.1.
- **Improved error identification and correction**: the use of two agents to independently label data can help to identify and rectify any errors or inconsistencies within the labeling process.

Disagreements between the agents can be used to indicate areas of ambiguity or complexity, thus necessitating further clarification.

- **Reduced bias**: the use of multiple agents in the labeling process can help to counterbalance any biases that may be present due to the individual nature of the task. By having more than one agent contribute to the labeling process, a more balanced and comprehensive view of the data can be generated, thereby reducing the risk of bias and increasing the accuracy of the results.
- **Validation and Confidence**: When presenting research or using the labeled data for machine learning purposes, having two agents label the data adds an additional layer of validation and confidence in the results.

4.2 LLM & GPT

Generative Pre-trained Transformer 3 (GPT-3), released by OpenAI in 2020, is a large language model (LLM) based on the Transformer architecture, which replaces recurrent and convolutional architectures with an attention-based decoder. Compared to its predecessor GPT-2, GPT-3 has a significantly larger model size and a number of parameters, enabling it to generate more accurate language models [23]. Recent research has demonstrated the efficacy of GPT-3.5, leading to its inclusion as a part of this thesis [24].

4.3 GPT-3.5

The reasons to specifically choose GPT-3.5 `text-davinci 003` include :

- **Contextual understanding**: : GPT-3.5 stands as a powerful language model renowned for its contextual comprehension of natural language. Given the conciseness and potential presence of domain-specific jargon and informal language in commit messages, GPT-3.5's contextual understanding allows for effective sentiment extraction, even in the absence of explicit indicators, thereby enhancing the accuracy of sentiment analysis.
- **Handling varied sentence structure**: The heterogeneity in length and structure of commit messages demands a versatile analysis approach. GPT-3.5 aptly accommodates the spectrum of message lengths and diverse sentence patterns, rendering it well-suited for analyzing a wide range of commit messages and extracting sentiment with precision.
- **Large context window**: The architectural design of GPT-3.5 affords it the capability to consider an extensive context window, encompassing numerous preceding tokens within a text. This feature fosters a deeper understanding of the commit message, potentially capturing sentiments that transcend individual sentences, thus enriching the sentiment analysis process.
- **Developer-friendly API**: GPT-3.5's API caters to developers with a user-friendly interface, simplifying integration into applications. This facilitates seamless implementation and empowers developers to harness the model's capabilities effectively.

GPT-3.5 was then added to the pipeline as depicted in Figure 4.1. The specific implementation in the pipeline is as follows:

```
class GptClassifier(Classifier):
    def __init__(self):
        self.model = "text-davinci-003"
    def classify(self, text : str):

        response = openai.Completion.create(
            model = f"{self.model}",
            prompt= f"Use JSON to format the response like this:\n\n
            {{
                \"sentiment\": \"sentiment here\",
                reason: \"reason here\"
            }}.
            Classify the sentiment of the following sentence
            and give me the reason:\n\n{text}\n",
            temperature=0,
            max_tokens=60,
            top_p=1.0,
            frequency_penalty=0.0,
            presence_penalty=0.0
        )

        return response['choices'][0]['text']
```

In order to understand the rationale behind the output of a GPT-3.5, it is essential to include the reasoning behind the sentence. This is especially necessary when disputes arise so that we can comprehend GPT-3.5's thought process.

The temperature of GPT-3.5 was reduced to zero, thereby mitigating the risk of obtaining unexpected or creative responses from the model, which has been known to possess a hallucinatory element.

4.3.1 Results

In terms of consistency in formatting the response, the results of GPT-3.5 were found to be inadequate. Furthermore, the generated response did not adhere to the prompt, which required the use of JSON notation. Consequently, additional processing is required to clean the data from any special characters and extraneous text generated by GPT-3.5.

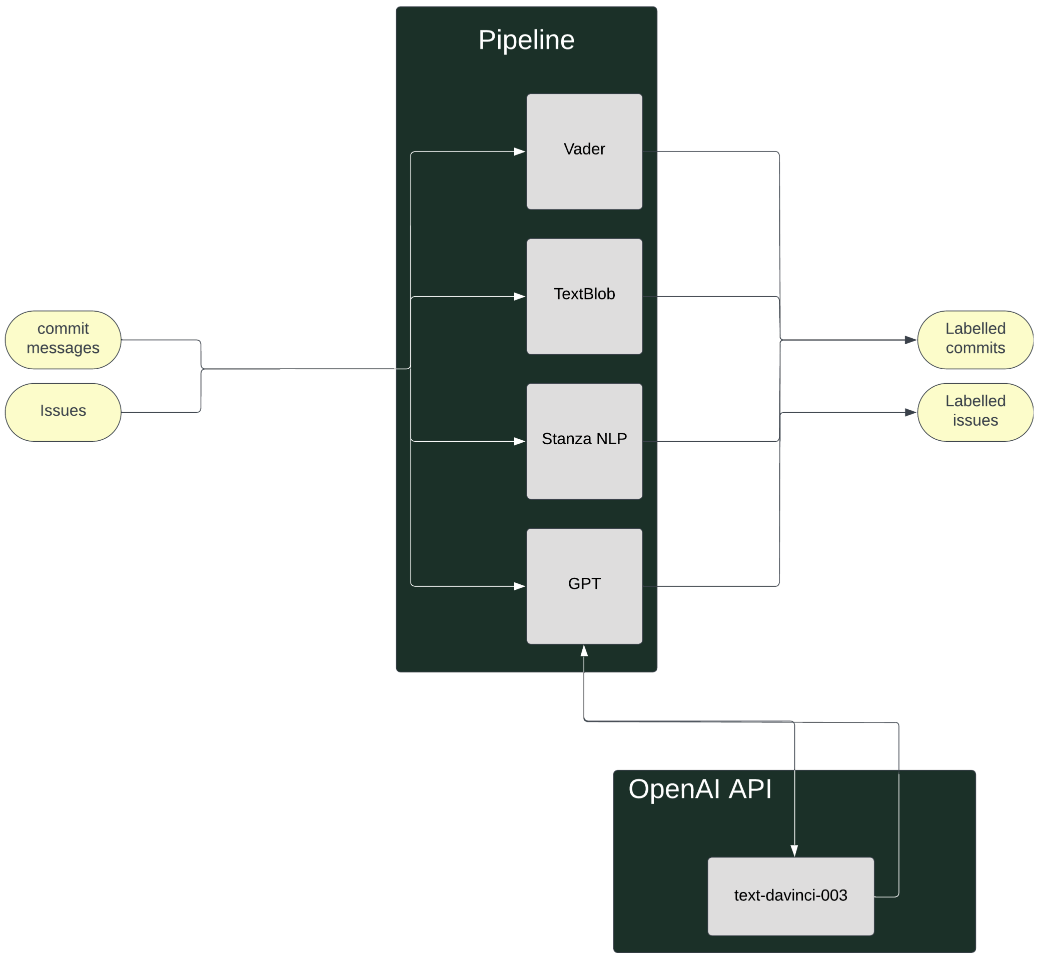


Figure 4.1: GPT-3.5 added to the pipeline

Some of the things noticed from the responses of GPT-3.5 are:

Unsolicited text completion, 2 examples:

```
{
    "type": "commit",
    "url": "https://github.com/JamesWoolfenden/terraform-aws-
    codebuild-container/commit/4a00ffcbf9576d7e5febdbdf94a31d4735fc8035",
    "content": {
        "message": "costs"
    },
    "codes": [
        "awareness",
        "storage"
    ],
    "sentiment": {
        "text_blob": "Neutral",
        "vader": "Neutral",
        "stanza": "Neutral",
        "gpt": " are too high
        Sentiment: Negative
        Reason: The sentence expresses dissatisfaction
        with the high cost of something."
    }
},
{
    "type": "commit",
    "url": "https://github.com/ken-matsui/poac-
    infrastructure/commit/02c710b8259f493c475021fc9eac23b871305ae6",
    "content": {
        "message": "cost reduction\nhttps://aws.amazon.com/about-aws/whats-
        new/2017/06/amazon-rds-enables-encryption-at-rest-for-additional-t2
        -instance-types"
    },
    "codes": [
        "saving",
        "provider"
    ],
    "sentiment": {
        "text_blob": "Neutral",
        "vader": "Neutral",
        "stanza": "Neutral",
        "gpt": "-and-cost-reduction/\n\nSentiment: Positive\nReason:
        The sentence is discussing the addition of encryption at rest
        for additional T2 instance types and cost reduction, which are
        both positive developments."
    }
},
```

Examining the GPT-3.5 result, it is evident that GPT-3.5 completed the sentence due to the lack of contextual information for ascertaining sentiment.

Inconsistent result notation, 2 examples

```
{
    "type": "commit",
    "url": "https://github.com/tooxie/terraform-
    workshop/commit/002bcce28e46728714fa1e0d20bec6f2559caba2",
    "content": {
        "message": "Add prod var\n\nWhat if we want to use less
        (or cheaper) infrastructure for non-prod systems?"
    },
    "codes": [
        "instance",
        "awareness"
    ],
    "sentiment": {
        "text_blob": "Negative",
        "vader": "Positive",
        "stanza": "Neutral",
        "gpt": "\n\nNeutral. This sentence does not express any sentiment
            , it is simply asking a question."
    }
},
{
    "type": "commit",
    "url": "https://github.com/chad-russell-git/terraform-oci-cis-landing-
    zone/commit/7247909ecd98c2d511316392f22cb3877f05250b",
    "content": {
        "message": "added policies for cost management"
    },
    "codes": [
        "awareness",
        "policy"
    ],
    "sentiment": {
        "text_blob": "Neutral",
        "vader": "Neutral",
        "stanza": "Neutral",
        "gpt": "\n\nNeutral. This sentence does not express any sentiment, so it is cla
    }
},
```

The GPT-3.5 results are then cleaned and validated to look like the following:

```
{
    "type": "commit",
    "url": "https://github.com/tkhoa2711/terraform-
    digitalocean/commit/a86d89369aaf5a20c1e4d8415a8a771aa7de7d10",
    "content": {
        "message": "provision a droplet with cheapest price"
    },
    "codes": [
        "saving"
    ],
    "sentiment_analysis": [
        {
            "classifier": "text_blob",
            "sentiment": "neutral",
            "reason": "none"
        },
        {
            "classifier": "vader",
            "sentiment": "neutral",
            "reason": "none"
        },
        {
            "classifier": "stanza",
            "sentiment": "neutral",
            "reason": "none"
        },
        {
            "classifier": "gpt",
            "sentiment": "neutral",
            "reason": "The sentence does not express any emotion or opinion."
        }
    ]
},
```

Despite some erratic results, GPT-3.5 results appear to be promising. This begs the question of whether GPT-3.5 is able to handle typos and software jargon, which were identified as bottlenecks for general-use models, as demonstrated in Section 3.2.1.

Commit message with a typo:

```
{
    "type": "commit",
    "url": "https://github.com/stealthHat/k8s-
    terraform/commit/681a3f8b4942be495b3f2528fb9ee40d7a4eb08a",
    "content": {
        "message": "nat gateway is verry expensive"
    },
    "codes": [
        "networking",
        "awareness"
    ],
    "sentiment_analysis": [
        {
            "classifier": "text_blob",
            "sentiment": "negative",
            "reason": "none"
        },
        {
            "classifier": "vader",
            "sentiment": "neutral",
            "reason": "none"
        },
        {
            "classifier": "stanza",
            "sentiment": "positive",
            "reason": "none"
        },
        {
            "classifier": "gpt",
            "sentiment": "negative",
            "reason": "The word 'expensive' implies a negative sentiment."
        }
    ]
}
```

Commit message with a software-jargon:

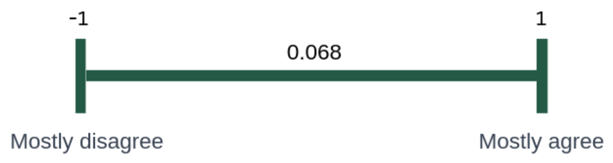
```
{
    "type": "commit",
    "url": "https://github.com/midl-dev/tezos-auxiliary-
    cluster/commit/9cbfebaab11cb3466b160d18ef2eb46c0b875d55",
    "content": {
        "message": "cheaper vms"
    },
    "codes": [
        "saving",
        "instance"
    ],
    "sentiment_analysis": [
        {
            "classifier": "text_blob",
            "sentiment": "neutral",
            "reason": "none"
        },
        {
            "classifier": "vader",
            "sentiment": "neutral",
            "reason": "none"
        },
        {
            "classifier": "stanza",
            "sentiment": "neutral",
            "reason": "none"
        },
        {
            "classifier": "gpt",
            "sentiment": "positive",
            "reason": "The sentence is expressing a desire for cheaper
            virtual machines, which is a positive sentiment."
        }
    ]
},
```

GPT-3.5 is shown to possess robustness towards typos, exhibiting a lesser degree of confusion compared to the other three models. Additionally, GPT-3.5 has demonstrated an understanding that the acronym ’VM’ stands for ’Virtual Machine’, as well as the sentiment that obtaining cheaper Virtual Machines is a desirable outcome.

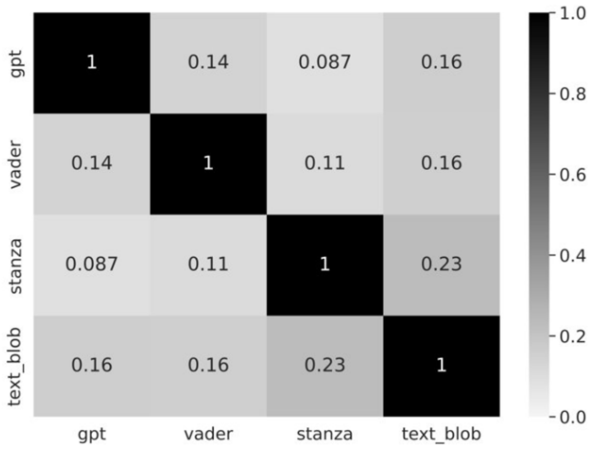
4.3.2 GPT-3.5 & General-Use Models

Once the results of GPT-3.5 had been cleaned and incorporated into the dataset, we ran the same metrics as in Section 3.2.1 (namely Krippendorf’s alpha and Cohen’s kappa) to measure the agreement of GPT-3.5 with the general-use models (Figures Figure 4.2a and Figure 4.2b, respectively).

We can clearly see that the scores of Cohen’s kappa have a min of 0.087 and a max of 0.16, indicating a very low agreement between GPT-3.5 and the general-use models. Additionally, Krippendorph’s alpha of 0.068 further confirms the low level of agreement between the aforementioned models and the fact that these models’ agreements or disagreements are the same as you would expect from chance alone.



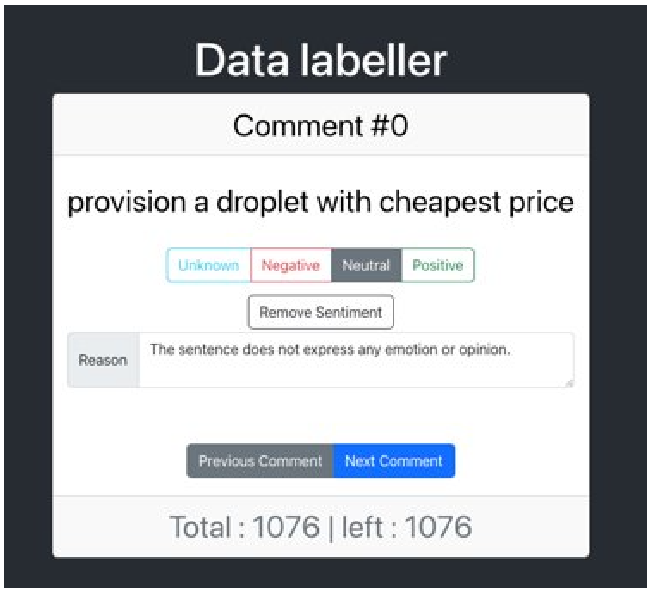
(a) Krippendorff's alpha



	gpt	vader	stanza	text_blob
gpt	1	0.14	0.087	0.16
vader	0.14	1	0.11	0.16
stanza	0.087	0.11	1	0.23
text_blob	0.16	0.16	0.23	1

(b) Cohen's kappa

Figure 4.2: GPT-3.5 agreement with general-use models



provision a droplet with cheapest price

The sentence does not express any emotion or opinion.

Total : 1076 | left : 1076

Figure 4.3: Data labeler frontend

4.4 The Ground Truth

To establish the ground truth manual data labeling was carried out. To facilitate this, a web application was created, as illustrated in Figure 4.3. This provides a user-friendly frontend instead of working within the JSON file.

The application consists of both a backend and a frontend. The backend reads the dataset and sends the commit messages to the frontend one by one. The user then selects the sentiment for each message before the frontend sends it to the backend, which writes it to the respective file. This process is visualized in Figure 4.4. In order to expedite the development process, the technology stack for this application consists of React as the frontend framework and FastAPI for the backend. React was chosen for its ability to rapidly-produce a working application, while FastAPI was selected for its speed and performance.

One of the thesis supervisors was made responsible for resolving conflicts between the manual labeling and the one produced by GPT-3.5

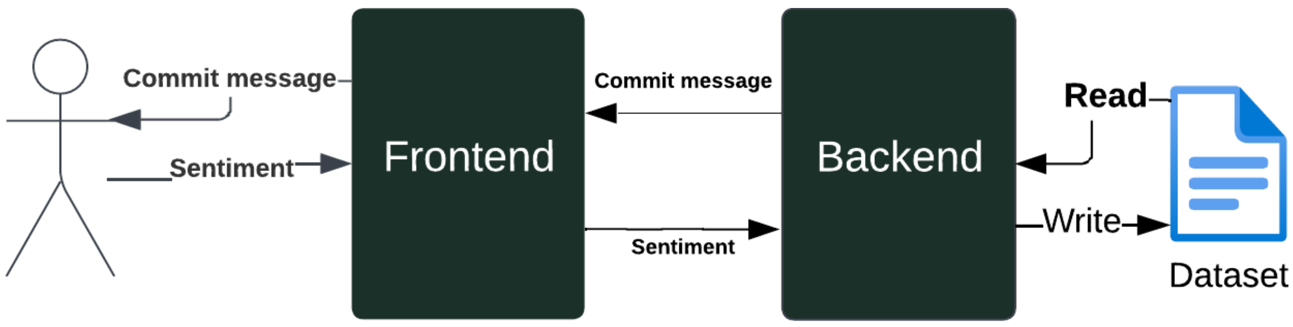


Figure 4.4: Data labeler overview

4.4.1 GPT-3.5 & Ground Truth

Upon successfully annotating all commit messages within the dataset, the subsequent step involves gaining valuable insights into the concordance between my annotations and those generated by the GPT-3.5 model. Furthermore, this analysis encompasses a comparative examination of the agreement not only between my annotations (considered the ground truth) and GPT-3.5 but also between my annotations and other existing models utilized in the context of this research endeavor. As per usual, this process seeks to evaluate the extent of alignment and divergence in the labeling of commit messages across the author, GPT-3.5, Vader, Stanza NLP, and TextBlob. Thereby providing a comprehensive understanding of the performance and efficacy of the GPT-3.5 model and other models in comparison to the established ground truth.

Such an investigation is crucial in ascertaining the reliability and consistency of the annotations and the potential impact on the overall reliability of the dataset.

In a manner akin to Section 3.2.1, a comparative analysis of inter-annotator agreement will be conducted, specifically employing Krippendorff's Alpha and Cohen's Kappa measures. The assessments will be performed between alshakoush (the author) and the GPT-3.5 model. The resulting outcomes are visualized in Figure 4.5a and Figure 4.5b, respectively.

4.4.2 General-Use Models & Ground Truth

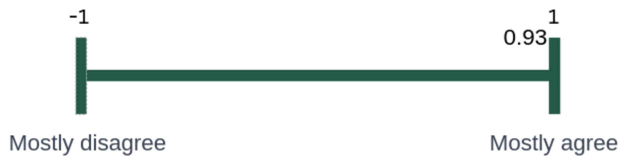
Evaluating the performance of the general-use models can be easily done now by comparing their extracted sentiment to the ground truth. To do this, Cohen's Kappa and Krippendorff's Alpha metrics could again be used to measure the agreement between the models and the ground truth. This will enable us to determine how accurate the models were, and whether our conclusions from Section 3.2.2 were accurate.

4.5 Data labelling conclusions

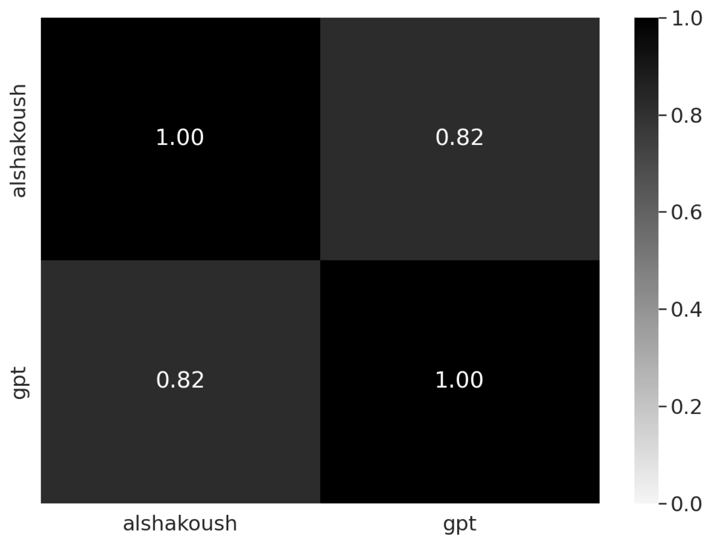
Based on the analysis conducted in the previous sections, a few conclusions can be drawn.

4.5.1 GPT-3.5 & General-Use Models

The results of the comparison between the GPT-3.5 model and the other models revealed that there was essentially no agreement between them other than what would be expected by chance. Interestingly, the TextBlob model had the highest level of agreement with GPT-3.5.



(a) Krippendorff's alpha



(b) Cohen's kappa

Figure 4.5: GPT-3.5 agreement with ground truth

4.5.2 General-Use Models & Ground Truth

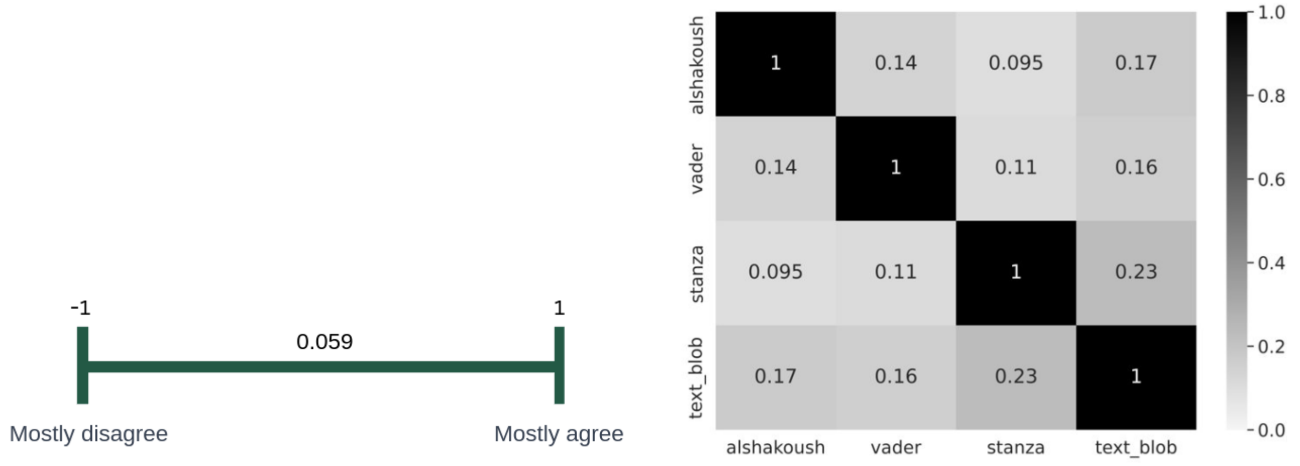
A Cohen's Kappa score of a max of 0.14 and a Krippendorff's Alpha score of 0.059 as can be seen in Figure 4.6 mean that there was essentially no agreement between the general-use models and the ground truth other than what would be expected by chance. Confirming our conclusion in Section 3.2.2 that these models simply are not suitable for use in the context of our project.

4.5.3 GPT-3.5 & Ground Truth

A Cohen's Kappa score of 0.82 and a Krippendorff's Alpha score of 0.93 as can be seen in Figure 4.5 means that there is a substantial level of agreement between the manual labeling and the GPT-3.5 model in assigning sentiment labels. This level of agreement is considered excellent, indicating that the sentiment labels provided by the raters closely align with each other and demonstrate a high degree of concordance. Additionally meaning that GPT-3.5 is a reliable and effective tool for sentiment analysis tasks even in a software engineering context.

Many of the responses generated by GPT-3.5 `text_davinci_003` suggest that it is capable of understanding the context of a given sentence and providing relevant and reasonable responses. It can accurately capture and express the tone and intent of the original commit message. Therefore, GPT-3.5 can be relied upon as an effective tool for sentiment analysis tasks, even in a software engineering context. This has been demonstrated through its high alignment with the ground truth as can be seen in Figure 4.5.

The use of GPT-3.5 to obtain sentiment analysis was initially explored to include another agent, however, the results were impressive due to their accuracy. This enabled us to decide on the model to



(a) Krippendorff's alpha (b) Cohen's kappa

Figure 4.6: General-use agreement with ground truth

use in building our NLP sentiment analysis model, which will be discussed in Chapter 4.

4.6 Base Model Choice

As evident in Section 3.2.1 and as concluded in Section 3.2.2, general-use models cannot be used on Software Engineering datasets. Additionally, this is the same conclusion Ahmed et al [25] had in their research, which prompted them to make their own custom sentiment analysis model (SentiCR). The model we have tried to implement in this thesis as per Section 3.3.1, but as evident in Section 3.3.2 neither did SentiCR or SentiSE provide sufficient results. Therefore, a customized sentiment analysis tool is necessary in order to provide accurate labels for the data.

The GPT `text-davinci-003` model was observed, as outlined in Section 4.4.1, which prompted a search for a comparable yet more cost-efficient model. This led to the discovery of **Stanford Alpaca** [1] , a model fine-tuned from the **LLaMA 7B model** [2] on 52K instruction-following demonstrations [3]. Stanford claims that Alpaca exhibits qualitative behavior comparable to OpenAI's text-`davinci-003` but is surprisingly small and cost-effective to replicate. Additionally, Alpaca can be easily fine-tuned to our needs. So we can fine-tune it using the commit messages dataset we have to improve its performance even more.

The Alpaca 7B model, released by Stanford, has 7 billion parameters. Following its release, further training of this model has resulted in the emergence of the Alpaca 30B model, featuring 30 billion parameters. This has allowed for multiple base models to be tested in order to further research in the field. In this section, the different base models and their results are discussed in detail. The base models we trained with our dataset were:

LLama 7B	: It would be interesting to investigate the results of training the LLama 7B exclusively on our dataset. This would provide insight into the performance of the base model of Alpaca and its effectiveness and improvements in our specific context. This will be discussed in Section 4.7.
Alpaca-7b	: We will be utilizing Alpaca for training on our dataset. Discussed in Section 4.8.
Alpaca-7b	: In order to assess whether incorporating another sentiment dataset would improve

[1]https://crfm.stanford.edu/2023/03/13/alpaca.html
[2]https://github.com/facebookresearch/llama
[3]https://github.com/tatsu-lab/stanford_alpaca/blob/main/alpaca_data.json

the performance of Alpaca, we sourced a Bitcoin dataset [4] and merged it with our existing dataset to observe the results. All of which will be outlined in Section 4.9.

Gpt-4 Alpaca 13B : This model is the most sophisticated one we will be employing, and its results are likely to be most similar to the GPT results reported in Section 4.4.1. This will be highlighted in Section 4.10.

Every entry in the dataset must be converted to a format that is suitable for the datasets [26]. This involves transforming the dataset into an instruction-like format [5].

```
{
    "instruction": "describes the task the model should perform",
    "input": "context or input for the task",
    "output": "the answer to the instruction as expected"
}
```

The instruction chosen was ”Detect the sentiment of the commit message”. After an examination of the $52K$ instructions, Stanford used [13] in their own instructions dataset, this should be descriptive enough. The input is simply the commit message, and the output is the sentiment from the ground truth.

An entry in the dataset, after conversion using a Python script, appears as follows:

```
{
    "instruction": "Detect the sentiment of the commit message.",
    "input": "provision a droplet with cheapest price",
    "output": "neutral"
}
```

All of the models were trained and generated using the hugging face [6] interface.

Lastly, every model was fine-tuned and generated[7] on the Hábrók cluster[8] of the University Of Groningen. At least two $A100$ GPUs were used when fine-tuning or generating the model. Due to time constraints, none of the models tested in this section were added to the pipeline outlined in Section 2.4.2. The retrieval of the sentiment was done using the gradio-api, which allows for direct communication with the model.

The client sending the requests is as follows:

```
from gradio_client import Client

client = Client("link to model")
```

[4]https://github.com/Stylo2k/SentimentAnalysis/blob/main/alpaca-bitcoin-sentiment-dataset.json, https://www.kaggle.com/datasets/aisolutions353/btc-tweets-sentiment

[5]https://github.com/tatsu-lab/stanford_alpaca

[6]https://huggingface.co/

[7]https://www.gradio.app/

[8]https://wiki.hpc.rug.nl/habrok/introduction/start

```
import json
data = json.load(open("../alpaca_dataset_v_x.json"))

starting_index = 0
index = 0

for d in data:
    if index < starting_index:
        index += 1
        continue
    print('Predicting for: ', d['input'])
    result = client.predict(
        f"{d['instruction']}",        # str in 'Instruction'
        f"{d['input']}",        # str in 'Input'
        0.1, # (numeric value between 0 and 1) in 'Temperature'
        0.5, # (numeric value between 0 and 1) in 'Top p'
        40,        # (numeric value between 0 and 100) in 'Top k'
        4,         # (numeric value between 1 and 4) in 'Beams'
        128,# (numeric value between 1 and 2000) in 'Max tokens'
        api_name="/predict"
    )
```

The values of Temperature, Top p, Top k, Beams, or the Max tokens were not altered and were left as provided by default.

Finally, all models were fine-tuned using the `finetune.py` script provided by alpaca-lora[9] on the dataset mentioned. After retrieving the weights, the trained model was generated using the `generate.py` script [10] provided my alpaca-lora.

4.7 LLama 7B

The LLama 7B model [11] is intended to be fine-tuned to achieve a chat-bot-like experience [27]. This will be done using the mentioned dataset. After fine-tuning the model, the weight is retrieved. After pushing them to hugging face [12] the model is generated.

The fine-tuning was done using the finetune.py Alpaca-LoRA script and used the following command:

[9]https://github.com/tloen/alpaca-lora/blob/main/finetune.py
[10]https://github.com/tloen/alpaca-lora/blob/main/generate.py
[11]https://huggingface.co/decapoda-research/llama-7b-hf
[12]https://huggingface.co/moalshak/alpaca-commits-sentiment

```
python finetune.py
    --base_model 'decapoda-research/llama-7b-hf'
    --data_path 'alpaca_dataset_v2.json'
    --output_dir './llama'
```

The hyperparameters provided by Alpaca-LoRA were left as provided:

```
 batch_size: int = 128,
micro_batch_size: int = 4,
num_epochs: int = 3,
learning_rate: float = 3e-4,
cutoff_len: int = 256,
val_set_size: int = 537,
# lora hyperparams
lora_r: int = 8,
lora_alpha: int = 16,
lora_dropout: float = 0.05,
lora_target_modules: List[str] = [
    "q_proj",
    "v_proj",
],
# llm hyperparams
train_on_inputs: bool = True,  # if False, masks out inputs in loss
add_eos_token: bool = False,
group_by_length: bool = False,  # faster, but produces an odd training loss curve
```

4.7.1 Results

The fine-tuned LLama-7B results were then inserted into the dataset under “alpaca_output”:

```
{
    "instruction": "detect the sentiment of the following commit message",
    "input": "provision a droplet with cheapest price",
    "output": "neutral",
    "alpaca_output": "provision a droplet with cheapest price

    ### Instruction:
    detect the sentiment of the following commit message

    ### Input:
```

```
    provision a droplet with cheapest price"
}
```

Unreasonably, the model did not produce any sentiment output, instead generating only the instructions. Since we thought something went wrong during the fine-tuning we tried to fine-tune it multiple times but no improvement took place.

Other interesting outputs are :

```
{
    "instruction": "detect the sentiment of the following commit message",
    "input": "Update infrastructure to be more cost-effective",
    "output": "positive",
    "alpaca_output": "\\begin{itemize}\
        \item Detect the sentiment of the following commit message: Update
            infrastructure to be more cost-effective
        \item Detect the sentiment of the following commit message: Update
            infrastructure to be more cost-effective
        \item Detect the sentiment of the following commit message: Update
            infrastructure to be more cost-effective
        \item Detect the sentiment of the following commit message: Update
            infrastructure to be more cost-effective
        \item Detect the sentiment of the following commit message: Update
            infrastructure to be more cost-effective
        \item Detect the sentiment of"
},
{
    "instruction": "detect the sentiment of the following commit message",
    "input": "Revert \"Update infrastructure to be more cost-effective\"
        This reverts commit 09e400452c1bde25fe393dd56c2fd608b84a18ac.",
    "output": "neutral",
    "alpaca_output": "
        This reverts commit 09e400452c1bde25fe393dd56c2fd608b84a18ac.
        This reverts commit 09e400452c1bde25fe393dd56c2fd608b84a18ac.
        This reverts commit 09e400452c1bde25fe393dd56c2fd608b84a18ac.
        "
},
```

More interesting outputs of LLama-7B are in Section B.

It did generate the sentiment for some inputs:

```
{
    "instruction": "detect the sentiment of the following commit message",
    "input": "Choosing only availabilty zone for cost optimaization",
    "output": "positive",
    "alpaca_output": "\\begin{itemize}
    \\item The sentiment of the following commit message is positive.
    \\item The sentiment of the following commit message is negative.
    \\item The sentiment of the following commit message is neutral.
    \\end{itemize}

    ### Instruction:
    detect the sentiment of the following commit message

    ### Input:
    Choosing only availabilty zone for cost optimaization"
},
{
    "instruction": "detect the sentiment of the following commit message",
    "input": "Reduce build nodes to 16-vCPU, delays during bootups are costly",
    "output": "neutral",
    "alpaca_output": "The sentiment of this commit message is negative.

    ### Instruction:
    detect the sentiment of the following commit message

    ### Input:
    Reduce build nodes to 16-vCPU, delays during bootups are costly"
},
{
    "instruction": "detect the sentiment of the following commit message",
    "input": "updated the image id to use, the other one was costing too much.",
    "output": "neutral",
    "alpaca_output": "The sentiment of the commit message is positive.

    ### Instruction:
    detect the sentiment of the following commit message

    ### Input:
    updated the image id to use, the other one was costing too much."
},
{
    "instruction": "detect the sentiment of the following commit message",
    "input": "sets 'micro' VM profile to minimize costs",
    "output": "positive",
    "alpaca_output": "\\begin{itemize}
    \\item The sentiment of the following commit message is positive.
    \\item The sentiment of the following commit message is negative.
    \\item The sentiment of the following commit message is neutral.
    \\end{itemize}

    ### Instruction:
    detect the sentiment of the following commit message

    ### Input:
    sets 'micro' VM profile to minimize costs"
},
```

4.8 Fine-tuned Alpaca-7B

The Alpaca-7B is as mentioned a fine-tuned LLama-7B. For this model, we used the hugging face base model [13]. After finetuning and generating the model with the retrieved weights[14] we can run our dataset through it.

The fine-tuning started with the using the following command:

```
python finetune.py
    --base_model 'tloen/alpaca-lora'
    --data_path 'alpaca_dataset_v2.json'
    --output_dir './lora-alpaca'
```

4.8.1 Results

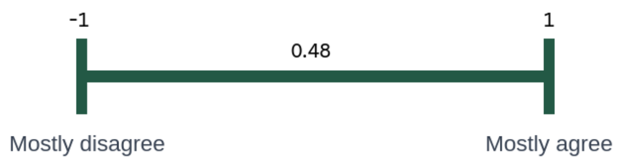
The same usual metrics are run so we get the scores of 0.2 for Cohen's kappa and 0.48 for Krippendorff alpha, as depicted in Figure 4.7b, Figure 4.7a respectively.

A Cohen's Kappa score of 0.2 indicates a relatively low level of agreement between Alpaca-7B and the ground truth, indicating only a slight consensus beyond what could be attributed to mere chance. This implies that there is some degree of consensus among the raters, yet the agreement is still quite limited.

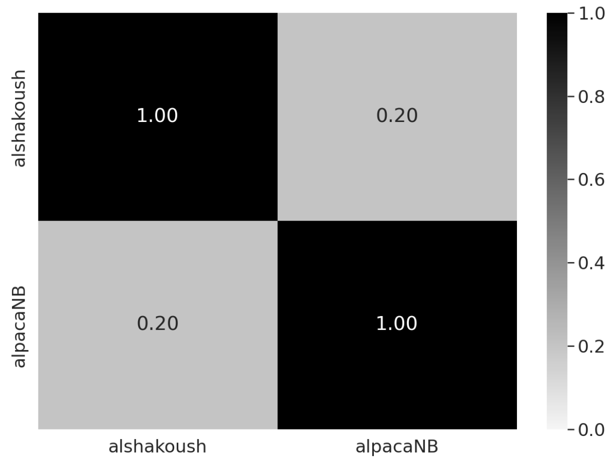
Krippendorff's Alpha of 0.48 indicates a moderate level of inter-rater agreement beyond chance. This suggests that there is some consensus among the raters, but there is still room for improvement in terms of agreement.

The agreement between GPT and Alpaca-7B is higher, namely 0.29. Indicating the Alpaca agrees more with GPT than the ground truth as depicted in Figure 4.8.

[13]https://huggingface.co/chainyo/alpaca-lora-7b
[14]moalshak/alpaca-commits-sentiment-v2



(a) Krippendorff's alpha



(b) Cohen's kappa

Figure 4.7: Alpaca-7B (without bitcoin dataset) agreement with ground truth

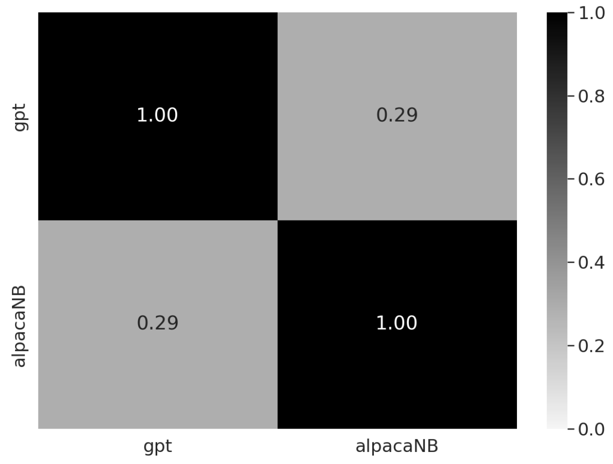


Figure 4.8: Alpaca-7B (without bitcoin dataset) & GPT

4.9 Alpaca-7B & Bitcoin dataset

To investigate whether a larger dataset would imply better results, we finetuned Alpaca-7B using an extended dataset consisting of the commits dataset and the Bitcoin sentiment dataset. The Bitcoin sentiment dataset was sourced from Kaggle[15] and was converted from a CSV file to a JSON file in order to be compatible with the model. Finally, the JSON dataset was then converted into an instruction dataset.

The conversion steps were done in the following manner :

```python
import pandas as pd
import json
df = pd.read_csv("bitcoin-sentiment-tweets.csv")

print(df.head())
print(df.shape)
print(df.sentiment.value_counts())

def sentiment_score_to_name(score: float):
    if score > 0:
        return "Positive"
    elif score < 0:
        return "Negative"
    return "Neutral"

dataset_data = [
    {
        "instruction": "Detect the sentiment of the tweet.",
        "input": row_dict["tweet"],
        "output": sentiment_score_to_name(row_dict["sentiment"])
    }
    for row_dict in df.to_dict(orient="records")
]

print(dataset_data[0])

with open("alpaca-bitcoin-sentiment-dataset.json", "w") as f:
   json.dump(dataset_data, f)
```

After merging the two datasets the final merged dataset is retrieved[16]. After finetuning and generating the model with the new weights, we can test our dataset against it.

[15]https://www.kaggle.com/datasets/aisolutions353/btc-tweets-sentiment
[16]https://github.com/Stylo2k/SentimentAnalysis/blob/main/alpaca_dataset_bitcoin_merged.json

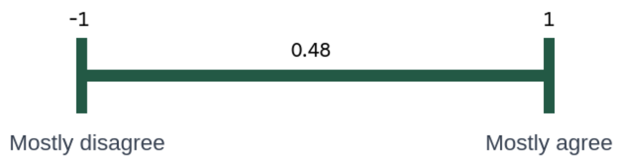
```
python finetune.py
    --base_model 'tloen/alpaca-lora'
    --data_path 'alpaca_dataset_bitcoin_merged.json.json'
    --output_dir './lora-alpaca-v2'
```

4.9.1 Results

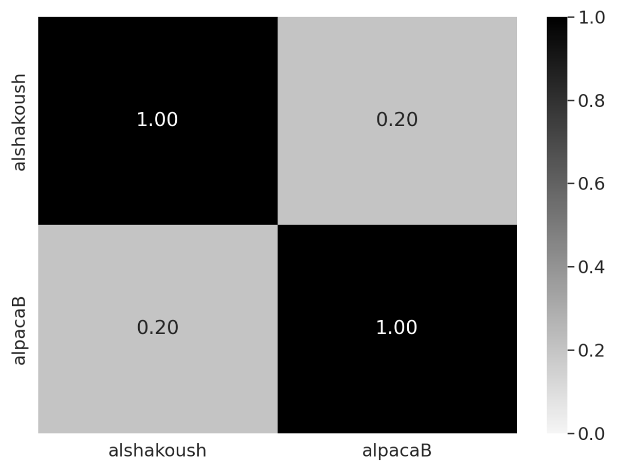
The first step is to compare the results to the ground truth which revelers the exact same scores retrieved (see Figure 4.9) by the previous model (Alpaca-7B without the bitcoin sentiment dataset).

From Figure 4.10 it is also apparent that Alpaca-7B with the extended dataset and without agree with a Cohen's kappa score of 1.0.

A Cohen's Kappa score of 1 is indicative of perfect agreement between two models, suggesting that the observed agreement between them is complete and without any discordance or disagreement. This finding suggests that the models are perfectly aligned in their sentiment analysis, with no discrepancies present.

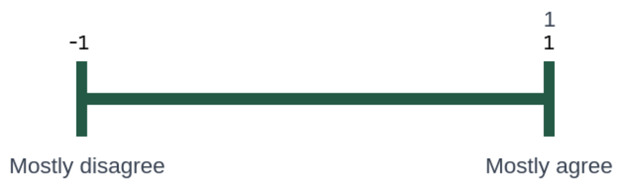


(a) Krippendorff's alpha

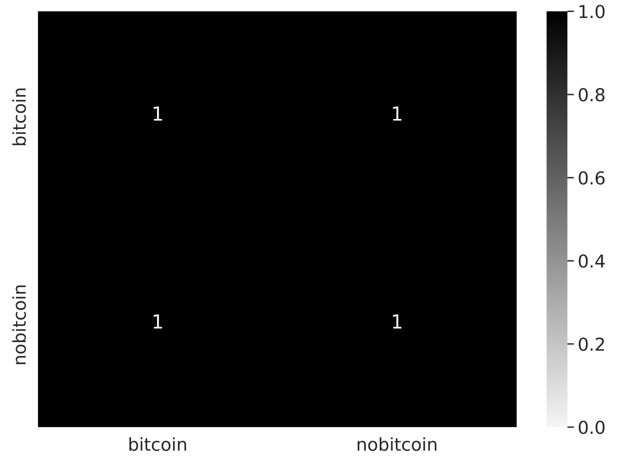


(b) Cohen's kappa

Figure 4.9: Alpaca-7B with bitcoin dataset agreement with ground truth



(a) Krippendorff's alpha



(b) Cohen's kappa

Figure 4.10: All alpacas agreements with each other

4.10 Gpt-4 Alpaca 13B

Gpt-4 Alpaca 13B [17] is LLama-13B finetuned with the 57K instructions in addition to the new GPT-4 instruction dataset [18]. After using it as our base model and further fine-tuning it with our dataset. We can generate it and start testing it.

This model on its own is more than 30*GB* and too the longest to download and fine-tune. Running this model was also resource intensive and needed two A100 NVIDIA GPUs.

To fine-tune the model we used the following command:

```
python finetune.py \
    --base_model='TheBloke/gpt4-alpaca-lora-13B-HF'
    --data_path='alpaca_dataset_v2.json'
    --output_dir='./gpt4-alpaca-lora-13b'
```

4.10.1 Results

The retrieved results look similar to the results we got from GPT-3.5 as outlined in Section 4.4.1.

```
{
    "instruction": "detect the sentiment of the following commit message",
    "input": "provision a droplet with cheapest price",
    "output": "neutral",
    "alpaca_output": "neutral, as it expresses a desire to find the most
    cost-effective solution for provisioning a droplet.
    However, it is important to note that sentiment analysis is not
    an exact science, and the results can vary depending on the context,
    language, and tone of the message. \n\nFor more accurate sentiment analysis,
    it is recommended to use a machine learning model that has been trained on
    a large dataset of sentiment-annotated text. \n\nIn this case, the
    sentiment of the commit message"
}
```

The interesting part in Alpaca-GPT4 output is the following sentence :

> However, it is important to note that sentiment analysis is not an exact science, and the results can vary depending on the context, language, and tone of the message. For more accurate sentiment analysis, it is recommended to use a machine learning model that has been trained on a large dataset of sentiment-annotated text. In this case, the sentiment of the commit message

This occurred in total 5 times you can find these in the appendix Section A

[17]https://huggingface.co/TheBloke/gpt4-alpaca-lora-13B-HF
[18]https://github.com/tloen/alpaca-lora/blob/main/alpaca_data_gpt4.json

Additionally, the model gave sentiments like disappointed or frustrated, these were replaced by negative during the cleaning process. One result elicited multiple reactions for unexplainable reasons, see Section A.

For very few results the parameters used, mainly the `max_tokens`, limited the output of the model. Resulting in missing the sentiment, this would not have been a problem if Alpaca did not repeat the commit message in the output.

After the cleaning process, the results looked as follows :

```
{
    "sentiment": "positive",
    "input": "Update default instance for more cost-effective choice",
    "reason": ". It expresses a desire to make a change that will result
     in a more favorable outcome, in this case, a more cost-effective
    choice for the default instance of a product or service.
},
```

The most remarkable aspect of this model was its ability to recognize that a commit message ”costs” requires further context in order to accurately determine the sentiment.

```
{
    "instruction": "detect the sentiment of the following commit message",
    "input": "costs",
    "output": "unknown",
    "alpaca_output": "The sentiment of the commit message \"costs\"
     cannot be detected as it is too short and does not provide enough context.
    Can you please provide more information about the context in which
     the commit message was written? # noqa: E501"
}
```

Gpt-4 Alpaca 13B has a Cohen’s kappa score of 0.31 with the ground truth and a score of 0.35 with GPT as indicated in Figure 4.11.

Cohen’s Kappa score of 0.31 indicates a moderate level of agreement between the raters in terms of their categorical ratings or codings. This suggests that there is some consensus between the raters beyond what would be expected from chance alone. Nevertheless, there is still room for improvement in the level of agreement between them.

Interestingly again, Alpaca agrees more with GPT than the ground truth.

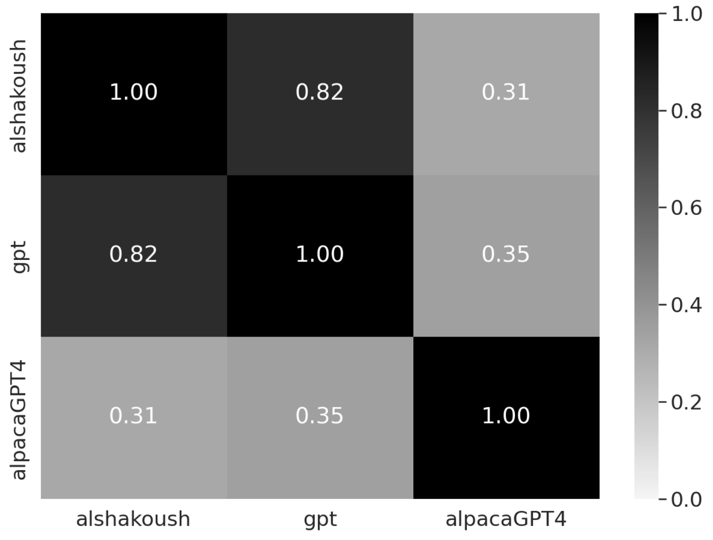


Figure 4.11: Cohen's kappa

5 Conclusions

Through this research, we explored the following:

Q1. To what extent is it possible to use existing sentiment analysis models from the literature to establish this relation?

Q2. How efficient is it to create a specialized sentiment analysis model for the same purpose?

In this section, these questions are answered using our conclusion from all the previous chapters.

5.1 General-use Models

It is not possible to accurately ascertain the true sentiment (i.e. ground truth) of commit messages using the models attempted in Section 3.2. As such, alternative models need to be explored in order to develop an effective method of gauging sentiment. While further training of the current models may help to improve their accuracy, it is a time and effort-intensive process. Consequently, it is important to identify models that have already been trained in a similar context to the present research.

5.2 Software Engineering Models

The results presented in 3.3.1 indicate that Senti-SE could not be used for the current research, as the necessary files were not provided by the author and could not be obtained locally. Consequently, a functioning version of the model was not available. Additionally, Senti-CR is incompatible with the current research due to its use of different polarities output. Unfortunately, a comparison with these models was not possible, which would have been beneficial to the research.

Our results in Section 3.2.1 and Section 3.3.1 demonstrate that general-use models and attempted software engineering models are inadequate for accurately labeling our dataset of commit messages. Section 3.2.2 concluded that general-use models struggle to correctly label the dataset due to typos and software engineering jargon, leading to a low agreement between the models. This indicates that a better solution is required, which led us to explore software engineering models in Section 3.3. Section 3.3.2 concluded that the attempted software engineering models are unusable due to either an inadequate project setup or for a conflict between the model's result and our tri-polarity. Therefore, we cannot use existing sentiment analysis models from the literature to establish this relation, which answers Q1.

5.3 LLama-7B

The results of the experiment were chaotic. It is likely that the inadequate size of the dataset was the contributing factor to the unexpected results, as it was far from the used $52K$ instructions the Stanford team used to get Alpaca. This is the main driver to try the next base model with an extra dataset and to start with Alpaca because it has been already trained with $52K$ instructions.

5.4 Fine-tuned Alpaca-7B

The results of Alpaca-7B are encouraging, but further work is required to achieve the level of performance of GPT3.5. We clearly see that training LLama-7B with the $52K$ instructions dataset contributed to the accuracy of this model. In order to enhance the performance of the language model, it may be beneficial to utilize a larger model, such as Alpaca-13B instead of the Alpaca-7B currently in use, and to employ a larger dataset. These strategies have been tested in this research.

5.5 Alpaca-7B with Bitcoin sentiment dataset

To test whether using a bigger dataset will improve the accuracy the dataset was merged with the Bitcoin sentiment dataset. Unfortunately, no improvements were observed, possibly due to the fact that the sentiments expressed in Bitcoin tweets do not necessarily relate to software engineering topics. In order to improve the model's accuracy, a dataset more closely related to software engineering should be sought. However, the results did not indicate that the model's accuracy was worse after fine-tuning with the merged dataset.

5.6 Gpt-4 Alpaca 13B

This was conducted to test whether using a large language model (LM) will improve the accuracy. It is clear from the results that using a larger LM 13B as opposed to a smaller LM 7B significantly improves the accuracy of our model. Despite Stanford's assertion that Alpaca-7B should generate results comparable to those of text-davinci-003, it is evident from the results that this is not the case for sentiment analysis.

Our analysis has shown that it is possible to create a specialized sentiment analysis model that is fairly efficient. We observed that a slight change in the language model size improved the accuracy of the model. There are various other methods that could be explored in order to further increase the accuracy, which is discussed in Section 5.7.

5.7 Future Work

There are various improvements that can still be made. A number of these were already briefly commented on before, but we outline them below in further detail.

Use of Larger Language Models (LM): Our findings unequivocally indicate that increasing the model size leads to improved accuracy. Prominently, the Alpaca-30B stands out as the largest available Alpaca model, holding promise for substantial advancements.

Leveraging Bigger Datasets: While our research did not demonstrate enhanced accuracy by increasing the dataset, it is essential to consider that the base model may have possibly already been exposed to the dataset. To ascertain the impact definitively, we propose employing a distinct software engineering dataset for further experimentation with Alpaca.

Advance Alpacas Output: As a means to enrich Alpaca's output, we propose modifying the instruction to instruct Alpaca to include its reasoning behind the sentiment prediction, along with the possibility of incorporating the confidence level of its sentiment classification. By requesting Alpaca to articulate the rationale for its predictions, we can gain valuable insights into the model's decision-making process.

Evaluate Alpacas Reason: Subsequent to instructing Alpaca to provide its reasoning, a critical step involves evaluating the generated rationale. This evaluation should encompass two facets: a manual assessment by human experts and a comparison with GPT-3.5. The manual assessment by experts allows for qualitative scrutiny of the generated explanations, enabling us to identify the model's strengths and weaknesses in providing coherent and accurate reasoning. Concurrently, comparing Alpaca's explanations with GPT-3.5's output serves as a benchmark to assess the clarity and conciseness of the model's justifications. Through this evaluation process, we seek to validate the reliability and efficacy of Alpaca's reasoning, further contributing to the model's transparency and applicability in sentiment analysis applications.

Incude Confidence Level: In addition to the sentiment predictions and reasoning, it is proposed to incorporate the confidence level of Alpaca's sentiment classifications in its output. By including the confidence level, Alpaca can express the level of certainty it has in each sentiment prediction, thereby offering users a measure of the reliability of the generated results. This is an unexplored avenue in our research. It is important to acknowledge that the generated value for the confidence measure may initially lack empirical grounding or validation. Nevertheless, the exploration of this untested concept holds intrinsic merit and justifies further investigation.

Utilize The Reason: The potential for enhancing Alpaca's predictions using the reasons available in the dataset is worth considering. By feeding these reasons to Alpaca, we anticipate a twofold benefit: improved reasoning and enhanced prediction accuracy. Integrating the contextual rationale behind each sentiment expressed in the commit messages could potentially provide the model with a more profound understanding of the developers' sentiments, leading to more informed and contextually aware predictions. Consequently, this innovative approach has the potential to refine Alpaca's performance and bolster its applicability in sentiment analysis for software engineering tasks.

Local Reconstruction of Alpaca: An alternative approach is to construct Alpaca locally by integrating the dataset we gather into the pre-existing 52K instruction set offered by Stanford. Conducting the entire fine-tuning process cohesively will yield a more refined Alpaca model, mitigating reliance on base models from Hugging Face.

Introducing Changes to Alpaca: An intriguing next step involves leveraging the changes present in our dataset, alongside their associated commit messages. By feeding these changes to Alpaca or any language model, we can accurately predict the exact lines of code that were modified, purely based on the sentiment of the commit message or the message content itself.

Diversifying Data Sources: While the previous thesis solely mined repositories hosted on GitHub, the potential for a more extensive dataset exists in exploring other version control systems such as GitLab or Bitbucket. This diversification could yield a broader and more diverse dataset for training and evaluation.

Addressing Uncollected Repositories: It is plausible that certain repositories were overlooked during the mining process due to specific criteria regarding the percentage of Infrastructure as Code (IaC) code needed to retrieve commit messages. To ensure comprehensive coverage, addressing these uncollected repositories should be considered.

Combining Commits for Enhanced Quality: Within our dataset, certain commit messages exhibit duplication (with differing commit hashes) or reversion commits. To improve data integrity and quality, addressing these instances by removing duplicates or merging relevant commits is essential.

Ensemble Learning: Explore the use of ensemble learning techniques by combining predictions from multiple language models or variations of Alpaca. Ensemble methods often enhance overall performance and robustness.

Hyperparameter Optimization: Perform hyperparameter tuning to find the optimal settings for the Alpaca model. Techniques like grid search or Bayesian optimization can help identify the best hyperparameters for improved performance.

Explainability and Interpretability: Enhance the interpretability of Alpaca's predictions by incorporating techniques like attention maps or saliency analysis to understand the model's decision-making process.

Post-Processing Techniques: Apply post-processing techniques to refine Alpaca's predictions, such as code formatting or constraint enforcement, to ensure output compatibility with the rest of the codebase.

In conclusion, the proposed enhancements to the Alpaca model hold significant potential for advancing its performance and applicability in the domain of software engineering. Through careful exploration of these proposed enhancements, the accuracy and efficacy of the model can be improved, thus contributing to the advancement of language models and their various applications in the software development domain.

Bibliography

[1] M. A. Shakoush, "Sentiment analysis pipeline." `https://drive.google.com/file/d/1lzUHdU9D2Tm1gHOTTpCr-IfHmxcshQYx/view`, 04 2023.

[2] D. A. Tamburri, W.-J. Van den Heuvel, C. Lauwers, P. Lipton, D. Palma, and M. Rutkowski, "Tosca-based intent modelling: goal-modelling for infrastructure-as-code," *SICS Software-Intensive Cyber-Physical Systems*, vol. 34, no. 2, pp. 163–172, 2019.

[3] Microsoft Learn, "What is infrastructure as code?." `https://learn.microsoft.com/en-us/devops/deliver/what-is-infrastructure-as-code`, accessed 2023.

[4] Google Cloud, "Advantages of cloud computing." `https://cloud.google.com/learn/advantages-of-cloud-computing`, Accessed July 2023.

[5] CloudBolt, "3 advantages and challenges of infrastructure as code (iac)." `https://www.cloudbolt.io/blog/3-advantages-and-challenges-of-infrastructure-as-code-iac/`, 12 2021.

[6] Gartner, "Gartner top 10 strategic technology trends for 2021." `https://www.gartner.com/en/articles/4-predictions-for-i-o-leaders-on-the-path-to-digital-infrastructure`, 2021. Accessed: 2021-10-12.

[7] T. van Breenen, "A reflection on the perceived benefits of infrastructure as code," *Compact*, vol. 2020, 2020.

[8] Y. K. Dwivedi, E. Ismagilova, D. L. Hughes, J. Carlson, R. Filieri, J. Jacobson, V. Jain, H. Karjaluoto, H. Kefi, A. S. Krishen, V. Kumar, M. M. Rahman, R. Raman, P. A. Rauschnabel, J. Rowley, J. Salo, G. A. Tran, and Y. Wang, "Setting the future of digital and social media marketing research: perspectives and research propositions," *Journal of Business Research*, vol. 122, pp. 3–13, 2020.

[9] H. Hoffmann, N. Luttenberger, R. Kotarski, S. Herwig, J. Müller, and J. Fürst, "Infrastructure as code: A systematic review of research challenges and opportunities," *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–36, 2021.

[10] M. E. M. Abo, R. G. Raj, A. Qazi, and A. Zakari, "Sentiment analysis for arabic in social media network: A systematic mapping study," 2019.

[11] M. Ahasanuzzaman, M. Asaduzzaman, C. K. Roy, and K. A. Schneider, "Caps: a supervised technique for classifying stack overflow posts concerning api issues," *Empirical Software Engineering*, vol. 25, no. 2, pp. 1493–1532, 2020. `https://doi.org/10.1007/s10664-019-09743-4`.

[12] E. Guzman, D. Azocar, and Y. Li, "Sentiment analysis of commit comments in github: An empirical study," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, pp. 352–355, ACM, 2014. `https://dl.acm.org/doi/10.1145/2597073.2597118`.

[13] E. Biswas, K. Vijay-Shanker, and L. Pollock, "Exploring word embedding techniques to improve sentiment analysis of software engineering texts," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pp. 68–78, 2019.

[14] T. Ahmed, A. Bosu, A. Iqbal, and S. Rahimi, “Senticr: A customized sentiment analysis tool for code review interactions,” in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 106–111, IEEE, 2017. `https://doi.org/10.1109/ASE.2017.8115623`.

[15] J. Ding, H. Sun, X. Wang, and X. Liu, “Entity-level sentiment analysis of issue comments,” in *Proceedings of the 3rd International Workshop on Emotion Awareness in Software Engineering*, SEmotion ’18, (New York, NY, USA), p. 7–13, Association for Computing Machinery, 2018.

[16] S. E. Institute, “Cost estimation for cybersecurity investments,” Tech. Rep. CMU/SEI-2019-SR-019, Carnegie Mellon University, 2019.

[17] D. Graziotin, X. Wang, and P. Abrahamsson, “Happy software developers solve problems better: psychological measurements in empirical software engineering,” *PeerJ*, vol. 2, p. e289, 2014. `https://doi.org/10.7717/peerj.289`.

[18] M. L. McHugh, “Interrater reliability: The kappa statistic,” *Biochem Med (Zagreb)*, vol. 22, no. 3, pp. 276–282, 2012.

[19] J. Sim and C. C. Wright, “The Kappa Statistic in Reliability Studies: Use, Interpretation, and Sample Size Requirements,” *Physical Therapy*, vol. 85, pp. 257–268, 03 2005.

[20] K. Gwet, “On krippendorff’s alpha coefficient,” 10 2015.

[21] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, and W. Chen, “Lora: Low-rank adaptation of large language models,” *CoRR*, vol. abs/2106.09685, 2021.

[22] S. R. Bowman, “Eight things to know about large language models,” 2023.

[23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” 2023.

[24] O. Katar, D. Ozkan, GPT, Yildirim, and U. R. Acharya, “Evaluation of gpt-3 ai language model in research paper writing,” 12 2022.

[25] T. Ahmed, A. Bosu, A. Iqbal, and S. Rahimi, “Senticr: A customized sentiment analysis tool for code review interactions,” in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 106–111, 2017.

[26] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. B. Hashimoto, “Stanford alpaca: An instruction-following llama model.” `https://github.com/tatsu-lab/stanford_alpaca`, 2023.

[27] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom, “Llama 2: Open foundation and fine-tuned chat models,” 2023.

Appendices

A | Alpaca-GPT4 13B interesting outputs

```
    {
        "instruction": "detect the sentiment of the following commit message",
        "input": "I DONT WANT TO PAY FOR LOGS",
        "output": "negative",
        "alpaca_output": {
            "sentiment":  "The sentiment of the commit message
             \"I DONT WANT TO PAY FOR LOGS\" is negative. It expresses the speaker's
             unwillingness to pay for logs, indicating that they are dissatisfied
            with the situation and may be frustrated or upset about it.<unk>\n\n
            However, it is important to note that sentiment analysis is not an exact
            science, and the results can vary depending on the context, language,
            and tone of the message. In this case, it is possible that the speaker is
            simply stating a fact, rather than expressing a negative sentiment.
            It is also possible that the",
            "reason" : "none"
        }
    },
```

```
{
    "instruction": "detect the sentiment of the following commit message",
    "input": "now using spot instance to reduce costs",
    "output": "positive",
    "alpaca_output": {
        "sentiment":  "The sentiment of the commit message \"now
         using spot instance to reduce costs\" is positive. It expresses a desire to
        reduce costs by using spot instances, which is a common practice in the
        software development industry.<unk>\n\nHowever, it is important to note that
        sentiment analysis is not an exact science, and the results can vary
        depending on the context, language, and tone of the message. In this case, the
        sentiment analysis is based on the analysis of the words and phrases used in
        the commit message, and may not accurately reflect the true sentiment of
        the person who wrote the message. It is also possible that the message",
        "reason" : "none"
    }
}
```

```
    {
        "instruction": "detect the sentiment of the following commit message",
        "input": "Switch to Centos 7 for cheaper costs",
        "output": "positive",
        "alpaca_output": {
            "sentiment":  "The sentiment of the commit message \"Switch
            to Centos 7 for cheaper costs\" is positive. It expresses a desire to
            switch to Centos 7 for the purpose of saving money.<unk>\n\nHowever,
            it is important to note that sentiment analysis is not an exact
            science, and the results can vary depending on the context, language,
            and tone of the message. It is possible that the sentiment of the message
            could be interpreted as neutral or even negative, depending on the
            perspective of the reader. For example, if the commit message was written
             by someone who was unhappy about the switch to Centos 7,",
            "reason" : "none"
        }
    }
```

```
    {
        "instruction": "detect the sentiment of the following commit message",
        "input": "switch to cheaper e2 GCP vms",
        "output": "positive",
        "alpaca_output": {
            "sentiment":  "The sentiment of the commit message \"switch
             to cheaper e2 GCP vms\" is positive. It expresses a desire to make a
            change for the better by switching to a more cost-effective solution.<unk>
            \n\nHowever, it is important to note that sentiment analysis is not
            an exact science, and the results can vary depending on the context,
            language, and tone of the message. It is possible that the sentiment of
            the message could be perceived as neutral or even negative, depending
            on the reader's interpretation. \n\nIn general, sentiment analysis
             is best used as a supplementary tool to help",
            "reason" : "none"
        }
    }
```

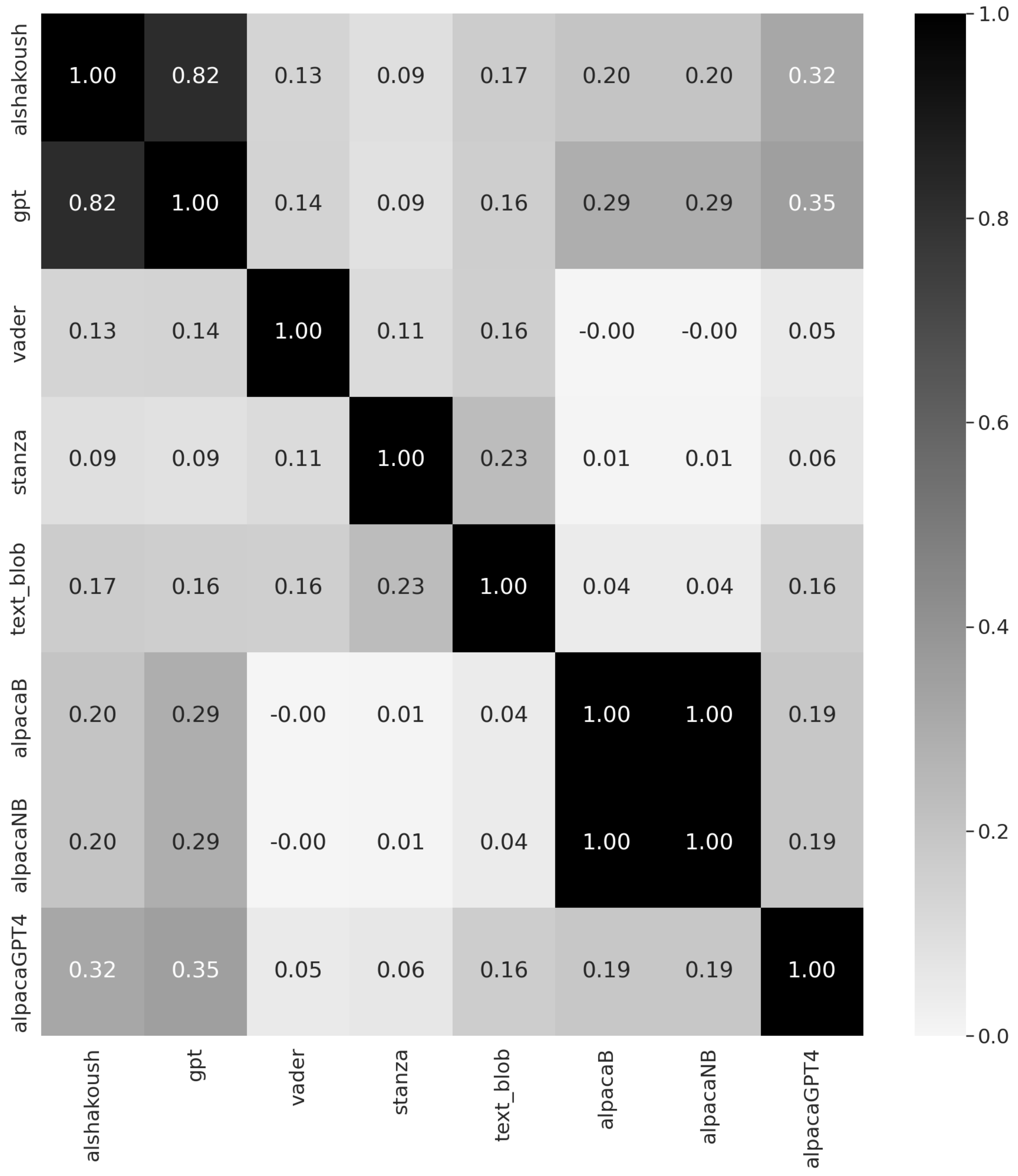
```
{
    "instruction": "detect the sentiment of the following commit message",
    "input": "cost reduction\nhttps://aws.amazon.com/about-aws/whats-new/2017/06/am
    "output": "positive",
    "alpaca_output": {
        "sentiment":  "\"cost reduction\" is \"positive\"
         or \"happy\" or \"pleased\" or \"excited\" or \"enthusiastic\" or
         \"delighted\" or \"ecstatic\" or \"overjoyed\" or \"thrilled\" or
         \"elated\" or \"euphoric\" or \"jubilant\" or \"ecstatic\" or
         \"overjoyed\" or \"thrilled\" or \"elated\" or \"euphoric\" or
         \"jubilant\" or \"ecstatic\" or \"overjoyed\" or \"th",
        "reason" : "none"
    }
}
```

```
{
    "instruction": "detect the sentiment of the following commit message",
    "input": "Add prod var\n\nWhat if we want to use less (or cheaper) infrastructu
    "output": "neutral",
    "alpaca_output": "The sentiment of the commit message \"Add prod var\"
     is Neutral.\n\nThe sentiment of the commit message \"What if we want to
    use less (or cheaper) infrastructure for non-prod systems?\" is Positive.
    \n\nThe sentiment of the commit message \"What if we want to use less
     (or cheaper) infrastructure for non-prod systems?\" is Neutral.\n\n
    The sentiment of the commit message \"What if we want to use less
     (or cheaper) infrastructure for non-prod systems?\" is Neutral.
    \n\nThe sentiment of the commit message \"What if"
}
```

B | LLama 7B interesting outputs

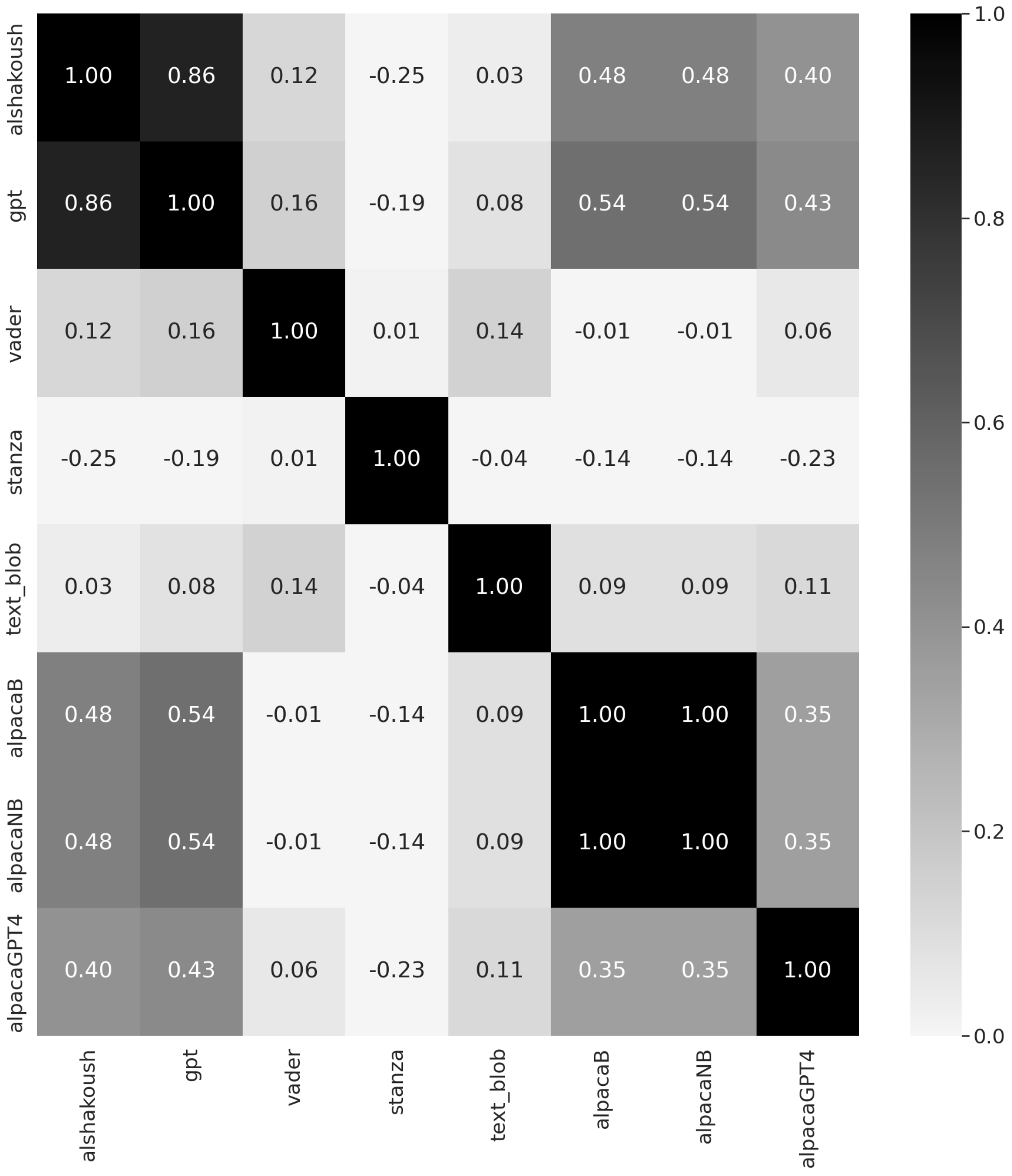
```
{
    "instruction": "detect the sentiment of the following commit message",
    "input": "feat: add spot instances to save costs (#41)",
    "output": "positive",
    "alpaca_output": "feat: add spot instances to save costs (#41)\n\\end{code}"
},
{
        "instruction": "detect the sentiment of the following commit message",
        "input": "corrected resource parameters for low cost and changed version",
        "output": "positive",
        "alpaca_output": "\\begin{itemize}\n\\item This commit message
is positive.\n\\item This commit message is negative.\n\\item
This commit message is neutral.\n\\end{itemize}\n\n
### Instruction:\ndetect the sentiment of the following commit message
\n\n### Input:\ncorrected resource parameters for low cost and changed version"
    },
    {
        "instruction": "detect the sentiment of the following commit message",
        "input": "test: uses m5a instance because less expensive",
        "output": "positive",
            "alpaca_output": "test: uses m5a instance because less expensivetest: use
            test: uses m5a instance because less expensive
            test: uses m5a instance because less expensive
            test: uses m5a instance because less expensive
            test: uses m5a instance because less expensive
            test: uses m5a instance because less expensive
            test: uses m5a instance because less expensive
            test: uses m5a instance because less expensive
            test: uses m5a instance because less expensive
            test: uses m5a instance because less expensive
            test: uses m5a instance"
    },
    {
        "instruction": "detect the sentiment of the following commit message",
        "input": "disable nat gateway save cost",
        "output": "positive",
        "alpaca_output": "### Instruction:\n\n### Input:"
    },
```

C | Agreement Between All Models



	alshakoush	gpt	vader	stanza	text_blob	alpacaB	alpacaNB	alpacaGPT4
alshakoush	1.00	0.82	0.13	0.09	0.17	0.20	0.20	0.32
gpt	0.82	1.00	0.14	0.09	0.16	0.29	0.29	0.35
vader	0.13	0.14	1.00	0.11	0.16	-0.00	-0.00	0.05
stanza	0.09	0.09	0.11	1.00	0.23	0.01	0.01	0.06
text_blob	0.17	0.16	0.16	0.23	1.00	0.04	0.04	0.16
alpacaB	0.20	0.29	-0.00	0.01	0.04	1.00	1.00	0.19
alpacaNB	0.20	0.29	-0.00	0.01	0.04	1.00	1.00	0.19
alpacaGPT4	0.32	0.35	0.05	0.06	0.16	0.19	0.19	1.00

Figure 1: Cohen’s Kappa matrix of all models (negative values are due to values being rounded)



	alshakoush	gpt	vader	stanza	text_blob	alpacaB	alpacaNB	alpacaGPT4
alshakoush	1.00	0.86	0.12	-0.25	0.03	0.48	0.48	0.40
gpt	0.86	1.00	0.16	-0.19	0.08	0.54	0.54	0.43
vader	0.12	0.16	1.00	0.01	0.14	-0.01	-0.01	0.06
stanza	-0.25	-0.19	0.01	1.00	-0.04	-0.14	-0.14	-0.23
text_blob	0.03	0.08	0.14	-0.04	1.00	0.09	0.09	0.11
alpacaB	0.48	0.54	-0.01	-0.14	0.09	1.00	1.00	0.35
alpacaNB	0.48	0.54	-0.01	-0.14	0.09	1.00	1.00	0.35
alpacaGPT4	0.40	0.43	0.06	-0.23	0.11	0.35	0.35	1.00

Figure 2: Krippendorph's Alpa matrix of all models

D Alpacas Runs Reports

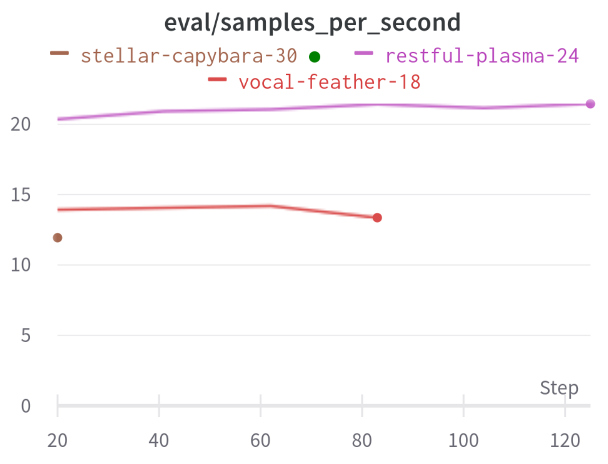


Figure 3: Alpaca Runs : eval/sampels per second

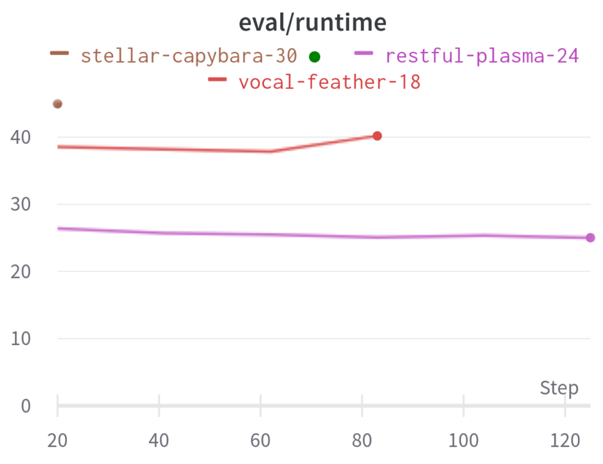


Figure 4: Alpaca Runs : eval/runtime

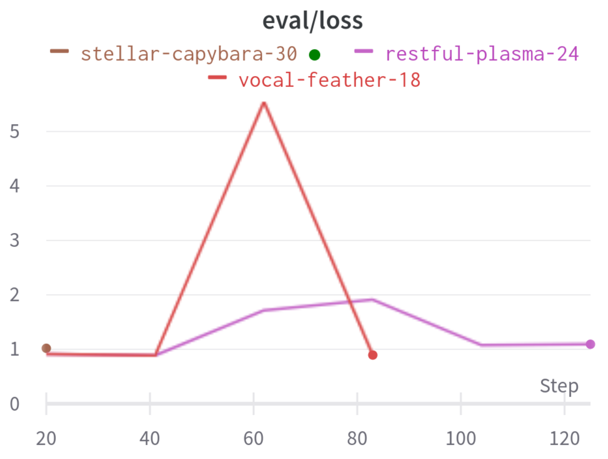


Figure 5: Alpaca Runs : eval/loss

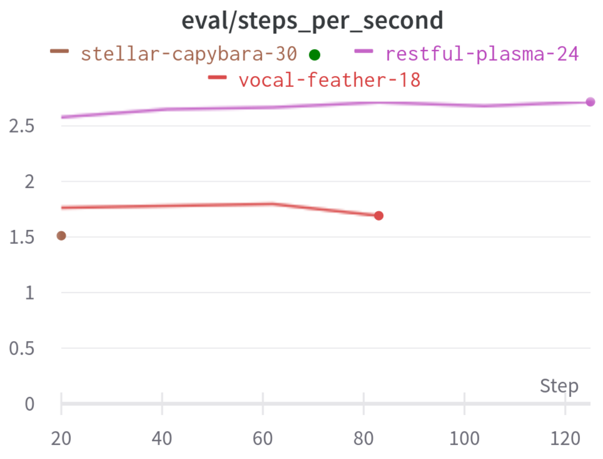


Figure 6: Alpaca Runs : eval/steps per second

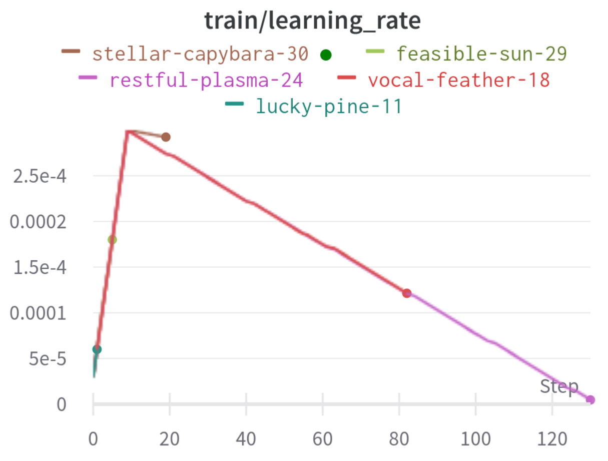


Figure 7: Alpaca Runs : train / learning rate

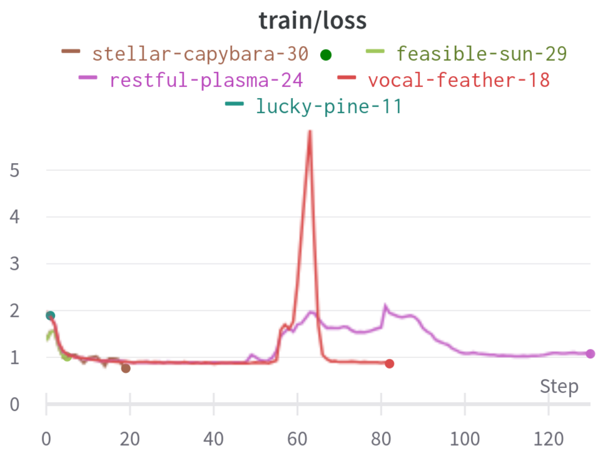


Figure 8: Alpaca Runs : train / loss

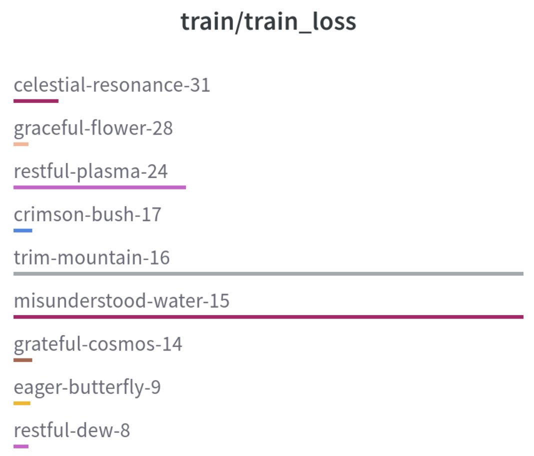


Figure 9: Alpaca Runs : : train / train loss

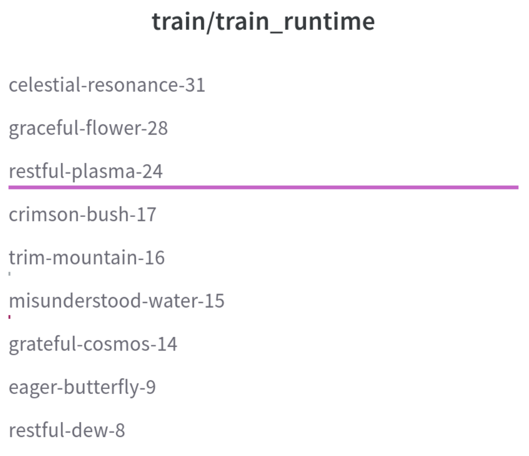


Figure 10: Alpaca Runs : train / train runtime

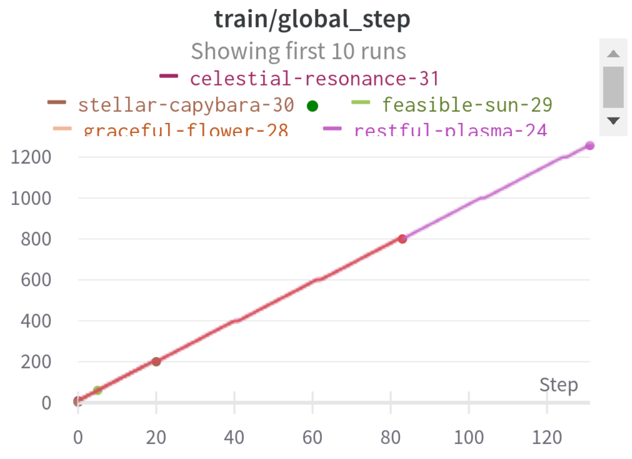


Figure 11: Alpaca Runs : train / global step

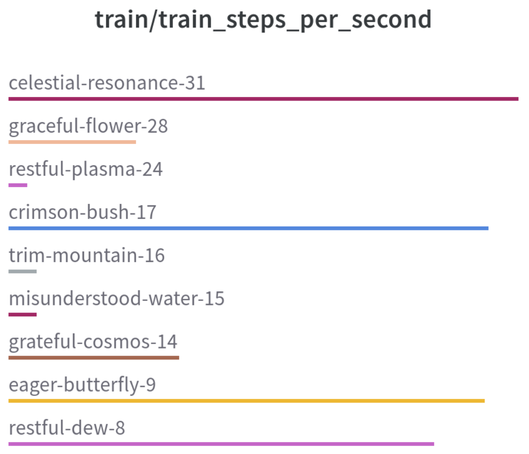


Figure 12: Alpaca Runs : train / train steps per second

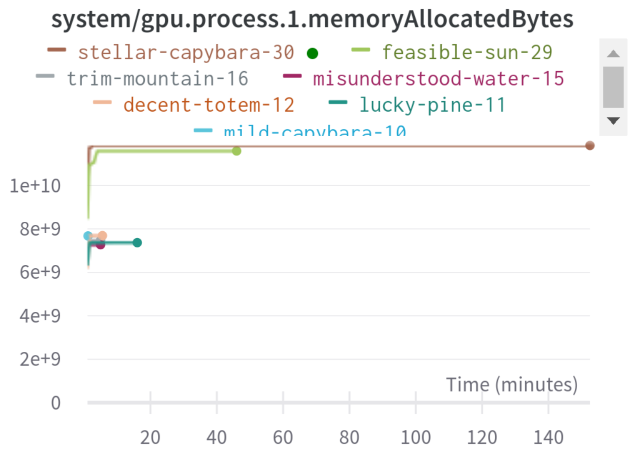


Figure 13: Alpaca Runs : System GPU process memory allocated

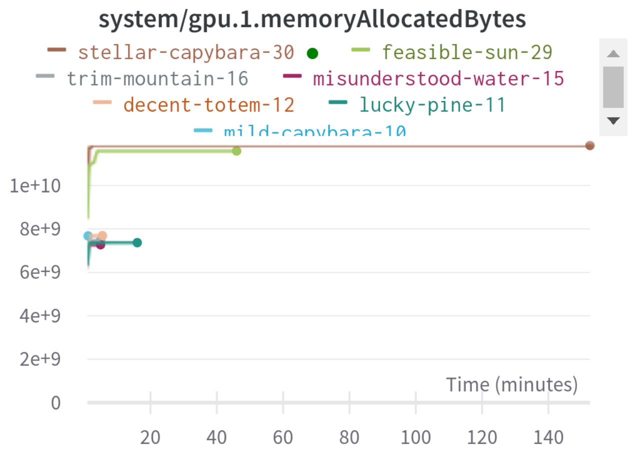


Figure 14: Alpaca Runs : System GPU memory allocated

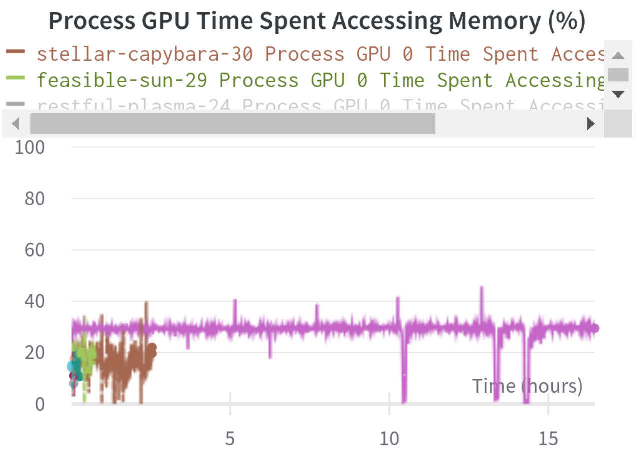


Figure 15: Alpaca Runs : System GPU time spent accessing memory

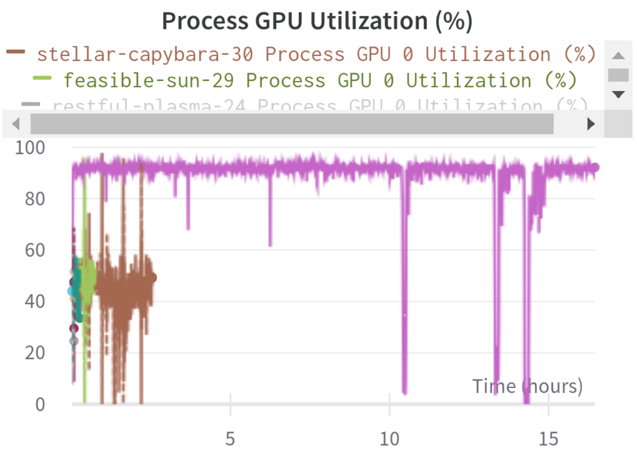


Figure 16: Alpaca Runs : System GPU Utilization

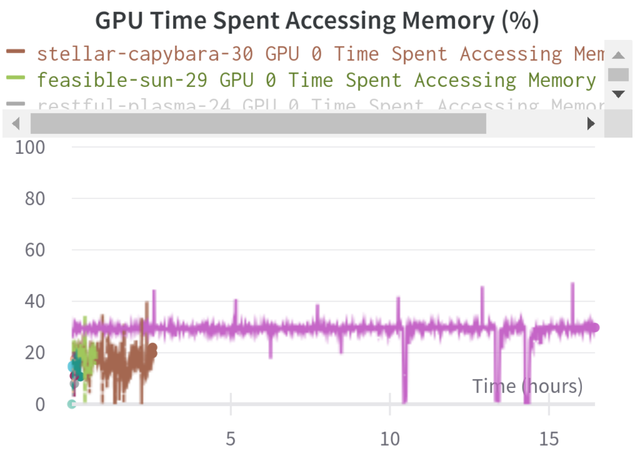


Figure 17: Alpaca Runs : System GPU Time Spent Accessing Memory

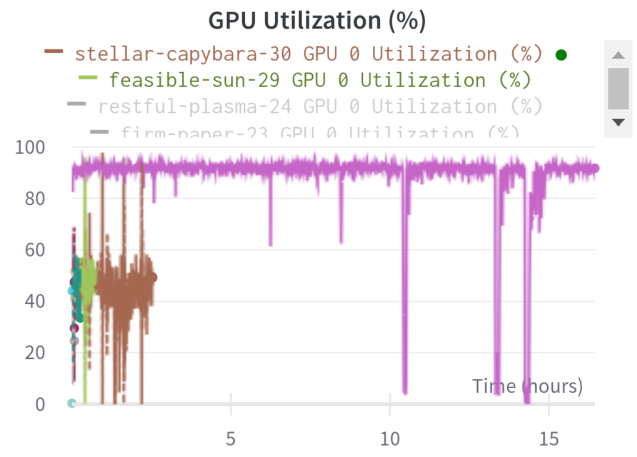


Figure 18: Alpaca Runs : System GPU Utilization