Uppsala University
Data Engineering I 1TD169 62015 VT2025

Data Engineering - I
Project Report

An Analysis with Reddit Comments Dataset

Group - 22

Eric Dario Márquez León

Battu Anudeep

Dona Harshani Kokila Wickramasinghe

Ramiza Maliha

Ville Holma

March 20, 2025

Table Of Content

Background

The dataset that is used throughout this project to implement a scalable data processing solution for a chosen analysis is the Webis TLDR Corpus (Völske et al. 59). The dataset consists of approximately 4 Million content-summary pairs extracted for Abstractive Summarization, from a Reddit dataset for the years 2006-2016. The task of automatic summarization is to effectively summarize shortened documents and texts to a coherent version that still includes and conveys the main points. These texts and articles that have been analyzed are often from newspapers and therefore properly written by professional journalists. Due to the subjects of these papers often being news, they follow a very similar format leading to a lack of diversity in the data used for training deep learning models that perform the automatic summarization tasks. Abstract summarization follows the same procedures as regular automatic summarization, but as it's used for social media posts from Reddit the text that is to be summarized utilizing deep learning solutions have to work with text that is written by different people, informal, unstructured, containing everyday topics and is often poorly written (Völske et al. 59-60).

This specific dataset has been used in two other studies apart from the one it was created for. Both of these two works are mostly written by the same authors as the original paper and acts as a continuation of the work that was started there. They are complimentary towards each other and furthers the work of abstract summarization by sharing the dataset for participants to take part of and generate a summary of a social media posting that is then to be analyzed and evaluated. Where one paper is used as an introductory to the challenge provided by the authors (Syed et al., **"Task proposal"**) and the other presents the results and evaluation of submissions (Syed et al. **"Towards summarization"**).

This project aims to utilize this dataset in order to analyze what topics are trending in subreddits that are tech related. This is done by counting the occurrences of predetermined relevant words.

Data Format

The dataset that is used for this project comes originally from a larger and more raw dataset that has been filtered down to be used for abstract summarization of the submissions (Völske et al. 61). The creators of the dataset did this, because in order to create a deep learning solution that summarizes text from social media posts, they needed to have a longer text and a summary that includes the main parts of the original text for the model to train on. This reduced the data of the larger Reddit dataset from 286 Million submissions and 1.6 Billion comments to 1.6 Million and 2.3 Million respectively (Völske et al. 61). The final dataset comes in as a JSON file, where each line is a JSON object that represents a submission or a comment from said Reddit dataset. Every JSON object contains the information of the author, which subreddit it was posted in, a title, the body which is the main text and a summary. These objects also include additional information of the already mentioned contents like the length and id for a more descriptive view.

Analysis of the data

First, the data is collected from an HDFS, the system the dataset has been uploaded to, and loaded as a JSON file to make it compatible with the python language. Then the data is preprocessed through filtering for a handful of tech-related subreddits, removing numbers, punctuation and symbols, as well as making the text lowercase. After the preprocessing the computational analysis was performed through counting the frequencies of a set of categories and accompanying words. The categories consisted of "AI/ML", "Programming", "Cloud Computing", "Cybersecurity", "Big Data & Analytics" and "Hardware". The whole set of words that were analyzed can be seen in Table 1. Thereafter it was possible to construct a histogram for each category that showcases the frequency of the words in each category. The whole procedure was timed in order to see how long it took to run the whole code, as well as the specific steps like preprocessing, plotting etc. The preliminary results of the analysis can be seen in Figure 1.

Table 1: Showcasing the categories and words that are analyzed in the dataset.

Categories	Relevant Words
AI/ML	"tensorflow", "pytorch", "scikit", "keras", "fastai", "huggingface", "gpt"
Programming	"python", "java", "javascript", "typescript", "c", "c++", "rust", "golang", "ruby"
Cloud Computing	"aws", "azure", "gcp", "kubernetes", "docker", "lambda", "serverless"
Cybersecurity	"cybersecurity", "hacking", "ransomware", "phishing", "encryption", "firewall", "hashing"
Big Data & Analytics	"hadoop", "spark", "kafka", "snowflake", "databricks", "powerbi", "tableau"
Hardware	"gpu", "cpu", "nvidia", "amd", "intel", "raspberry", "arduino", "iot", "vr", "ar"

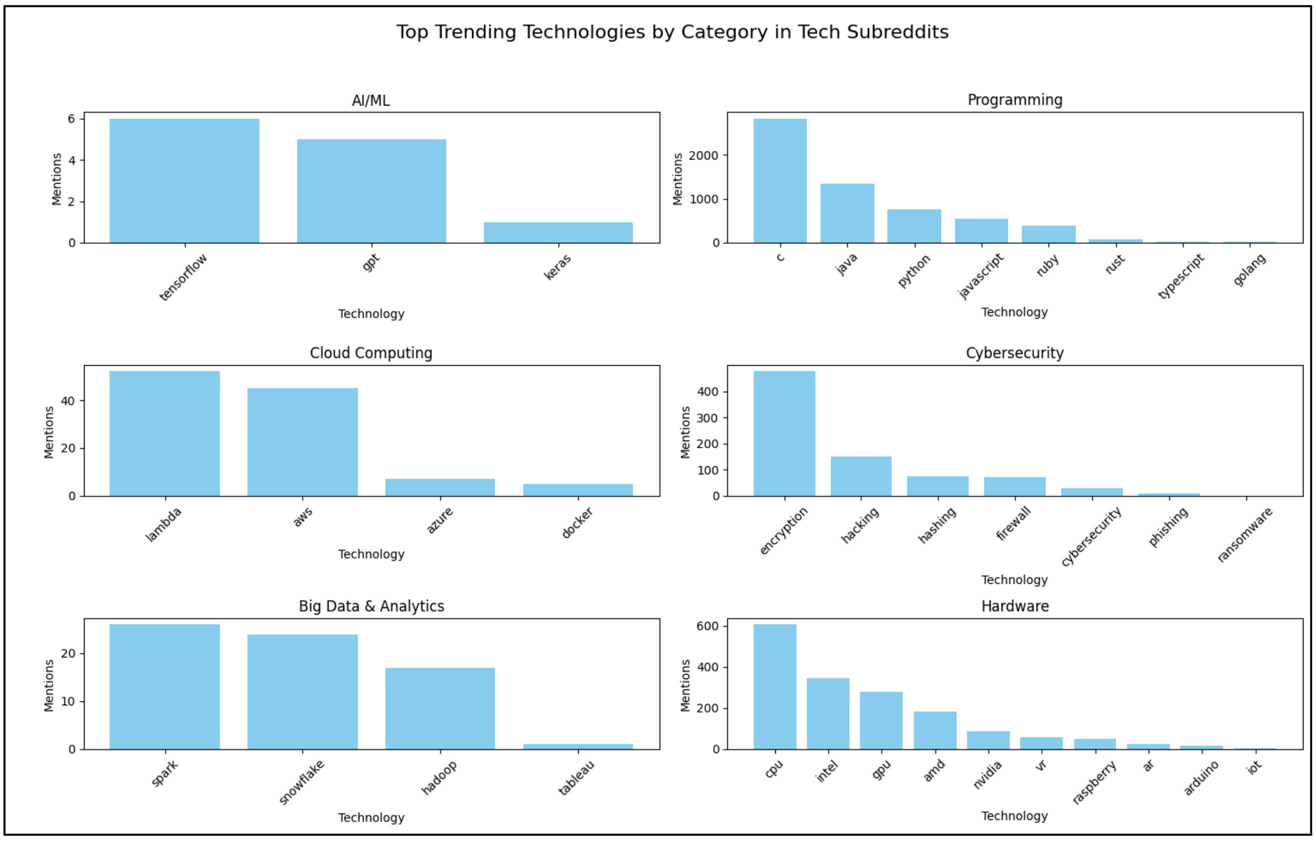


Figure 1: Preliminary results showing the trending topics by category.

Computational Experiments

The goal of this project is to perform an analysis on the chosen dataset and to observe how Spark automatically scales horizontally according to the operations executed on the dataset. Thereafter, the scalability approach is to be demonstrated through the computational experiments performed on the dataset. As mentioned before, the analysis that is to be performed using the dataset is to find trending topics of relevance to tech-related subreddits.

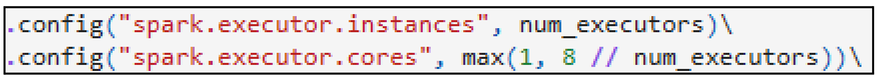
Scalability in a distributed system is the ability of how the platform is able to handle larger data without the risk of the performance being hindered. The aspect of horizontal scalability is when the system allocates or removes additional resources in the form of nodes or machines to perform the scalability task (Lee et al. 300). The problem analysis was achieved by utilizing ApacheSpark and its cluster mechanism.

Experiment 1 - Strong Scaling

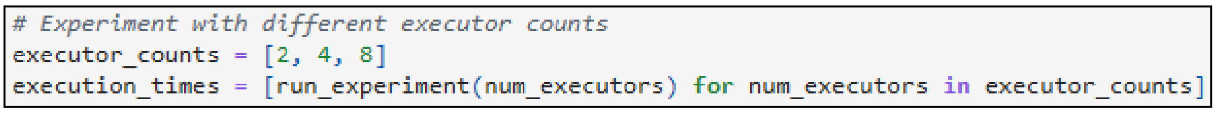
This experiment was conducted by manually increasing the number of nodes processing the data while allocating cores per node, with a maximum core limit of 8 due to cluster constraints.

To evaluate strong scaling, we specified a list of executors and ran our script for each configuration, measuring the total execution time. Figure 2 visualizes the change in execution time as the number of nodes increases. The execution times observed for 2, 4, and 8 executors were 264.59 seconds, 207.79 seconds, and 172.35 seconds, respectively.

Spark session configuration:



```
.config("spark.executor.instances", num_executors)\
.config("spark.executor.cores", max(1, 8 // num_executors))\
```



```
# Experiment with different executor counts
executor_counts = [2, 4, 8]
execution_times = [run_experiment(num_executors) for num_executors in executor_counts]
```

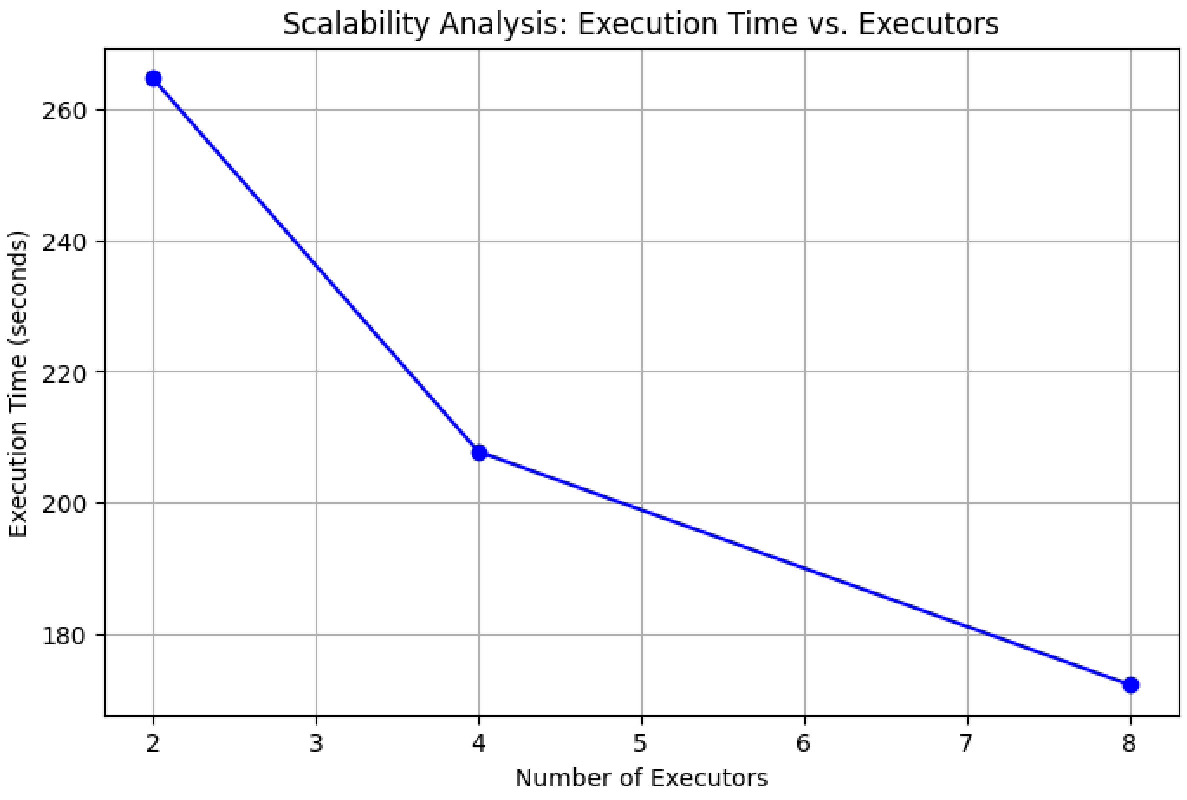


Figure 2: Showcasing the change in execution time w.r.t number of executors.

Experiment 2 - Data Scaling

Goal:

To evaluate horizontal scalability by running several analyses with gradually increasing sizes of dataset while maintaining the constraints given by the instructors, such as the maximum number of cores at 8 and the following Spark session configuration:

Table 2: Spark session configuration used for data scaling

Configuration	**Value**	**Description**
dynamicAllocation.enabled	True	Enables dynamic allocation of executors, allowing Spark to adjust resources based on workload demands.
dynamicAllocation.shuffleTracking. enabled	True	Allows Spark to track shuffle dependencies and remove inactive

		executors sooner, improving resource utilization.
shuffle.service.enabled	False	Since the external shuffle service is disabled, Spark will manage shuffle files itself. This is appropriate if executors are short-lived and storage overhead is not a concern.
dynamicAllocation.executorIdleTimeout	30s	If an executor is idle for 30 seconds, it will be removed to free up resources. This balances responsiveness with resource efficiency.
cores.max	8	Limits total CPU usage to 8 cores, ensuring the experiment does not exceed project constraints.
executor.cores	2	Each executor gets 2 cores, providing parallelism while preventing a single executor from monopolizing CPU resources.
executor.memory	3G	Each executor gets 3GB of memory, a balance between preventing memory spills and avoiding excessive allocation.
dynamicAllocation.initialExecutors	2	Starts with 2 executors to ensure a reasonable baseline for parallelism at launch.
dynamicAllocation.minExecutors	1	Allows scaling down to a single executor if the workload decreases, minimizing idle resource usage.
dynamicAllocation.maxExecutors	4	Limits the total number of executors to 4 to control resource consumption and prevent excessive scaling.

Implementation:

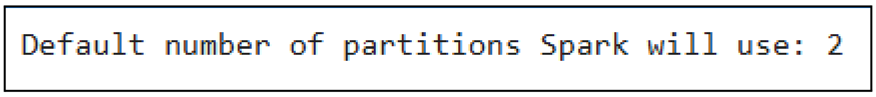
It is expected that the number of executors, cores, and memory usage will vary depending on the computational intensity of each stage of the job and to verify this, a function was created to monitor the resources of the cluster and to retrieve and print a list with the following information:

- Worker nodes
- Executors
- Executor cores
- Executor memory
- Total memory used
- Partition number used by the RDD or DataFrame

Such a function is called at each section of the code in which a transformation or an action is executed on a dataframe and therefore a change in the resources could be observed. Below are the results obtained.

1. Pre-loading the data

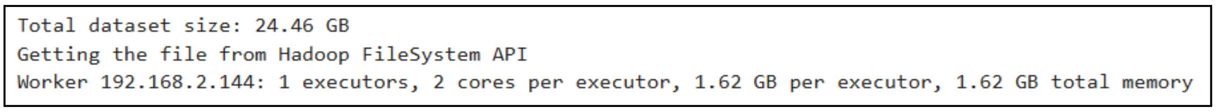
To check for the default parallelism, before loading the data the default number of partitions Spark will use is shown.



```
Default number of partitions Spark will use: 2
```

2. Getting the files from Hadoop FileSystem

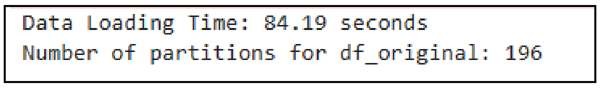
To verify the size of the original data and the resources utilized so far:



```
Total dataset size: 24.46 GB
Getting the file from Hadoop FileSystem API
Worker 192.168.2.144: 1 executors, 2 cores per executor, 1.62 GB per executor, 1.62 GB total memory
```

3. Creating the first (whole) dataframe

To verify the time taken for loading the whole data and the number of partitions used.



```
Data Loading Time: 84.19 seconds
Number of partitions for df_original: 196
```

The number 196 is because the Spark's default shuffle partitions value (`spark.sql.shuffle.partitions`) is 200, however the actual number can vary slightly based on the dataset.

4. Processing incremental fractions of the dataset

The analysis of the data was done incrementally in fractions of the whole dataset.

```
dataset_fractions = [0.1, 0.25, 0.5, 1.0]
```

As stated before, the monitoring of resources was done in those parts of the code in which a transformation or an action is executed on a dataframe, these corresponded to:

- Loading the fraction of the dataset
- Removing the stopwords
- Exploding words into individual rows
- Finding words matching any category
- Counting occurrences for each word

Table 3: Resource and partitions usage at each stage of the analysis

Step	**Fraction of Data**	**Workers (IP Addresses)**	**Executors p/ Worker**	**Cores per Executor**	**Memory per Executor**	**Total Memory**	**Partitions**
Fraction of the data	0.1 - 1	192.168.2.131, 192.168.2.134, 192.168.2.203, 192.168.2.144	1	2	1.62 GB	1.62 GB	196
Remove stopwords	-	Same as above	Same as above	Same as above	Same as above	Same as above	196
Explode words	-	Same as above	Same as above	Same as above	Same as above	Same as above	196
Find word categories	-	Same as above	Same as above	Same as above	Same as above	Same as above	196
Count word occurrences	-	Same as above	Same as above	Same as above	Same as above	Same as above	1

Finally a plot was created to observe the behavior of Execution Time vs Dataset size.

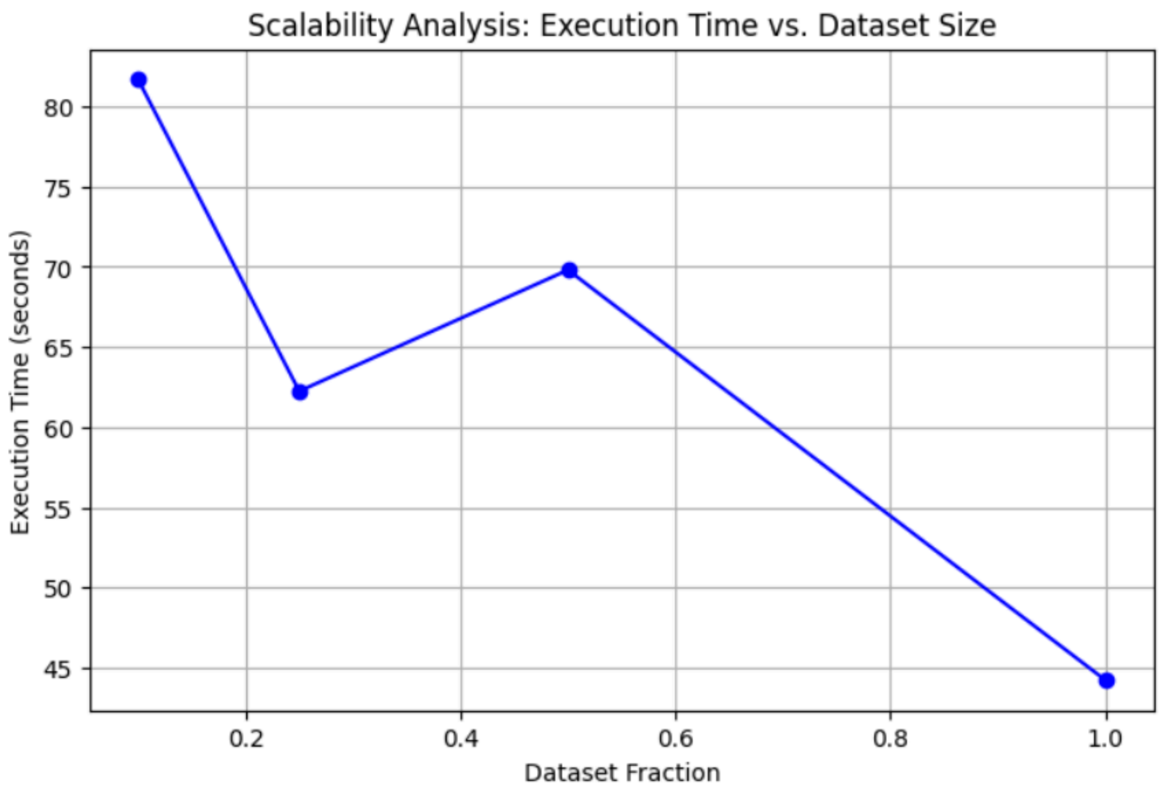


Figure 3: Showcasing the change in execution time w.r.t dataset size.

Discussion and Conclusion

Our experimental results show the effectiveness of horizontal scaling for processing large-scale text data from the Webis TLDR Corpus. The outcome of the strong scaling experiment shows a significant performance improvement when we increase executors from 2 to 8, with the execution time decreasing from 81.7 seconds to 69.81 seconds. This is almost a 14.55% reduction in processing time, although the efficiency gain diminishes with each additional executor this is in line with Amdahl's Law, that parallel speedup is limited by the sequential components of a work to be executed.

We tested dynamic allocation of executors so that the resources would scale up or down automatically based on the workload needs. This worked well for our text analysis, which had varying amounts of computational needs at different points in processing. To find a balance between performance and resources we used, we tuned parameters like the 30-second idle timeout, preventing the waste of resources during lighter processing phases.

The analysis of tech-related subreddits provided valuable insights into the popularity of different technology topics. From Figure 1, programming-related discussions had the highest frequency, with C, Java and python as the most commonly mentioned languages.

In contrast, AI/ML technologies had relatively few mentions during the 2006-2016 period, showing their lower popularity during the study period. Deep learning platforms like TensorFlow gained traction only at the end of this period, while topics which related to Big Data & Analytics and Cloud Computing had limited representation, indicating that these technologies were still emerging and had not yet reached mainstream adoption.

One of the challenges of our method is the predefined list of keywords that we used in our analysis. Technologies and terminology change rapidly, and some relevant discussions may have been missed due to alternative terminology or new topics or else because of new concepts which are not included in our keyword lists. However, our filtering for tech-related subreddits may have uncovered relevant discussions occurring in more general forums.

Overall, the project successfully demonstrated how Apache Spark can be utilized for scalable big data processing. From our study, we were able to confirm that increasing the number of executors improves execution time, but excessive scaling may not always result in significant improvements. Dynamic resource allocation effectively managed memory and CPU usage, providing efficient execution with varying dataset sizes. However, minor variations in execution time were observed, likely due to partitioning strategies and shuffle operations.

For future improvements, we plan to focus on optimizing the partitioning strategies to reduce shuffle overhead, implementing caching techniques to improve processing speed, and exploring GPU acceleration for further performance enhancements. Additionally, fine-tuning adaptive resource management could also help to improve efficiency in real-time applications. This study highlights the importance of scalable data engineering solutions and demonstrates how Apache Spark can effectively process large-scale unstructured text data from social media platforms.

See Code In GITHUB:

DE-1-Group22 (https://github.com/Ramizoro/DE-1-Group22.git)

References

Völske, Michael, et al. "Tl; dr: Mining reddit to learn automatic summarization." *Proceedings of the Workshop on New Frontiers in Summarization*. 2017.

Syed, Shahbaz, et al. "Task proposal: The tl; dr challenge." *Proceedings of the 11th International Conference on Natural Language Generation*. 2018

Syed, Shahbaz, et al. "Towards summarization for social media-results of the TL; DR challenge." *Proceedings of the 12th International Conference on Natural Language Generation*. 2019.

Lee, Jae Yoo, et al.. "Software approaches to assuring high scalability in cloud computing." *2010 IEEE 7th International Conference on E-Business Engineering*. IEEE, 2010.

Group Contribution

Project: An Analysis with Reddit Comments Dataset
Date: 20-03-2025
Group name: DE-Group 22
Group members: Eric Dario Márquez León, Dona Harshani Kokila Wickramasinghe, Ramiza Maliha, Battu Anudeep, Ville Holma

Contributions:

Eric Dario Márquez León: Adapted code for setting up a second experiment and to monitor additional cluster parameters. Wrote descriptions and scope. Participated in group discussion and ideation.

Dona Harshani Kokila Wickramasinghe: Wrote discussion and conclusion and participated in group discussion and ideation.

Ramiza Maliha: Did data analysis, worked on discussion and conclusion,edited and formatted report and participated in group discussion and ideation.

Battu Anudeep: Did preliminary analysis on the dataset. Wrote about the first computational experiment in the report. Participated in group discussion and ideation.

Ville Holma: Wrote background and data format. Did the two experiments on horizontal scalability. Participated in group discussion and ideation.

<u>Signature and Date:</u>
Ville Holma 19-03-2025
Battu Anudeep 20-03-2025
Kokila Wickramasinghe 20-03-2025
Ramiza Maliha 20-03-2025
Eric Dario Márquez León 20-03-2025