Proceedings of the Symposium

on

Recent Advances in Natural Language Processing

University of Colorado, Colorado Springs

August 2, 2024

Editors: Jugal Kalita and Armin Moin

Funded by

National Science Foundation

Preface

It is with great pleasure that we present to you the papers describing the research performed by the NSF-funded Research Experience for Undergraduates (REU) students, who spent 10 weeks during the summer of 2024 at the University of Colorado, Colorado Springs. Within a very short period, the students were able to choose cutting-edge projects involving machine learning in the areas of natural language processing, and applications of natural language processing in software engineering. Additionally, they also wrote proposals; designed interesting algorithms and approaches; developed code; performed analysis; and wrote scholarly papers describing their findings. We hope that the students will continue working on these projects and submit papers to conferences and journals within the next few months. We also hope that it is the beginning of a fruitful career in research and innovation for all our participants.

We thank the National Science Foundation for funding our REU site. We also thank the University of Colorado, Colorado Springs, for providing an intellectually stimulating environment for research. We thank Dr. Terrance Boult, who was a helpful mentor for the REU students. We also thank Sharon Huscher for working out all the financial and administrative details. We thank our students and recent graduates, in particular, Ali AlShami, Brendan Bena, Timothy Flink, Justin Leo, Melkamu Mersha, Ryan Rabinowitz, Eugene "Jeno" Saghi, and Joseph Worsham for helping the students with ideas and with presentations on some of the latest papers, systems, and programming issues. Our gratitude to Ginger Boult for being the "REU Mom" and having the welfare of the REU interns in her heart throughout the summer. Thanks to Ethan Capehart for exceptional help with the computing infrastructure, including the GPU machines. Special thanks to Harrison Gietz (UCCS REU 2024, BS University of Louisiana); Wesley Robbins (UCCS REU 2021; BS Montana State University; UCCS MS 2023; PhD student, University of Texas, Austin); and Abigail Swenor (UCCS BS 2022, UCCS REU 2021; PhD student, Notre Dame University); and Joshua Zingale (UCCS REU 2023, BS San Diego University; heading to University of California, Riverside for his PhD studies) for giving presentations and/or taking part in panel discussions on how to apply to graduate school.

Sincerely,

Jugal Kalita
jkalita@uccs.edu

Armin Moin
amoin@uccs.edu

August 2, 2024

Table of Contents

Sentiment Analysis with BERTopic of Large Open-Source Software Repositories for Bug Destiny Prediction

Sophie Pope
University of Wisconsin-La Crosse
poppysc@icloud.com

Armin Moin
University of Colorado Colorado Springs
amoin@uccs.edu

Abstract

This paper aims to find a novel way to predict bug time to resolution instead of time to fix with features reported before the bug is resolved with a dataset from a Bugzilla Eclipse Project. We conduct sentiment analysis to find an emotionality score and an emotion (positive or negative). We utilize these, the bug's priority, and the bug reports topic from a BERTopic model as features for a Convolutional Neural Network (CNN) and a Multilayer Perceptron (MLP). Using BERTopic with sentiment analysis provides a better prediction than without BERTopic. With sentiment analysis and BERTopic, our CNN model gets an F1 score of 0.65 and an accuracy of 0.74.

1 Introduction

Large open-source projects offer an issue tracking system or open bug repository, where developers and users can report the software defects they find or any new feature requests they may have. These reports are called *bug reports*. We focused on the natural language data available in descriptions of bug reports and the priority of bug reports. The priority of a bug report is the assigned importance of a bug from the developers. For example, the eclipse bug project has five levels, P1-P5. P1 indicated the bug was very severe, and P5 meant that the bug was not severe at all. An example of a P1 bug is if the software crashes and users cannot use it.

Moreover, developers often use revision control systems (version control systems or source code repositories), such as Git. They usually write a commit log or commit message to commit code to these repositories. These logs or messages also included valuable natural language data. In some cases, they also include the unique identifier (ID) of the bug report to which this specific change in the code base is related. For instance, if they managed to resolve an issue, they typically include the bug ID in their commit log.

We applied recent advances in Natural Language Processing (NLP) to analyze the *sentiments* of the above-mentioned natural language text (i.e., bug reports and commit logs) and explore possible correlations between the detected sentiments and the bug reports and how much time it took to fix the bugs, i.e., *time-to-fix*. We also looked at *time-to-resolution*. The difference between these is that *time-to-resolution* referred to the time until a conclusion of a bug is reached, which included labels such as *won't-fix*, *invalid*, *works-for-me*, *duplicate*, *Not_Eclipse*, and *nduplicate*. In contrast, *time-to-fix* only applies to bugs that have been fixed. We proposed that predicting *time-to-resolution* of a bug is more useful in practice because we do not know if a reported bug will be fixed. Therefore, to make our analysis more practical, we focused on *time-to-resolution* and did not exclude the bugs that are not fixed. We discuss this further in the 3 section.

Also explained in Section 3, prior works in the literature have applied Natural Language Processing (NLP) to software repositories. This often falls under the broad topic of Mining Software Repositories (MSR). For instance, AI-based approaches have addressed the bug triage problem. Examples include using Machine Learning (ML) (Anvik et al., 2006) (including graph learning (Jeong et al., 2009)) and Information Retrieval (IR) (Sun et al., 2011) to (semi-)automate bug triage. *Bug triage* is finding invalid and duplicate bug reports and assigning the valid and unique ones to the developers working on the project so they can fix and resolve them. In large open-source projects, a group of developers called *triagers* are typically responsible for the bug triage task.

By predicting the time-to-resolution for bug reports, we can enable triagers to assign bugs that are prone to be time-consuming to developers with a higher rate of availability or who are known to be committed to the project for a more extended pe-

riod. This can reduce the likelihood of *bug tossing* (i.e., reassignment of bug reports to other developers), which is costly for the project.

In this paper, we proposed a novel approach to automated bug destiny prediction using sentiment analysis and BERTopic from natural language text available in software bug and code repositories. We aimed to predict how long developers would take to resolve a bug. Sentiment analysis has already been applied to bug repositories. For instance, Umer et al. (Umer et al., 2018) analyzed *emotions* in bug reports to predict the priority level of the report. We also analyze the emotional sentiments in the descriptions in bug reports. However, we do this to predict the bug report's time-to-resolution (we call them the bug *destiny*).

This paper's contribution is twofold: First, it proposed and implemented a novel approach to automated bug report destiny prediction. This approach is validated by our experimental results using available data from open reference datasets deployed in related work in the literature. Second, it provides an open-source prototype that enables other open-source projects using similar issue tracking and revision control systems to use the proposed approach, thus benefiting from effective and efficient automated bug report destiny prediction.

This paper is structured as follows: Section 2 provides some background information about open bug repositories and NLP. Further, we review the literature in Section 3. In Section 4, we propose our novel approach and report on our experimental results in Section 5. Moreover, Section 6 discusses the results and points out potential threats to validity. Finally, we conclude and suggest future work in Section 7.

2 Background

2.1 Open Bug Repositories

These repositories are open to users, and they contain reports about software issues or enhancements and track the progress (Anvik et al., 2005). These bugs can include wanted enhancements or issues with the software. We use these open bug repositories to analyze bug reports' destinies, including eventual status and time to resolution. As seen in Figure 1, the destiny of bugs will be entered where the status is and will change from new to fixed, *won't-fix*, *invalid*, or *works-for-me*. Predicting the eventual status of a bug to help semi-automate the triage of bugs will save developers time and money because they will not be tasked to fix a bug that is predicted not to be fixable. Examples of bugs without fixed outcomes are duplicate bugs or user errors.

Figure 1: Sample Bug Report

2.2 Natural Language Processing

Software development projects often include open bug repositories. Natural Language Processing (NLP) is a part of computer science that works to help computers better understand human written or spoken language(Patten and Jacobs, 1994). We used natural language processing techniques to analyze the bug reports. Before we inputted the bug reports into a neural network, we needed to preprocess the reports. As shown in Figure 2, before we can analyze the text, we have to process it. This included tokenization, fixing spelling, removing stop words, and stemming. We used the package NLTK to do this (Loper and Bird, 2002). In our tokenization step, we separated the words in the reports to make each word a token and lowercase. We removed stop words, which are words that are often used but do not add value to sentences, such as "the," "I," or "on." This allowed us only to analyze the words that were added to the report. Stemming of the tokens involves changing words to the core part to include more words from a sentiment analysis dictionary. An example of stemming is changing "advancement" to "advance" (Singh and Gupta, 2017). After preprocessing bug reports, we used sentiment analysis and the SentiWordNet lexicon (Baccianella et al., 2010). This dictionary assigned the words it had matches for a positive, negative, and objective (neutral) score.

2.3 BERTopic

BERTopic (Grootendorst, 2022) is an advanced topic modeling technique that uses Bidirectional Encoder Representations from Transformers (BERT) to generate topics from textual data

and bug report descriptions. First, we had to pretrain it on a training set so that it could create embeddings that attempt to capture the meaning of the text. It then groups bugs that have similar embeddings and generates topics.

3 Related Work

3.1 Sentiment Analysis in Bug Reports

Ramay et al. (Ramay et al., 2019) used sentiment analysis to predict whether a bug was severe or not severe by using the Senti4SD lexicon and a Convolutional Neural Network (CNN). Found that neural networks with sentiment analysis outperformed the leading machine learning algorithms Multinomial Naïve Bayes (MNB) and Random Forest (RF).

Baarah et al. 2021 (Baarah et al., 2021) used sentiment analysis and found that Multilayer Perceptron (MLP) predicts severity better than the lexicon-based neural network. F-Score of 0.81 compared to 0.52. For their sentiment analysis, they took the difference between a report's positive and negative score of a report. If the difference was positive, it was labeled a positive report. If the difference is negative, they labeled it a negative report before testing the neural network's success in predicting severity.

Umer et al. (Umer et al., 2020) used a Convolutional Neural Network (CNN) to prioritize bugs after calculating the sum of positive and negative emotion scores to get a total emotionality score. They inputted that sum and the feature vectors of each bug report into a CNN and got an F-Score of 0.6364.

We used these three sources as ideas for conducting sentiment analysis on bug reports because they had been proven successful in predicting other aspects of bug reports, such as priority and severity. We took inspiration to use a CNN model and an MLP model (Umer et al., 2020) (Baarah et al., 2021).

3.2 Time-to-Fix

Because there are so many ways to categorize time to fix as either numerical or categorical, comparing metrics across different prediction techniques was very hard. Another challenge was that different platforms for bug reports included unique information about each report. In this section, we will first discuss the papers with the highest F1 scores when predicting the time to fix, and then we will discuss the papers that work with similar datasets and use similar metrics for short and long.

Habayeb et al. (Habayeb et al., 2018) used a Hidden Markov Model (HMM) and found promising results when looking at a Firefox project and used the temporal dataset of activities that occurred during the life cycle of a bug.

Ardimimento in 2024 (Ardimento, 2024) had successful results using Long Short Term Memory (LSTM) to predict the time to fix. They only looked at bugs that had been fixed. The dataset they used was an Eclipse project, and both examined the chronological order in which bug reports are fixed using temporal sequences. However, our dataset does not include the activities or steps to fix a bug. So, our project is more inclusive of those bug reports that still need to be completed and can better be applied to bug reports as they come in.

Ardimento in 2020 (Ardimento and Mele, 2020) successfully used Bert to predict bug fixing time. This model also only looked at bug reports that were eventually labeled fixed. They labeled the longest 50% of bugs as long and the rest as short.

Marks et al. (Marks et al., 2011) looked at an Eclipse project similar to ours and used information about the bug report that can be found before it gets fixed, such as reporter, location (component), and description. They used a random forest classifier and got an accuracy of 0.65.

Ardimento in 2017 (Ardimento and Dinapoli, 2017) used knowledge extraction and looked at data sets from LiveCode, Novell, and OpenOffice to predict the time to fix. They split up the data 75% of bugs labeled as fast and 25% labeled as slow. This was similar to splitting our data into slow and fast categories. They used features that can be created or found before the bug is fixed, such as description, comments from the developer, priority, severity, and more. They got the best results using the LiveCode dataset with an accuracy of 0.52, a recall rate of 0.62, and a precision of 0.22. Even though this is a different dataset, we will compare our results with these results because they use features provided early in the triaging process.

Panjer (Panjer, 2007) examined the time-to-resolve all bugs instead of the time-to-fix. They used data mining techniques to get an accuracy of 0.34.

These two sources had similar information in their data sets as we had in ours, so we found com-

paring our work to theirs to be the closest comparison.

3.3 Time-to-Resolution

This paper focused on correctly using the term time-to-fix for bug reports and how that is compared to time-to-resolution. Time-to-fix is strictly for bugs that have been fixed and predicting how long those will take to fix. Time-to-resolution is more practical and should be labeled as such. Papers often (Ardimento and Mele, 2020) (Ardimento and Dinapoli, 2017) use them interchangeably and talk about how they use Bert to predict bug resolution time, but in their proposed approach, they sort out only the bugs that say fixed as a resolution. We suggest a more holistic approach when referring to either or and being consistent across papers.

4 Proposed Approach

We used an Eclipse Bugzilla dataset (LogPAI, 2013) with 85156 bugs with no developer comments and only descriptions from a reporter. Because our dataset did not include time to resolution, the first thing we did was add duration labels. To create a column for it, we subtracted the resolution time from the day reported. To predict time labels, we created a binary classification: long and short. From there, we made a TimeLabel column and labeled every bug report whose time was in the lowest 70% Before we performed sentiment

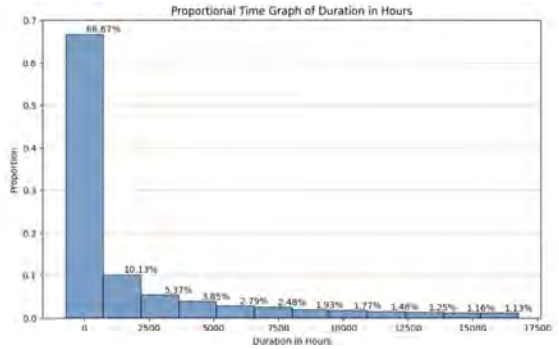


Figure 2: Distribution of Time to Resolution

analysis, we needed to preprocess the report's description, as shown in Figure 2. This includes tokenization, removing stop words, and stemming. We used the package NLTK to do this (Loper and Bird, 2002). Similar to Umer et al. (Umer et al., 2020), we analyzed the sentiments and emotions in the bug report's descriptions.

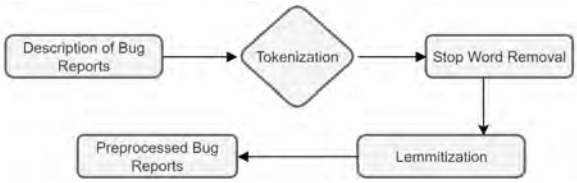


Figure 3: Preprocessing of bug reports

After preprocessing bug reports, we used sentiment analysis and the SentiWordNet lexicon (Baccianella et al., 2010). As far as we know, previous approaches do not look to predict a bug report's time-to-resolution or use sentiment analysis to predict the time to fix (we call them the bug *destiny*) by using sentiment analysis. Other bug report triaging techniques that used sentiment analyses to predict priority or severity included emotionality as the sum of the assigned weights for positive and negative words or the difference between positive and negative to categorically label them (positive and negative) before using a machine learning technique. We calculated these separately so bug reports are labeled positive or negative and the sum so we could see the proportion of negativity in sentiment and positivity. This aimed to help the neural network create a better prediction model.

After conducting the sentiment analysis, we used a BERTopic model to predict topics for each description. We trained the model on a training set from the Eclipse bug repository to create 20 topics. My model then uses this finetuned BERTopic model (Grootendorst, 2022) to assign topics to each bug report. Perceptron Model (MLP).

Unlike the time-to-fix prediction papers, we did not sort out the bug reports labeled fixed in the resolution column. After completing all of these different features, we used them through two different neural networks: Convolution Neural Network (CNN) and MultiLayer Perceptron Model (MLP). We compared how different models did with features such as unique Emotion, Emotionality, Priority, and Predicted topic combinations. We measured these different models by using weighted recall scores, precision scores, F1 scores, and accuracy. We used weighted scores for each metric to account for differences in class distribution. A high recall score means that the model correctly identifies many positive instances.

$$\text{Recall}_i = \frac{\text{True Positives (TP)}_i}{\text{True Positives (TP)}_i + \text{False Negatives (FN)}_i}$$

A high recall measure implies that the model cor-

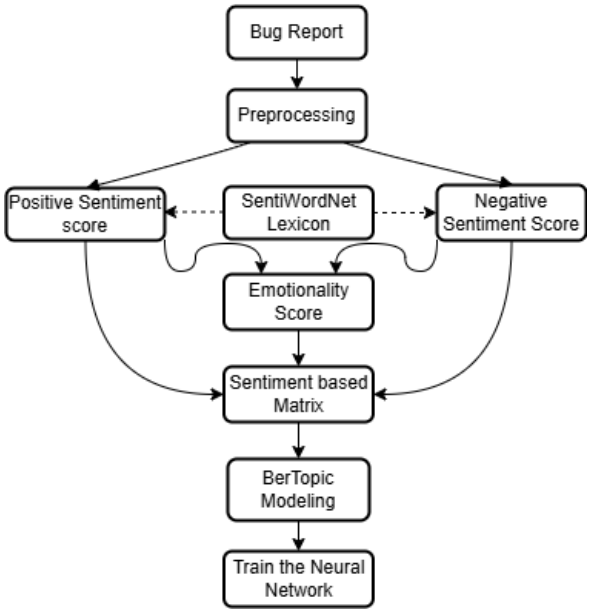


Figure 4: High Level Pipeline

rectly identifies the most positive instances.

$$\text{Precision}_i = \frac{\text{True Positives (TP)}_i}{\text{True Positives (TP)}_i + \text{False Positives (FP)}_i}$$

A high precision measure implies that if the model guesses a positive instance, it will likely be one.

$$\text{F1-Score}_i = 2 \cdot \frac{\text{Precision}_i \cdot \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i}$$

The F1 score is a combined metric of precision and recall.

$$\text{Weighted Recall} = \sum_{i=1}^{N} \frac{T_i}{T} \cdot \text{Recall}_i$$

$$\text{Weighted Precision} = \sum_{i=1}^{N} \frac{T_i}{T} \cdot \text{Precision}_i$$

$$\text{Weighted F1-Score} = \sum_{i=1}^{N} \frac{T_i}{T} \cdot \text{F1-Score}_i$$

Where T_i is the number of actual instances in class i, and T is the total number of cases across all classes.

Accuracy is the number of bug reports the model labels correctly as short or long.

5 Experimental Results

5.1 Predicting Time Label

After the preprocessing stages and creating the topics, we used an MLP model to predict the Time to Resolution. We did this by training a model on Emotion, Emotionality, and Priority (developer-assigned level of importance of a bug report). We repeated this using a CNN model and four Epochs to prevent overfitting of our model. We used these two as baselines to find if using BERTopic to predict topics for bugs helps with sentiment analysis of bug reports when predicting time to resolution. Then, we repeated both steps except for adding a feature for the predicted topic of the bug reports. This succeeded in both the MLP and the CNN models by improving the F1 score, recall, precision, and accuracy. All of these scores can be seen in Table 1. This implies that using a BERTopic with sentiment analysis can successfully predict the time to resolution with an accuracy of 0.75. We also looked into how our model would compare to the related work by sorting out only the bug reports with fixed resolution to find the predictions for time to fix instead of time to resolution. Our model worked better on time to fix than time to resolution, but only by 0.01. Our model succeeded when being more real-world applicable for triaging bugs and can compare to many related works for time to fix.

Table 1: Comparison between our Proposed Models

Model	F1 Score	Recall	Precision	Accuracy
MLP (Emotion, Emotionality, Priority)	.53	.51	.577	.51
CNN (Emotion, Emotionality, Priority)	.58	.70	.49	.70
MLP (Emotion, Emotionality, Priority, Predicted Topic)	.66	.73	.69	.73
CNN (Emotion, Emotionality, Priority, Predicted Topic)	.65	.74	.75	.74
CNN (Emotion, Emotionality, Priority, Predicted Topic) for time to Fix	.66	.75	.75	.75

We also ran experiments using a Support Vector Machine as a regression model to find if there was

a relationship between emotionality and time-to-resolution. We created a scatter plot of this data to visualize it before running the machine learning model, as seen in Figure 5. We found a R^2=-0.23.

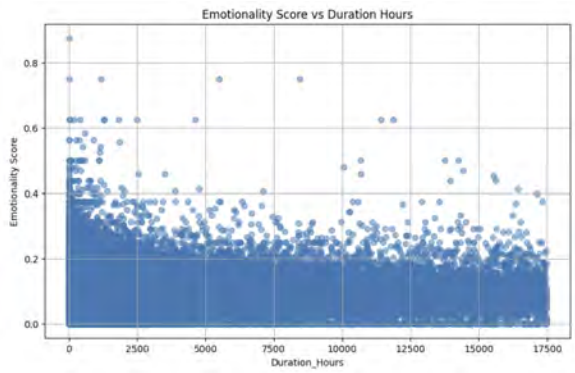


Figure 5: Scatter plot of Emotionality Score's Relationship with Time to Resolution

6 Discussion

Table 2 shows how our approach compares to previous methods that used information after the bug was resolved. The three other approaches in this table looked at how the variable length of steps taken to fix a bug impacts the time to fix, and this is not information included in our dataset of eclipse bugs, which means that we had to come up with a new way to predict them without that information. We also looked at time-to-resolution compared to time-to-fix, which is more applicable for triaging bugs because we can not assume that a bug will be fixed and how it was fixed. To the best of our knowledge, table 3 shows how our work compares to the state-of-the-art approaches for predicting bug-fixing time using information from before the bug was fixed. All of these approaches except Panjer (Panjer, 2007) look specifically at time-to-fix instead of resolution. Because our model works even better for time-to-fix, we achieved state-of-the-art results using BERTopic and sentiment analysis.

Table 2: Comparison of Model Performance Metrics

Model	F1 Score	Recall	Precision	Accuracy
HMM (Ardimento, 2024)	0.70	0.69	0.69	0.71
DEEPLSTM (Sepahvand et al., 2020)	0.96	0.87	0.79	0.88
LSTM (Ardimento, 2024)	0.97	0.88	0.79	0.87
CNN (Emotion, Emotionality, Priority, Predicted Topic)	0.65	0.74	0.75	0.74

Table 3: Comparison of Model Performance Metrics Using Features From Before a Bug is Resolved

Model	F1 Score	Recall	Precision	Accuracy
Data Mining (Panjer, 2007)	–	–	–	0.34
Random Forest(Marks et al., 2011)	–	–	–	0.65
Knowledge Extraction (Ardimento and Dinapoli, 2017)	–	0.62	.22	0.52
CNN (Emotion, Emotionality, Priority, Predicted Topic)	0.65	0.74	0.75	0.74

6.1 Threats to Validity

It is hard to compare our results to other approaches because we used a different dataset that did not include comments from developers. It may be formatted differently and include more or less information than our dataset. Because time-to-fix does not have set labels, we put them in binary classes, so knowing the exact time to fix them, past short or long, is unknown. Also, the metrics for labeling bug reports as long or short are different from other work done on bug fixing time. An assumption can also be made that if our model achieves similar results when examining time to fix compared to time to resolution, other approaches may, too.

7 Conclusion and Future Work

We have found a relationship between how emotional a description report is and how long it takes. Further work could include exploring if using emotionality as an added feature in other models can improve them, such as we did with

BERTopic, and continuing to explore how sentiment analysis can help with time prediction. Another example of possible future work is looking at whether emotionality's effect on time to resolution depends on whether a user or a developer reported the bug. Overall, we successfully predicted bug reports time-to-resolution using sentiment analysis and BERTopic.

Acknowledgment

All work herein reported is supported by the Nation Science Foundation under Grant No. 2349452.

Any opinion, finding, or conclusion in this study is that of the authors and does not necessarily reflect the views of the National Science Foundation.

References

John Anvik, Lyndon Hiew, and Gail C. Murphy. 2005. Coping with an open bug repository. In *Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange*, eclipse '05, pages 35–39, New York, NY, USA. Association for Computing Machinery.

John Anvik, Lyndon Hiew, and Gail C. Murphy. 2006. Who should fix this bug? In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, page 361–370, New York, NY, USA. Association for Computing Machinery.

Pasquale Ardimento. 2024. Enhancing Bug-Fixing Time Prediction with LSTM-Based Approach. In *Product-Focused Software Process Improvement*, pages 68–79, Cham. Springer Nature Switzerland.

Pasquale Ardimento and Andrea Dinapoli. 2017. Knowledge extraction from on-line open source bug tracking systems to predict bug-fixing time. In *Proceedings of the 7th International Conference on Web Intelligence, Mining and Semantics*, pages 1–9, Amantea Italy. ACM.

Pasquale Ardimento and Costantino Mele. 2020. Using BERT to Predict Bug-Fixing Time. In *2020 IEEE Conference on Evolving and Adaptive Intelligent Systems (EAIS)*, pages 1–7. ISSN: 2473-4691.

Aladdin Baarah, Ahmad Aloqaily, Hala Zyod, and Nasser Mustafa. 2021. Sentiment-Based Neural Network Approach for Predicting the Severity of Bug Reports. In *2021 Fifth International Conference On Intelligent Computing in Data Sciences (ICDS)*, pages 1–8, Fez, Morocco. IEEE.

Stefano Baccianella, Andrea Esuli, and Fabrizio Sebastiani. 2010. SentiWordNet 3.0: An Enhanced Lexical Resource for Sentiment Analysis and Opinion Mining. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta. European Language Resources Association (ELRA).

Maarten Grootendorst. 2022. BERTopic: Neural topic modeling with a class-based TF-IDF procedure. Version Number: 1.

Mayy Habayeb, Syed Shariyar Murtaza, Andriy Miranskyy, and Ayse Basar Bener. 2018. On the Use of Hidden Markov Model to Predict the Time to Fix Bugs. *IEEE Transactions on Software Engineering*, 44(12):1224–1244.

Gaeul Jeong, Sunghun Kim, and Thomas Zimmermann. 2009. Improving bug triage with bug tossing graphs. In *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC/FSE '09, page 111–120, New York, NY, USA. Association for Computing Machinery.

LogPAI. 2013. Bughub: Eclipseplatform. `https://github.com/logpai/bughub/tree/master/EclipsePlatform`. Data set.

Edward Loper and Steven Bird. 2002. NLTK: the Natural Language Toolkit. In *Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics - Volume 1*, ETMTNLP '02, pages 63–70, USA. Association for Computational Linguistics.

Lionel Marks, Ying Zou, and Ahmed E. Hassan. 2011. Studying the fix-time for bugs in large open source projects. In *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, pages 1–8, Banff Alberta Canada. ACM.

Lucas D. Panjer. 2007. Predicting Eclipse Bug Lifetimes. In *Fourth International Workshop on Mining Software Repositories (MSR'07:ICSE Workshops 2007)*, pages 29–29. ISSN: 2160-1860.

T. Patten and P. Jacobs. 1994. Natural-language processing. *IEEE Expert*, 9(1):35–. Conference Name: IEEE Expert.

Waheed Yousuf Ramay, Qasim Umer, Xu Cheng Yin, Chao Zhu, and Inam Illahi. 2019. Deep Neural Network-Based Severity Prediction of Bug Reports. *IEEE Access*, 7:46846–46857.

Reza Sepahvand, Reza Akbari, and Sattar Hashemi. 2020. Predicting the bug fixing time using word embedding and deep long short term memories. *IET Software*, 14(3):203–212. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1049/iet-sen.2019.0260.

Jasmeet Singh and Vishal Gupta. 2017. Text Stemming: Approaches, Applications, and Challenges. *ACM Computing Surveys*, 49(3):1–46.

Chengnian Sun, David Lo, Siau-Cheng Khoo, and Jing Jiang. 2011. Towards more accurate retrieval of duplicate bug reports. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, pages 253–262.

Qasim Umer, Hui Liu, and Inam Illahi. 2020. CNN-Based Automatic Prioritization of Bug Reports. *IEEE Transactions on Reliability*, 69(4):1341–1354. Conference Name: IEEE Transactions on Reliability.

Qasim Umer, Hui Liu, and Yasir Sultan. 2018. Emotion based automated priority prediction for bug reports. *IEEE Access*, PP:1–1.

Automated Bug Report Prioritization in Large Open-Source Projects

Riley Pierson
Siena College
NY, USA
ra26pier@siena.edu

Armin Moin
Department of Computer Science
University of Colorado Colorado Springs
CO, USA
amoin@uccs.edu

Abstract

There are multiple automated approaches to bug triage, though many have proven less effective than the manual alternative. These approaches are convenient for detecting duplicate bugs and assigning bugs to various developers depending on their general problem area. However, automated approaches tend to lack the ability to categorize bug priority from bug reports accurately. In this study, we approach the prioritization dilemma through a novel approach to classify bugs into appropriate categories using a multi-faceted technique using recent advancements in *Natural Language Processing* (NLP). To this aim, we analyze the natural language used within bug reports, using topic modeling and text classification to prioritize bug reports properly.

1 Introduction

Large open-source projects offer an issue tracking system or open bug repository, where developers and users can report the software defects they find or any new feature requests they may have. These reports are called *bug reports*. However, the projects' resources are limited, while processing and resolving the bug reports is typically very costly. Hence, not all bug reports in the open bug repository can be processed and handled at once. Since some of them have a higher priority for the project, developers should resolve these bugs first.

Bug triage is a crucial process that excludes invalid and duplicate bug reports and assigns valid and unique ones to the developers working on the project so they can fix and resolve pertinent issues. Developers should give bug reports assigned a higher priority level priority. In fact, part of the bug triage process is the task of priority assessment for each bug report based on the project's specific requirements and resources.

In large open-source projects, a group of developers called *triagers* are often responsible for the bug triage task. Over the past decades, (semi-)automated approaches based on various Artificial Intelligence (AI) methods and techniques have been proposed to make the bug triage process more effective and efficient. This way, the total project costs will be reduced under automated approaches since the triagers' time will be saved and used more reasonably. Moreover, a more effective and efficient priority assessment for bug reports will help developers always put their time and energy into the most vital issues of the project.

In general, it is essential to distinguish between *severity* and *priority*, although in some systems, such as Android, they are used interchangeably (Umer et al., 2018). Bug reporters (e.g., a user) usually assign severity and describe the bug's impact on them. In contrast, priority is assigned by the developers (e.g., bug triagers) and characterizes the importance they place on fixing the bug. In Bugzilla, a popular issue tracking system, P1 is the highest priority, whereas P5 is the lowest (Eclipse Foundation AISBL).

As explained in Section 3, prior works in the literature have studied the bug triage problem and proposed various AI-based approaches to address it, such as approaches based on Machine Learning (ML) (Anvik et al., 2006) (including graph learning (Jeong et al., 2009)) and Information Retrieval (IR) (Sun et al., 2011). Most of them have used the natural language textual data in the bug reports as a crucial source of information. Hence, NLP plays a vital role. Automated bug triage can be considered as a sub-field of *Mining Software Repositories* (MSR).

This paper delivers a novel approach to automated priority assessment of bug reports in open bug repositories. First, we create numerous topic models and assign each report to one of them since our approach is based on NLP. This step is based on the work of Xia et al. (Xia et al., 2017). Second, we train a text classifier for each topic, which

can predict the priority level of a new bug report that belongs to that topic. For the latter, we deploy three different text classification tools to assess which achieved the greatest performance, Gaussian Naive Bayes, Multinomial Naive Bayes, and, lastly, BERT (Bidirectional Encoder Representations from Transformers) which is based on the work of Ali et al. (Ali et al., 2024).

This paper's contributions are twofold: First, we propose and implement a novel approach to automated bug report priority assessment, validated by our experimental results using available data from open reference datasets. Second, we provide an open-source prototype that enables other open-source projects using similar issue-tracking systems to use the proposed approach, thus benefiting from effective and efficient automated priority assessment.

This paper is structured as follows: Section 2 provides some background information about open bug repositories and NLP. Further, we review the literature in Section 3. In Section 4, we propose our novel approach and report on our experimental results in Section 5. Moreover, Section 6 discusses the results and points out potential threats to validity. Finally, we conclude and suggest future work in Section 7.

2 Background

2.1 Open Bug Repositories

Open bug repositories, such as Bugzilla, have advantages and disadvantages because they are publicly available. Since Bugzilla is open-source, it is easy to report problems, have problems fixed due to the number of developers available, find potential areas for enhancement, and engage in focused conversations regarding archived issues about the system of interest (Anvik et al., 2005). There are pitfalls to open bug repositories concerning bug duplication or proper assignment of bugs to developers. However, in many cases, the potential benefits can outweigh the problematic aspects of the repositories.

Bug reports are prioritized, specifically within Bugzilla, on a P1 through P5 scale. Developers consider P1 the most detrimental and impactful, while P5 is the most manageable bug within a specific product and component. Along with the prioritization level, each bug report includes some essential elements, including the bug ID, bug title, status, product, component, assignee, and problem description.

As far as textual information within a bug report is concerned, the bug title and description are both textually free-form, whereas the product and component are textually pre-defined fields within the report. For example, Figure 1 represents a relatively standard Bugzilla bug report.


Bug 232502 - TVT34:TCT307: PLK: truncation in "New static web project" wizard

Status: NEW

Alias: None

Product: WTP Java EE Tools
Component: wst.web (show other bugs)
Version: 3.0
Hardware: PC Linux-GTK

Importance: P2 major (vote)
Target Milestone: Future
Assignee: Roberto Sanchez Herrera

CDE Administration 2008-05-16 10:18:50 EDT Description

<response_by> Moravia IT at 2008.05.15.13.19.18 </response_by>
OS: Linux, Windows
Build date: 0514
Component/Function name: WTP
Blocking: NO
Tester Name: Kamil Ignacak

Steps to recreate the problem:

1. in main menu go to File -> New -> Project
2. Select ?Static Web Project? under the Web Folder. Click ?Next?.

Problem description:
Content of last frame in wizard window is truncated - there are three lines of text, last one is hidden. Please note that the frame does not protect its content from being hidden.

Figure 1: One shortened example of a Bugzilla bug report, Bug 232502 (Administration, 2012)

As demonstrated in Figure 1, the bug-id is a unique identifier for this issue, and the reporter generated the title. The status indicates whether the problem has been solved, assigned, or another one of several categories. The product and component indicate the target of the report, the software, and the specific area within the software that is of interest. Bugzilla requires some basic information about the problem to address the concern adequately.

Product and component are two areas of interest, as they may hold greater significance than previously thought. The product is the general location of the bug, while the component is the element within the product that is leading to precise issues. Many even compare the product's component with the bug as the root cause of the bug since bugs are more likely to originate within newer components rather than stabilized ones (Zhang and Challis, 2019).

2.2 Natural Language Processing

Natural language processing evolved from natural language understanding. NLP is an amalgamation of computational procedures, including syntactical, discourse, and semantic analysis, among many other processes, which, in turn, lead to the

examination of language by machines (Chowdhary, 2020). While it is possible to obtain analysis from machines, the main downfall is that a machine still underachieves compared to the level of analysis of humans at this stage (Chowdhary, 2020).

When preprocessing, NLP can tremendously reduce the noise within a dataset, making it much simpler for a machine to process efficiently (Ali et al., 2024). Some standard preprocessing techniques with natural language processing include word tokenization, stop-word removal, and lemmatization (Tian et al., 2015; Umer et al., 2018; Xia et al., 2017). Word tokenization extracts words as tokens from a text with a delimiter for separation between tokens (Tian et al., 2015). Stop-word removal ensures that only the more important words will be incorporated into the set of tokens, and lemmatization processes these relevant tokens into their baseline form (Umer et al., 2018). Our research mainly focuses on work tokenization and vectorization, where words are transformed into vector representations.

3 Related Works

3.1 Topic Modeling

Blei et al. created *Latent Dirichlet Allocation* (LDA), a three-tiered form Bayesian model used as a generative probabilistic model for a specific set of vocabulary (Blei et al., 2003). It is still used in many forms of research today; however, most use it as the baseline comparison to new state-of-the-art topic modeling approaches.

Xia et al. improved bug triage techniques using a topic model referred to as the *multi-feature topic model* (MTM) within their TopicMiner program (Xia et al., 2017). MTM has a more complex structure incorporating product and component elements of a standard bug report within its program. Another aspect of MTM that likely aided in impressive results within the program was its topic distribution for each bug report.

Xia et al., within their TopicMiner approach, tested the effectiveness of MTM over LDA by directly comparing results in their research of TopicMiner^{MTM} with that of TopicMiner^{LDA} (Xia et al., 2017). While MTM has similar features to the standard LDA, the TopicMiner^{MTM} program achieved unparalleled results.

Alternatively to MTM or LDA, there are *Neural Topic Models* (NTMs) as presented by Wu et al. (Wu et al., 2024). This approach is less conventional than the previous two discussed; NTMs are more scalable and flexible due to their parameters not requiring model-specific origins (Wu et al., 2024). However, several critical problems prevent NTMs from surpassing the successes found with MTM or LDA, including a lack of reliability and standard evaluation techniques, and the topics are usually of a low caliber (Wu et al., 2024).

3.2 Prioritization

Ali et al. predicted, using Bidirectional Encoder Representations from Transformers (BERT), the severity of bug reports using deep learning rather than machine learning (Ali et al., 2024). They collected publicly available datasets, computed the sentiment of each report within their data, and incorporated BERT for data formatting and creating prioritization predictions. Their results surpassed other well-known programs by a significant amount, partially due to their utilization of BERT. BERT is advantageous as it responds better to high-paced development and has a higher natural language understanding, a characteristic essential for text classification (Ali et al., 2024).

DRONE is an automated priority assessment approach presented by Tian et al. to base predictions on multi-faceted factor analysis (Tian et al., 2015, 2013). They also created a text classification engine that enhances linear regression with their original approach. Their findings improved effectiveness significantly from previous approaches (Tian et al., 2015, 2013).

Umer et al. investigated the use of emotions presented within bug reports to enhance prioritization techniques to accurately determine the priority level for a specific report using DRONE (Umer et al., 2018). Within this research, they created an emotional value calculation with approximately 50,000 positive and negative words based on bug report descriptions (Umer et al., 2018). While Umer et al. found a positive correlation between sentiment and bug priority and used that to outperform other approaches, potential issues may arise within open-source bug reporting as anyone can report bugs (Umer et al., 2018).

Yadav and Rathore attempted to overcome existing methods in bug prioritization with *hierarchical attention networks* (HAN) (Yadav and Rathore, 2024). HAN leverages the structure presented within a text to analyze the content at multiple levels. There are apparent deficiencies with automatic approaches, such as DRONE; therefore,

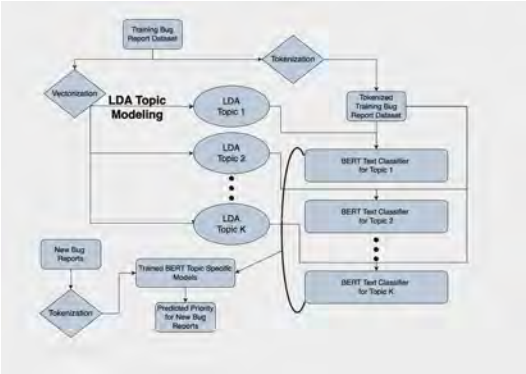


Figure 2: Our BERT Approach Overview (Devlin et al., 2018)

Figure 3: Our Naive Bayes Approach Overview

Yadav and Rathore attempted to solve this dilemma through HAN (Yadav and Rathore, 2024). By first preprocessing, tokenizing, and embedding with the help of DistilBERT, then using the HAN model for feature extraction, final predictive values are given with accuracy that exceeds some other programs.

4 Proposed Approach

Intending to increase the efficiency and effectiveness of prioritization in bug reports, we present an untried algorithmic approach. To ensure consistency among bug report classification within priority levels, we will use Bugzilla's software system for our bug reports as they use P1 through P5 prioritization for classifying their reports. From there, we will execute the approach dictated within subsections 4.1 through 4.3, also demonstrated within Figures 2 and 3.

4.1 Topic Modeling

Once our data has gone through the initial vectorization phase, we plan to create topic models revolving around the bug reports. Most topic models are more generic, though, to ensure that the topics are relevant, we incorporate a similar strategy to Xia et al.'s approach, MTM (Xia et al., 2017). MTM incorporated the product and component fields into their topic modeling approach, textually pre-defined elements within a bug report, leading to improvement within their research. Since the code for MTM is not publicly available, we decided to use LDA with more textual components than simply the description, a technique also used within MTM (Xia et al., 2017).

4.2 Text Classification

After completing the topic modeling segment, we trained multiple text classification systems incorporating BERT, Multinomial Naive Bayes, and Gaussian Naive Bayes for each topic created within the LDA topic modeling to assess which text classifier performs best.

We complete several additional preprocessing techniques for our Naive Bayes text classifiers than we take while using BERT. For Naive Bayes, we use stop word removal, lowercase each word, and tokenize the textual elements of each bug report before training the model.

Before implementing our data within the BERT text classification model, we tokenize the dataset accordingly to make it more manageable for BERT.

4.3 Evaluating Our Results

To test our method, we will compare our results with a baseline approach, Naive Bayes, using precision, recall, and f-measure, similar techniques to various priority improvement approaches (Ali et al., 2024; Umer et al., 2018; Tian et al., 2015); these formulas are included within Table 1.

Table 1: Metrics for Analysis

Formula	Micro-Averaging	Macro-Averaging
Accuracy	$\frac{\sum_i TP_i + \sum_i TN_i}{\sum_i TP_i + \sum_i FP_i + \sum_i FN_i + \sum_i TN_i}$	
Precision	$\frac{\sum_i TP_i}{\sum_i TP_i + \sum_i FP_i}$	$\frac{1}{n}\sum_i \frac{TP_i}{TP_i + FP_i}$
Recall	$\frac{\sum_i TP_i}{\sum_i TP_i + \sum_i FN_i}$	$\frac{1}{n}\sum_i \frac{TP_i}{TP_i + FN_i}$
F-Measure	$\frac{2\sum_i TP_i}{2\sum_i TP_i + \sum_i FP_i + \sum_i FN_i}$	$\frac{1}{n}\sum_i \frac{2 \cdot \frac{TP_i}{TP_i + FP_i} \cdot \frac{TP_i}{TP_i + FN_i}}{\frac{TP_i}{TP_i + FP_i} + \frac{TP_i}{TP_i + FN_i}}$

TP_i are the bug reports predicted as P_i, that specific priority level, and are P_i, FP_i are the bug reports predicted as P_i and are not P_i, TN_i are bug reports accurately described as a different P_i than the one being assessed, and FN_i are the bug reports not predicted as P_i where they actually are part of P_i (Umer et al., 2018).

Accuracy measures what percentage of the time the correct priority classification is made, while precision and recall provide more clarification for the minority classes (Weiss, 2013). Precision is the total number of correctly predicted positive results out of all bug reports expected to be that priority level, similar to recall, which is the total number of correct predictions out of the total number of bug reports with that priority level. F-measure helps gauge the relative importance between precision and recall (Weiss, 2013).

The main difference between micro-averaging and macro-averaging is how skewed data is handled. In micro-averaging, implemented over macro-averaging by most papers involving automated priority levels in bug reports, each bug report has an equal weight. In macro-averaging, however, each class has equal weight instead of each report. Since the priorities are not evenly distributed, with a priority of P3 overtaking 87.9 percent of the data, shown within Figure 4, micro-averaging takes all bug report instances into account equally whereas macro-averaging takes all priority levels into account equally, a beneficial approach for skewed data.

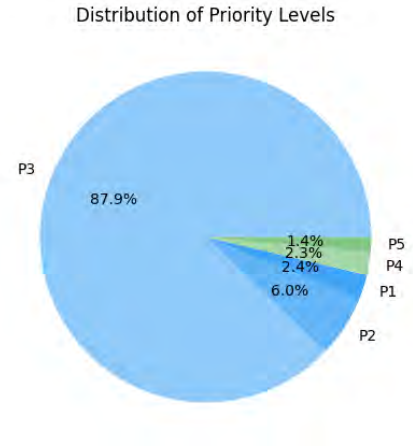


Figure 4: Percentage of Data for Each Priority Level within the Eclipse Platform Project (dat, 2018; ecl, 2018).

5 Experimental Results

5.1 Dataset

For our dataset, we are using a dataset with 85,156 bug reports with resolution dates between 2001 and 2014, as these bugs are much less likely to change priority status after ten or more years, a similar strategy to the one proposed by Tian et al. (Tian et al., 2015). These bugs were obtained through Eclipse, an open software repository for Bugzilla (ecl, 2018). The majority of these bug reports are prioritized as P3, as shown in Figure 4, so there is a strong emphasis within Bugzilla on bug reports with mid-level priority (P3).

5.2 Latent Dirichlet Allocation Topic Modeling

The LDA process used in each of our methods was consistent. We incorporated the title, component, and description to create more accurate topics than simply using the description portion of the bug reports. While the component has pre-defined options within a bug report, we still integrate it within the method because it is a textual field that provides insight into the problem at hand. From this textual data, LDA created ten topics of varying sizes for the baseline approaches to separate the bug reports.

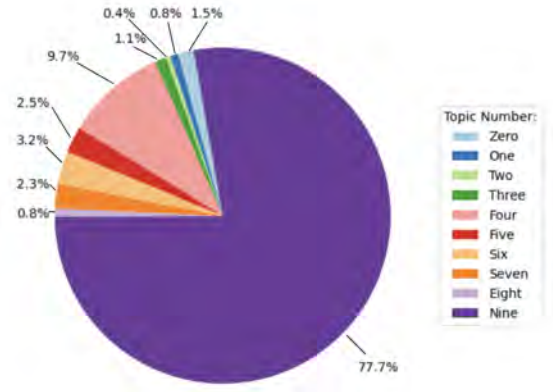


Figure 5: Topic Distributions Using LDA

Within Figure 5, the topic distributions are of visibly differing sizes, with topic nine overtaking most of the data. For the next phase in our pipeline, text classification specific to each topic, we maintain these ten topics for each text classification model.

5.3 Naive Bayes Text Classification

To test the effectiveness of our approach, in addition to comparisons with state-of-the-art methods, we compared our BERT model results to two prioritization pipelines that use Naive Bayes (NB) for text classification. All three of our pipelines used LDA to create topic models, though they deviate when it comes to their text classifiers. For the first comparison to BERT, we used a Gaussian Naive Bayes Classifier for each topic to predict priority. In contrast, we used a Multinomial Naive Bayes Classifier with the same premise in the sec-

ond.

We implemented the Gaussian and Multinomial Naive Bayes text classification methods for each topic to assess the appropriate priority levels. For better context, we analyzed the precision, recall, and f-measure on both a macro-averaging and micro-averaging scale for these NB methods with our BERT approach, and the results for each classification technique are visible in Table 2.

The Multinomial Naive Bayes had slightly better results within the micro-analysis than the Gaussian Bayes. However, when performing a macro-analysis, Gaussian Naive Bayes performed marginally better in precision and recall as it could more easily handle the imbalanced priority levels within our dataset.

One of the main issues with the Multinomial Naive Bayes classification is that it prioritizes almost every bug report under P3 since it is so common within the dataset. At the same time, Gaussian Bayes could have been more accurate in assessing whether a bug report would have a priority other than P3; it still attempted to include other priority levels within its assessment.

When we trained our classifiers with 68,124 bug reports, the Multinomial Naive Bayes classifier took the shortest amount of time and the least memory. Table 3 shows that the Multinomial Naive Bayes classifier takes less than half the time of the Gaussian Naive Bayes model and a small fraction of the time compared to the BERT model's training. Note that our BERT model's peak memory usage exceeds the capacity of our GPU, therefore we partially enlist the help of our CPU during training as well.

Once we obtained the trained models for each approach, we tested the classifiers using the remaining 17,032 bug reports, as depicted in Table 4. For the Gaussian Naive Bayes classifier, it takes 0.02 seconds to prioritize a single bug report. Similarly, a bug report is prioritized in 0.03 seconds for the Multinomial Naive Bayes classifier and 0.27 seconds for the BERT classifier.

Once the Naive Bayes models are trained, contrary to their respective memory usage during training, they use the same amount of memory to classify bug reports. While the Gaussian Bayes classifier took longer in the initial training phase than the Multinomial Naive Bayes classifier, in testing, the Gaussian model takes merely two-thirds the time of the Multinomial model. BERT still takes much more time and memory than the NB alternatives; however, it takes about a quarter of a second to classify a bug report and less than half of a gigabyte (GB) of memory.

5.4 BERT Text Classification

We enlisted BERT for text classification for our proposed approach because of its performance within various tasks (Ali et al., 2024; Yadav and Rathore, 2024). While it excelled in text classification for this task, with results that exceeded all other methods we created and compared to, it took excessive time to train. Regarding the testing phase, BERT also took much longer to classify bug reports than other methods, as visualized in Table 3.

When we trained BERT for text classification, we used fifteen epochs, the number of iterations the model made through our dataset, for the first nine topics. However, for topic nine, since it takes up most of the bug reports conceptualized within Figure 5, it almost took more time than the other topics combined. Therefore, when training our model on this topic, we tested the efficacy using one and three epochs instead of fifteen. Table 5 details the results of using one versus three epochs for this topic.

It was surprising to find that one epoch yielded better results than three epochs. When we compare our pipeline's results using BERT to baseline approaches, we rely on our results from one epoch for topic nine rather than our experimental results, which allocated three epochs to topic nine's training.

6 Discussion

Within Table 6, we compare our results, the Gaussian NB, Multinomial NB, and BERT Priority Assessments, to the works of Yadav et al., Umer et al., and Tian et al. (Yadav and Rathore, 2024; Umer et al., 2018; Tian et al., 2015, 2013). The highest scores for the three main assessment metrics, precision, recall, and f-measure, are in bold.

Our BERT approach achieved more significant precision, recall, and f-measure results than our other methods and baseline comparison approaches. Our precision surpassed other baseline methods by 8.17 percent, recall by 25.12 percent, and f-measure by 19.1 percent. The downfall of this approach is the amount of time it takes BERT compared to other methods.

If we are more concerned about time than greater

Table 2: Comparison of Performance Metrics for Our Approaches

Analysis Method	Prioritization Method	Precision	Recall	F-Measure	Accuracy
Micro	Gaussian	0.4784	0.5234	0.4999	0.7517
	Multinomial	0.4935	0.4968	0.4951	**0.9743**
	BERT	**0.6579**	**0.6514**	**0.6546**	0.9667
Macro	Gaussian	0.2020	**0.2203**	0.1839	0.7517
	Multinomial	0.1949	0.2000	0.1974	**0.9743**
	BERT	**0.3998**	0.2079	**0.2140**	0.9667

Table 3: Efficiency Metrics for Our Methods During Training

Priority Method	Time Approximation	Peak Memory Usage Estimation
Gaussian NB Training	7.22 Minutes	10393 MB
Multinomial NB Training	3.46 Minutes	4075 MB
BERT Training	12+ Hours	11264+ MB

Table 4: Efficiency Metrics for Our Methods During Testing

Priority Method	Time Approximation	Peak Memory Usage Estimation
Gaussian NB Testing	5.80 Minutes	144 MB
Multinomial NB Testing	8.61 Minutes	144 MB
BERT Testing	76.82 Minutes	482 MB

Table 5: BERT Results Using Different Epochs for Topic Nine, Micro-Averaging

Epochs	Precision	Recall	F-Measure	Accuracy
3	0.6087	0.5895	0.5989	0.9676
1	0.6579	0.6514	0.6546	0.9667

Table 6: Micro-Averaging Performance Metrics for Priority Assessment Methods

	Priority Method	Precision	Recall	F-Measure
Our Methods	Gaussian NB	0.4784	0.5234	0.4999
	Multinomial NB	0.4935	0.4968	0.4951
	BERT	**0.6579**	**0.6514**	**0.6546**
Baseline Methods	Emotion Based (Umer et al., 2018)	0.5762	0.3878	0.4636
	DRONE (Tian et al., 2015)	0.2566	0.2669	0.2594
	DRONE (Tian et al., 2013)	0.2973	0.3202	0.2974
	HAN (Yadav and Rathore, 2024)	0.3952	0.4002	0.3739

precision, recall, and f-measure, the emotion-based method performed best in precision, indicating a low rate of false labels for priority levels. In contrast, the Gaussian Naive Bayes achieved higher results than any of the other methods in recall. The likely cause of its higher recall score compared to precision indicates that there may be more falsely labeled priorities than seen within Umer et al.'s research. The Gaussian Naive Bayes method also superseded other approaches in terms of f-measure. While our precision may not exceed others', our method's f-measure emphasizes a more ideal equilibrium between precision and recall.

6.1 Related Works and Our Work

When we compare our approach to the state-of-the-art approaches for topic modeling and text classification, we see some similarities between their methodology and ours.

We incorporated LDA for our topic modeling, though similarly to MTM, we incorporated more natural language than simply the description (Blei et al., 2003; Xia et al., 2017). We differed in our topic modeling approach from Wu et al. because the NTM approach is not as reliable as LDA, and there are apparent flaws aforementioned within our research (Wu et al., 2024).

Regarding text classification, our most effective method used a BERT model for each topic, similar to Ali et al.; however, they were predicting severity (Ali et al., 2024). Tian et al. created their text classification engine, DRONE, though our experimental results show that BERT outperforms DRONE for prioritization tasks (Tian et al., 2013, 2015). One area that we have included in our potential areas for future improvement is the inclusion of sentiment analysis to predict priority, a method also used by Umer et al. with positive results (Umer et al., 2018). Within our approach, we refrained from including this aspect due to the bug reports being open source; since the bug reports are open source, there is potential inconsistent sentiments between user-reported bugs and developer-reporter bugs. We also avoided Yadav and Rathore's approach, HAN, because they did not achieve as significant of results in comparison to the results we achieved using BERT (Yadav and Rathore, 2024).

6.2 Threats to Validity

The approaches in Table 6 are some of the

attempts at automatic prioritization methods for bug reports; however, there are some discrepancies between their work and ours.

All four papers use varying datasets compared to ours, and Yadav and Rathore even extended their work further than Bugzilla bug reports (Yadav and Rathore, 2024). While Yadav and Rathore achieved statistically significant results, the priority distribution between the multiple bug report software systems used within their works does not follow the same prioritization distribution pattern when compared to Bugzilla (Yadav and Rathore, 2024). Since Bugzilla has such skewed prioritization levels, as visualized within Figure 4, it is difficult to accurately predict its priority while assessing it for other software repository systems within the same model. Therefore, to combat this data inconsistency, we exclusively use Bugzilla bug reports to predict the priority of our data.

Then, within Umer et al.'s work, there is a discrepancy in their precision, recall, and f-measure scores for their baseline comparison (Umer et al., 2018). Umer et al. compared their approach to Tian et al.'s approach, using the same dataset, and stated the precision as 54.45 percent, recall as 31.77 percent, and f-measure as 40.12 percent (Umer et al., 2018; Tian et al., 2013). In comparison, Tian et al.'s paper described their precision as 29.73 percent, recall as 32.02 percent, and f-measure as 29.74 percent (Tian et al., 2013). To compare DRONE's results and our own, we used the results obtained within Tian et al.'s papers rather than the statistics presented in Umer et al.'s paper (Umer et al., 2018; Tian et al., 2013). Please note that Tian et al. have two papers to which we compare our results, both of which use identical datasets (Tian et al., 2013, 2015). While none of these papers uses the same dataset as ours, we use a more recent dataset focusing solely on Bugzilla bug reports.

7 Conclusion and Future Work

Natural language processing is the cornerstone of better bug report prioritization. By using LDA and then creating topic-specific text classification models with BERT, automatic priority assignment could become the standard approach. Automatic prioritization may enhance bug triage, saving developers time and money globally.

In our future work, we will seek improvement in several areas. First, we plan to incorporate a linear regression for priority levels, then adjust the weights of priority classes and explore an integrated technique within prioritization that incorporates sentiment analysis. While there are areas of reservation with sentiment analysis, specifically with open-source projects that allow anyone to report bugs, there may be room for further achievement in bug prioritization.

Acknowledgment

All work herein reported is supported by the National Science Foundation under Grant No. 2349452.

Any opinion, finding, or conclusion in this study is that of the authors and does not necessarily reflect the views of the National Science Foundation.

References

2018. bughub.

2018. Eclipseplatform.

CDE Administration. 2012. Tvt34:tct307: Plk: truncation in 'new static web project' wizard.

Asif Ali, Yuanqing Xia, Qasim Umer, and Mohamed Osman. 2024. Bert based severity prediction of bug reports for the maintenance of mobile applications. *Journal of Systems and Software*, 208:111898.

John Anvik, Lyndon Hiew, and Gail C. Murphy. 2005. Coping with an open bug repository. In *Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology EXchange*, eclipse '05, page 35–39, New York, NY, USA. Association for Computing Machinery.

John Anvik, Lyndon Hiew, and Gail C. Murphy. 2006. Who should fix this bug? In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, page 361–370, New York, NY, USA. Association for Computing Machinery.

David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022.

K. R. Chowdhary. 2020. *Natural Language Processing*, pages 603–649. Springer India, New Delhi.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Eclipse Foundation AISBL. How to set severity and priority. https://wiki.eclipse.org/WTP/Conventions_of_bug_priority_and_severity#:~:text=Severity%20is%20assigned%20by%20a,priority%2C%20P5%20is%20the%20lowest. Accessed: 2024-05-28.

Gaeul Jeong, Sunghun Kim, and Thomas Zimmermann. 2009. Improving bug triage with bug tossing graphs. In *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC/FSE '09, page 111–120, New York, NY, USA. Association for Computing Machinery.

Chengnian Sun, David Lo, Siau-Cheng Khoo, and Jing Jiang. 2011. Towards more accurate retrieval of duplicate bug reports. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, pages 253–262.

Yuan Tian, David Lo, and Chengnian Sun. 2013. Drone: Predicting priority of reported bugs by multi-factor analysis. In *2013 IEEE International Conference on Software Maintenance*, pages 200–209.

Yuan Tian, David Lo, Xin Xia, and Chengnian Sun. 2015. Automated prediction of bug report priority using multi-factor analysis. *Empirical Softw. Engg.*, 20(5):1354–1383.

Qasim Umer, Hui Liu, and Yasir Sultan. 2018. Emotion based automated priority prediction for bug reports. *IEEE Access*, PP:1–1.

Gary M. Weiss. 2013. *Foundations of Imbalanced Learning*, chapter 2. John Wiley & Sons, Ltd.

X. Wu, T. Nguyen, and A. T. Luu. 2024. A survey on neural topic models: methods, applications, and challenges. *Artificial Intelligence Review*, 57(18).

Xin Xia, David Lo, Ying Ding, Jafar M. Al-Kofahi, Tien N. Nguyen, and Xinyu Wang. 2017. Improving automated bug triaging with specialized topic model. *IEEE Transactions on Software Engineering*, 43(3):272–297.

Anurag Yadav and Santosh Singh Rathore. 2024. A hierarchical attention networks based model for bug report prioritization. In *Proceedings of the 17th Innovations in Software Engineering Conference*, ISEC '24, New York, NY, USA. Association for Computing Machinery.

Wei Zhang and Chris Challis. 2019. Software component prediction for bug reports. In *Proceedings of The Eleventh Asian Conference on Machine Learning*, volume 101 of *Proceedings of Machine Learning Research*, pages 806–821. PMLR.

Automated Duplicate Bug Report Detection in Large Open-Source Projects

Clare Laney
George Mason University
VA, US
clarelaney26@gmail.com

Armin Moin
University of Colorado Colorado Springs
CO, US
amoin@uccs.edu

Abstract

This paper presents the development of an algorithm designed to detect duplicate bug reports and output their bug IDs automatically. We evaluate six methodologies: topic modeling, Gaussian Naive Bayes, deep learning, time-based organization, clustering, and summarization. Additionally, we introduce a novel threshold-based approach for duplicate identification, in contrast to the conventional top-k selection method. Our threshold approach demonstrated promising results across all five methods, achieving accuracy rates ranging from the high 70%'s to the low 90%'s.

1 Introduction

Large open-source projects offer an issue tracking system or open bug repository, where developers and users can report the software defects they find or any new feature requests they may have. These reports are called *bug reports*. However, some bug reports in the open bug repository are duplicates; they essentially describe the same software defect or feature request with some different phrasing.

Bug triage is the process of finding invalid and duplicate bug reports and assigning the valid and unique ones to the developers working on the project so they can fix and resolve them. In large open-source projects, a group of developers called *triagers*, are typically responsible for the bug triage task. Over the past decades, (semi-)automated approaches based on various Artificial Intelligence (AI) methods and techniques have been proposed to make the bug triage process more effective and efficient, including duplicate bug report detection. This way, the total project costs will be reduced since the developers' time will not be wasted on processing duplicate inquiries and working on redundant tasks.

As explained in Section 3, prior works in the literature have studied the bug triage problem and proposed various AI-based approaches to address it, such as approaches based on Machine Learning (ML) (Anvik et al., 2006) and Information Retrieval (IR) (Sun et al., 2011). Most of them have used the natural language textual data in the bug reports as a crucial source of information. Hence, Natural Language Processing (NLP) plays a vital role. In general, automated bug triage can be considered as a sub-field of Mining Software Repositories (MSR).

This paper proposes a novel approach to automated detection of duplicate bug reports in open bug repositories. Our methods are based on recent advances in NLP, including the BERT transformer model. To this aim, we analyze the natural language textual information in the summary, description, and comments of existing bug reports for a specific project. First, we create several topics and assign each report to one of them. This step is based on the work of Xia et al. (Xia et al., 2017). Second, we use several similarity measures to find highly similar or potentially identical bug reports among those that belong to the same topic.

The contribution of this paper is threefold: First, it proposes and implements a novel approach to automated duplicate bug report detection in which bugs are classified as duplicate or not duplicate. Second, we propose an algorithm to find other similar bugs if a bug is identified as a duplicate. Thirdly, our paper provides an open-source prototype that enables other open-source projects using similar issue-tracking systems to use the proposed approach, thus benefiting from effective and efficient automated detection of duplicate bug reports.

This paper is structured as follows: Section 2 provides some background information about open bug repositories and NLP. Further, we review the literature in Section 3. In Section 4, we propose our novel approach and report on our experimental results in Section 5. Moreover, Section 6 discusses the results and points out potential threats to validity. Finally, we conclude and suggest future work in Section 7.

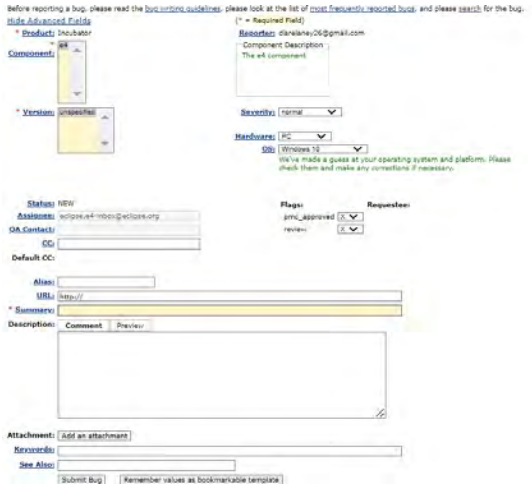


Figure 1: Example of Bug User Interface

2 Background

2.1 Open bug repositories

Bug repositories are a database of bug reports for a software project. In large open-source projects, anyone who creates an account can file a bug report. This leads to many reports. For example, in 2005, a Mozilla triager said, "Everyday, almost 300 bugs appear that need triaging. This is far too much for only the Mozilla programmers to handle" (Anvik et al., 2005)

2.2 Parts of a bug report

These bug reports include various free-form textual data and textually predefined data. Free-form textual data includes data users type in, such as summary and description. The summary is usually a sentence-long overview of the issue. The description is a long-form, detailed outline of the bug. Textually predefined data includes things that the users can select from a drop-down menu, such as the product or component. Product information could include the name or version. The component is a specific module or part of the software product. Figure 1 is an example of what the user interface for a bug report might look like.

2.3 Importance

Duplicate bug detection is important, so triagers are not resolving the same issue. It is also important because additional bugs from different perspectives provide more information on the issue, making it easier to solve. Duplicates occur quite often; up to 39 percent of bug reports in Eclipse between 2018 and 2021 were duplicates or invalid. (Jiang et al., 2023)

2.4 Natural Language Processing

Natural language processing is when a computer interprets human language. One subset of natural language is called preprocessing. Preprocessing is when language is put into a form that is easier for the computer to understand. This could include tokenizing, stemming, and removing stop words. Tokenizing is splitting a sentence into words known as tokens by delimiters. Delimiters include spaces, commas, or punctuation. Stemming is getting words that have multiple forms into a single form. Stop words include very common words with little meaning, such as "an," "is," "the," "and," and "was." Document frequency is how often words appear in a text. Maximum document frequency ignores prevalent words with little meaning (this could include stop words). Minimum document frequency ignores words that appear so infrequently that they are irrelevant. These steps are needed to prepare the summary and description for topic modeling.

3 Related Works

Various works have been published on automated bug reports. Xia et al. work (Xia et al., 2017) used a topic modeling approach to categorize bug reports. This paper shows that topic modeling is an effective way to assign bugs to developers. This paper proposed Multi-feature topic modeling (MTM) as an alternative to the traditional Latent Dirichlet Allocation (LDA). MTM considers the textually predefined data of the bug report (such as the product and component) to improve topic categorization. Our approach implements topic modeling as well. We use LDA to split the bug reports into topics and look for duplicates in the same topics. Sun et al. (Sun et al., 2011) is a relatively early paper on duplicate bug reports. This paper employed an information retrieval function, REP, a linear combination of textually predefined and freeform textual data. The weight of REP is optimized during training using stochastic gradient descent. The larger weights indicate that the feature better distinguishes between similar and non-similar bug reports. This paper also introduced an improved statistical information retrieval technique, BM25F ext, to analyze the similarity of long textual bug report queries. This function multiples the global importance of a word across reports and is local importance. Nguyen et al. (Nguyen et al., 2012) used topic modeling (T-model) and textual similarity (BM25F) to detect

duplicate bugs. This approach improved the REP function by up to 20 percent in some cases. This technique is especially effective for reports that describe the same issue with different technical terms. Similarly, our approach implements topic models before categorization. Zhang et al. (Zhang et al., 2023a) is a literacy review of the previous work on automated duplicate bug reports. Experimental results from this review showed that the REP outperforms more complex neural-based models, cementing the REP function as the state of the art. This paper also found that duplicate bug detection techniques that work well on old bug reports do not perform effectively on recent bug reports. It covers how duplicate bug detection techniques for one issue-tracking system do not work well on other issue-tracking systems. Zhang et al. (Zhang et al., 2023b) proposed a novel approach to duplicate bug reports by leveraging GPT. GPT summarizes long reports and reports with links before they are classified using the REP function created by Sun et al.(Sun et al., 2011). This improved the Recall Rate@10 of the REP by up to 8.7 percent. We also implement GPT for summarization in one of our methods. Xia et al. (Xia et al., 2017) approach trained a convoluted neural network (CNN) to identify duplicate bugs. A model called BR-CNN, developed from a traditional CNN, was trained for the binary classification of bug reports as either duplicates or non-duplicates.One of our approaches uses a neural network as well. Jiang et al. (Jiang et al., 2023) reviewed deep learning in duplicate bug detection. This paper found that deep learning techniques are better at classifying bugs as duplicates or non-duplicates than traditional IR or NLP techniques; however, they are worse at creating a list of possible duplicates of a given bug. In our approach, we tested traditional methods for outputting possible duplicate IDs, but we implemented deep learning techniques to identify a bug as duplicate or non-duplicate.

4 Proposed Approach

4.1 Topic Modeling

We will then conduct topic modeling of both the textual features to split the data into categories. We used LDA to perform topic modeling. This approach is more efficient as it only compares duplicates in the same topics. We split the dataset into seven topics.

4.2 Cosine Similarity

Next, we used the sentence transformer library in Python to embed the bug reports. We compared the bug reports using cosine similarity. Cosine similarity is:

$$\text{cosine\_similarity}(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

where $\mathbf{A} \cdot \mathbf{B}$ is the dot product of the word vectors, and $\|\mathbf{A}\|$ and $\|\mathbf{B}\|$ are the magnitudes (Euclidean norms) of the word embeddings. More similar word embeddings have a smaller angle between them in the word space, and the cosine is larger or closer to 1 (100 percent similar).

5 Experimental Results

5.1 The Dataset

The data set we used is from GitHub's BugHub. The data used is from 2002 to 2013 from Ecispe's Bugzilla. We used around 75,000 bug reports from the dataset. The data contains about 18 percent duplicates. 4.

5.2 Evaluation metrics

We categorized the bugs in two different ways, by threshold and by top-k. Top-k results can be found in table 1, and threshold results can be found in 2.

5.2.1 Threshold

We considered bugs above a certain threshold to be duplicates of each other. We consider the results to be a true positive if at least one bug above the threshold was a duplicate or if it correctly identified a non-duplicate and marked it as a zero. As seen in figure 2, a more restrictive threshold value performs better across all metrics. We evaluated this technique's accuracy, binary accuracy, precision, recall, f1, and accuracy. Accuracy is measured for each of our tasks:

Accuracy This measures how accurately the approach identifies the correct duplicate bug IDs or a zero if there are no duplicates. This evaluates the effectiveness of identifying duplicate bug IDs.

Binary Accuracy This measures the accuracy of the approach in determining whether a given bug is a duplicate or not. This evaluates the performance of a binary classification.

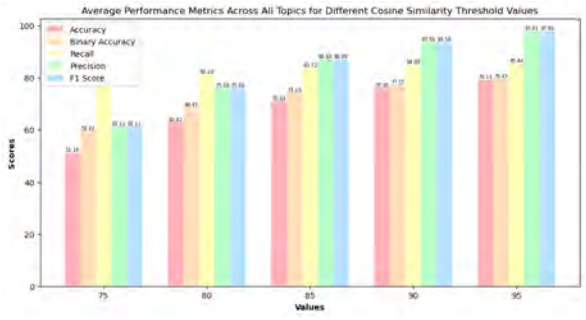


Figure 2: Testing Results for Different Thresholds

5.2.2 Top-K

Another method for categorizing duplicates involves outputting a fixed number of the top K bugs, regardless of their similarity. While top K is currently a state-of-the-art technique, it is less valuable than the previously mentioned threshold approach. The top K approach categorizes items as duplicates, even if they are not very similar, to meet the requirement of always outputting the K results.

RecallRate@K The RecallRate@K is defined as follows:

$$\text{Recall@}k = \frac{\text{Number of relevant items retrieved in top } k}{\text{Total number of relevant items}} \tag{1}$$

For our application, we define the RecallRate@K as:

$$\text{Recall@}k = \frac{|\text{Actual duplicate bug IDs} \cap \text{Experimental top } k \text{ bug IDs}|}{|\text{Number of duplicate bugs}|} \tag{2}$$

This is the state-of-the-art metric.

PrecisionRate@K The PrecisionRate@K is defined as follows:

$$\text{Precision@}k = \frac{\text{Number of relevant items retrieved in top } k}{k} \tag{3}$$

Anything less than 90% similar is considered a non-duplicate and is assigned a zero. For our application, we define the PrecisionRate@K as:

$$\text{Precision@}k = \frac{|\text{Actual duplicate bug IDs} \cap \text{Experimental top } k \text{ bug IDs}|}{|\text{Total number of bugs}|} \tag{4}$$

All of our results, unless otherwise specified, are averaged across all 7 topics.

5.3 The Baseline Approach: Cosine similarity

For our baseline, we used cosine similarity. Bugs above 85% similar are considered duplicates. We compared all bugs with all other bugs in the dataset.

5.3.1 Word Embeddings

An embedding is a way to represent a word numerically in what is sometimes known in NLP as 'word space'. The embeddings of the bug reports' title and description are from the sentence_transformers library. We used the paraphrase-MiniLM-L6-v2 model. Based on these embeddings, we calculated the cosine similarity.

5.4 Topic Modeling and Cosine Similarity

To improve the baseline, we implemented LDA topic modeling as well as cosine similarity to categorize bugs as duplicate or not and output similar duplicate IDs.

5.4.1 Latent Dirichlet Allocation

Preprocessing We merge each bug's title and description into a single column named full_text_data. We then tokenize this column, breaking it down into individual words. After tokenization, we remove irrelevant tokens, such as stop words, words that appear in over 90% of the bug reports, and words that occur less than twice.

Count Vector Next, we create a vector of the count of how often each token occurs in each bug report. The length of this vector is equal to the vocabulary size.

Document-Term Matrix After that, we create a sparse matrix in which each row represents a bug report, and each column represents a word in the vocabulary. We fill in our matrix with the previously calculated count vector.

Term Co-occurrence Matrix Then, LDA creates a term or word co-occurrence matrix, which shows how often words appear together. This model assumes that words that co-occur should be in the same topic. The algorithm then uses both the Document-Term Matrix and the Term Co-Occurrence Matrix to generate topic distributions.

Topic Distributions A topic distribution is a list of the most likely words associated with each of the topics and their probabilities. The topic distribution is chosen randomly and iteratively improved.

Gibbs Sampling Gibbs sampling assigns words to each topic. This process is also iterative. The first fraction represents how important a topic is in a document, and the second fraction represents how often a word is assigned to a given topic throughout

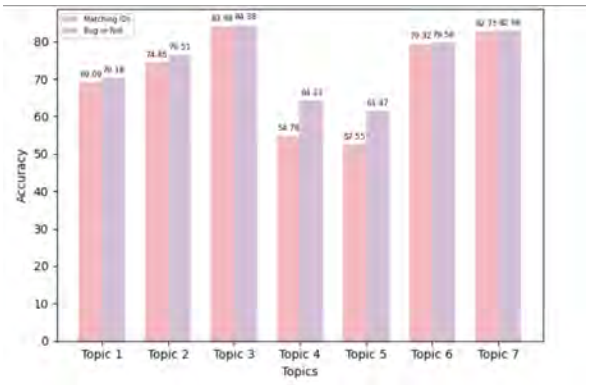


Figure 3: Accuracy for 85% Threshold Similarity Across Topics

all bug reports. The exact formula is below:

$$P(z_i = k \mid \mathbf{z}_{-i}, \mathbf{w}) \propto (n_{d,k}^{-i} + \alpha_k) \frac{n_{k,w_i}^{-i} + \beta}{n_k^{-i} + V\beta} \quad (5)$$

The LDA outputs k topics and assigns each bug report to a topic. We arbitrarily choose to sort the bugs into seven topics, though this needs further experimentation.7. Then we compare the cosine similarity of bug reports within the same topic. For our initial experimentation, we considered 85% similar to a duplicate. Our results for 85% similarity can be found in figure 3. In figure 2, the average performance across topics can be found.

5.5 Deep learning

We also experimented with deep learning techniques to categorize bugs as duplicates or not duplicates. We implemented three models: Multiple layer perceptron (MLP) 4 and BERT 3

5.5.1 Multiple layer Perceptron

MLP is a type of feedforward neural network. In this case, our features were the title and description, and they were embedded using the TfidfVectorizer, which uses the inverse document frequency and the term frequency. Our model used a ReLU activation function.

5.5.2 BERT

BERT or Bidirectional Encoder Representations from the Transformer library is the transformer model that we used to classify our bug reports as duplicate or not. We began by using BERT to tokenize the title and description of the bug reports. We used ten epochs to train the model on our bug dataset. We used the AdamW optimizer from hugging face for our model. The training loss was 0.684, and the validation loss was 0.683. The code used was from Kaggle (https://www.kaggle.com/code/harshjain123/bert-for-everyone-tutorial-implementation/notebook)

5.6 Gaussian Naive Bayes

Gaussian Naive Bayes is a classical machine learning technique. It assumes that the features are independent of each other and that they have a normal distribution. These assumptions are false for our data, hence the poor results.5.

5.7 Time seperated

We experimented with splitting the dataframe into different time sections and comparing the cosine similarity of bugs submitted within similar time frames. For our initial experimentation, we split bugs by quarters or 3-month periods. The theory is that the same issues will be submitted around the same time.

5.8 Clustering

We experimented with K-means clustering as an alternative to topic modeling. We set the hyperparameter of K to 10. Results can be found in table 2

5.8.1 K-means

We used K-means clustering as an alternative to topic modeling. We created ten different clusters and compared the bugs within each cluster. K-means clustering is an algorithm that starts by randomly initializing. K centroids and iteratively updates them to minimize the Euclidean distance between the data points and the centroids.

5.8.2 Choosing K

We chose a hyperparameter of 10 using the elbow method. We tested the even K values from zero to twenty and then computed the sum of squared distances between each point and its nearest centroid, known as its inertia. As the number of clusters increases, the inertia decreases because the distance between a point and its nearest centroids becomes smaller. We create a graph with the y-axis representing the interia and the x-axis representing the number of clustering. Then, by visual inspection, we identify the bend or "elbow" in the plot. This is the point at which the rate of decrease in the inertia slows significantly as you increase the number of clusters. This method should pick a point that balances model complexity and performance.

5.9 GPT Summarization

We experimented with using the GPT 3.5 turbo API to summarize the bug reports before using cosine similarity to compare them. We could only summarize about 2000 bugs due to the computational expenses of this method. Results can be found in table 2

6 Discussion

Now, we will discuss each method further.

6.1 Topic Modeling and Cosine Similarity

We created seven topics. For our initial experimentation, we considered a cosine similarity of above 85 percent threshold as a duplicate. If any bug reports in our experimental duplicate column were also in the actual duplicate column from the Bugzilla dataset, we counted it as accurately identifying the duplicate bug ID.

6.2 Deep learning

MLP models averaged 75 percent. Surprisingly, an MLP model that just had the component as a feature performed better than one that had the textual data as a feature. An MLP model with both the component and the free-form textual data performed similarly to one with jusb the free-form textual data. The transformer model, BERT, was trained on bug data for classification. Its accuracy was somewhat low at 60 percent. However, its F1 score was higher at 31 percent, suggesting it better learned the target.

6.3 Naive Bayes

A Gaussian naive Bayes model was also tested, which performed highest with an 82 percent accuracy for non-duplicates. However, the recall for duplicate bugs was just 6 percent, indicating that the model essentially always guessed "not duplicate," which makes sense with Figure 4.

6.4 Time separated

For this approach, we replaced topic modeling with time separation. We compared similar bug reports within the same 3-month period and categorized those with 85% or higher cosine similarity as duplicates. We created 27 different 3-month time frames. This method performed better than topic modeling. This is likely because the same issues are usually reported during similar time frames.

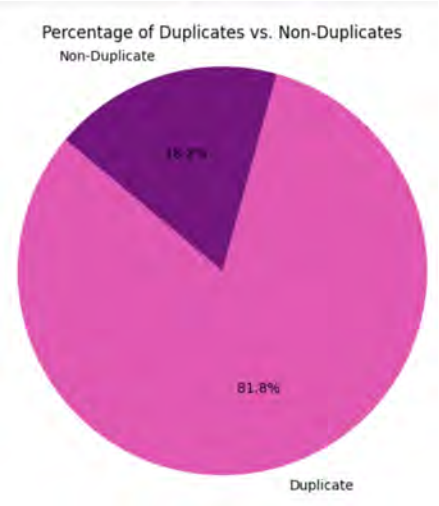


Figure 4: Graph of Percentage of Duplicate Bugs in Our Dataset

6.5 Clustering

We considered bug reports within the same cluster above a cosine similarity of 95% to be duplicates of each other. This method had similar results to topic modeling. This is likely because there were a similar number of topics and clusters. Another possible reason for this similarity is that the embeddings and cosine similarity were done the same way for both clustering and topic modeling.

6.6 GPT Summarization

By summarizing the bug reports, we hoped that the model would phrase the reports more simply so that they would be easier to compare. We also wanted summaries to remove hyperlinks that would be hard to embed accurately and potentially skew our cosine similarity values. We prompted the model with the below paragraph:

> "I want you to act as a collaborator for maintaining bug reports from a large open source software project. Your job is to rephrase the reports to avoid repetition while keeping semantic meaning. The goal of this process is to help filter duplicates. Please only output the summary of the bug report."

We then used cosine similarity with a threshold of 95% to compare and determine the duplicate bug ID based on the summarized descriptions. This heavily improves the accuracy, binary accuracy, and recall of the topic modeling and cosine similarity approach (using the same threshold value.)

Table 1: Average Top-K Results Over Topics

	Our Approach	**REP (State of the Art)**
RecallRate@5	15	64
RecallRate@10	17	70
RecallRate@15	19	73
RecallRate@20	20	77
PrecisionRate@5	80	-
PrecisionRate@10	80	-
PrecisionRate@15	80	-
PrecisionRate@20	80	-

Table 2: Experimental Results for Threshold

	No topic modeling	85% threshold	90% threshold	95% threshold	Time sorted by quarter	Cluster	GPT summary
Accuracy	77	70	76	79	83	81	92
Binary Accuracy	79	74	77	79	83	81	92
Recall	87	83	85	85	85	85	94
Precision	94	87	94	98	100	100	99
F1 Score	90	85	89	98	92	92	96

Class	Precision	Recall	F1-Score	Support
0	0.86	0.61	0.72	8419
1	0.22	0.52	0.31	1800
Accuracy: 0.60 (10219)				

Table 3: Results for BERT model

Class	Precision	Recall	F1-Score	Support
0	0.84	0.86	0.85	14061
1	0.25	0.23	0.24	2970
Accuracy: 0.75 (17031)				

Table 4: Report for Multi-Layer Perceptron

Class	Precision	Recall	F1-Score	Support
0	0.84	0.97	0.90	167
1	0.29	0.06	0.10	33
Accuracy: 0.82 (200)				

Table 5: Report for Native Bayes

The precision and F1 stayed the same. Further experimentation is needed to validate this result, as the summarized approach was implemented with fewer bugs than the topic modeling approach. This idea was inspired by Zhang et al.'s work (Zhang et al., 2023b), which also used summarization and a duplicate bug classification function.

6.7 Threats to Validity

The exact results for the recallRate@K for the REP function are unknown, so we had to look at the chart and estimate the values. We could not replicate the results of the REP function (Sun et al., 2011) on the dataset we used as the code is not public. The precision results for the REP function are not included in their work, which is why that section of the table is left empty.

7 Conclusion and Future Work

We hope to experiment with changing the number of topics used in our topic modeling in the future. We also want to improve the topic modeling from the standard LDA used in these experiments. In this paper, we have discussed six deduplication techniques: topic modeling, Gaussian Naive Bayes, deep learning, time-based organization, clustering, and summarization.

References

John Anvik, Lyndon Hiew, and Gail C. Murphy. 2005. Coping with an open bug repository. In *Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange - eclipse '05*, pages 35–39, San Diego, California. ACM Press.

John Anvik, Lyndon Hiew, and Gail C. Murphy. 2006. Who should fix this bug? In *Proceedings of the*

28th International Conference on Software Engineering, ICSE '06, page 361–370, New York, NY, USA. Association for Computing Machinery.

Yuan Jiang, Xiaohong Su, Christoph Treude, Chao Shang, and Tiantian Wang. 2023. Does Deep Learning improve the performance of duplicate bug report detection? An empirical study. *Journal of Systems and Software*, 198:111607.

Anh Tuan Nguyen, Tung Thanh Nguyen, Tien N. Nguyen, David Lo, and Chengnian Sun. 2012. Duplicate bug report detection with a combination of information retrieval and topic modeling. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pages 70–79, Essen Germany. ACM.

Chengnian Sun, David Lo, Siau-Cheng Khoo, and Jing Jiang. 2011. Towards more accurate retrieval of duplicate bug reports. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, pages 253–262.

Xin Xia, David Lo, Ying Ding, Jafar M. Al-Kofahi, Tien N. Nguyen, and Xinyu Wang. 2017. Improving automated bug triaging with specialized topic model. *IEEE Transactions on Software Engineering*, 43(3):272–297.

Ting Zhang, Donggyun Han, Venkatesh Vinayakarao, Ivana Clairine Irsan, Bowen Xu, Ferdian Thung, David Lo, and Lingxiao Jiang. 2023a. Duplicate bug report detection: How far are we? *ACM Trans. Softw. Eng. Methodol.*, 32(4).

Ting Zhang, Ivana Clairine Irsan, Ferdian Thung, and David Lo. 2023b. Cupid: Leveraging ChatGPT for More Accurate Duplicate Bug Report Detection. ArXiv:2308.10022 [cs].

Mining Software Repositories for Expert Recommendation

Chad Marshall
University of Colorado Colorado Springs
cmarsha6@uccs.edu

Armin Moin
University of Colorado Colorado Springs
amoin@uccs.edu

Abstract

Automatic *bug triaging* software such as Bugzilla, Jira, and GitHub Issues are *bug tracking* systems where users report issues encountered while using the software. The *bug report* is handled by assigning a person or team to verify the bug and assign it to an appropriate developer who can fix the bug. We propose implementing an automatic *bug triaging* system using BERTopic to train and extract latent information for each developer assigned to a triaging team. We use techniques from TopicMiner and its *multi-feature topic model* MTM algorithm to examine a bug report's features, such as its *product and component*. We add *priority and severity* to the features of a bug report to sort and filter developers based on their experience with specific combinations of new reports. We sort developers according to active dates to determine if they are currently assigned *bug reports* to fix. We measure our approach using *Top-k* accuracy and compare our findings with TopicMiner MTM, BUGZIE, *Bug triaging via deep Reinforcement Learning* BT-RL, and LDA-SVM. We use *bug reports* from Eclipse, JDT, Mozilla, Firefox, and Thunderbird to provide various projects.

1 Introduction

Large open-source projects offer an issue tracking system or open bug repository, where developers and users can report the software defects they find or any new feature requests they may have. These reports are called *bug reports*. In some cases, developers can volunteer to work on the reported issues they find interesting or relevant to their field of expertise. Additionally, they sometimes report issues and assign them to themselves. However, in many cases, particularly in large open-source projects, a group of developers, called *bug triagers*, decide who should process and fix a newly reported issue. In fact, *bug triage* is the process of finding invalid and duplicate bug reports and assigning the valid and unique ones to the developers working on the project so they can fix and resolve them.

Bug triagers typically take various factors, such as the developers' availability information and areas of expertise, into account while assigning bug reports to developers. However, this is a tedious and costly task. Over the past decades, (semi-)automated approaches based on various Artificial Intelligence (AI) methods and techniques have been proposed to make the bug triage process, including bug assignment, more effective and efficient. This way, the total project costs will be reduced since the developers' (including bug triagers') time will be used more efficiently. Note that if a bug report is assigned to a developer with the wrong field of expertise or with a lack of proficiency in the matter, it has to be reassigned (i.e., *tossed*) to another developer. This would be costly for the project since the developers' time is an invaluable resource for the project.

As explained in Section 3, prior works in the literature have studied the bug triage problem and proposed various AI-based approaches to address it, such as approaches based on Machine Learning (ML) (Anvik et al., 2006) (including graph learning (Jeong et al., 2009)) and Information Retrieval (IR) (Sun et al., 2011). Most of them have used the natural language textual data in the bug reports as a crucial source of information. Hence, Natural Language Processing (NLP) plays a vital role. In general, automated bug triage can be considered as a sub-field of Mining Software Repositories (MSR).

This paper proposes a novel approach to automated bug assignment in open bug repositories. Our approach is based on recent advances in NLP. To this aim, we analyze the natural language textual information in the summary, description, and comments of existing bug reports for a specific project. First, we create several topic models for each developer based on their bug reports. This step is based on the work of Xia et al. (Xia et al., 2017). Second,

we will recommend a bug report to a developer based on the active dates of the developer (Anvik et al., 2006) (Liu et al., 2022). This step allows us to use the developer's years of experience to recommend bug reports to the best developer.

The contribution of this paper is twofold: First, it proposes and implements a novel approach to automated bug assignment. This is validated by our experimental results using available data from open reference datasets deployed in the related work in the literature. Second, it provides an open-source prototype that enables other open-source projects using similar issue tracking systems to use the proposed approach, thus benefiting from effective and efficient automated bug assignment.

2 Related Work

This section discusses the necessary information for this paper and will help readers understand the approaches we will take.

2.1 Open-bug repositories

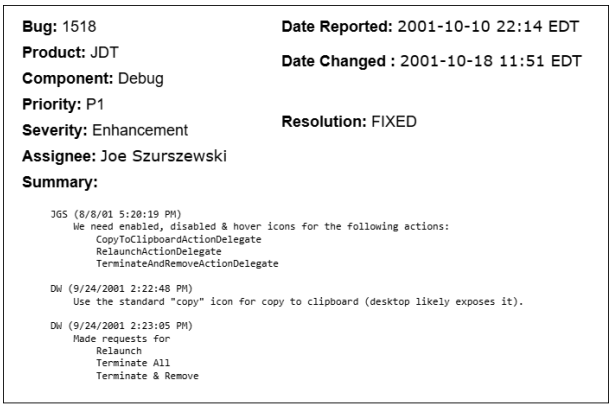
Open-bug repositories allow the user and developer to interact with bug reports and other issues the user may experience when using the software. There is a diverse selection of *bug reporting* software such as GitHub Issues(Fiechter et al., 2021) and Bugzilla (Serrano and Ciordia, 2005) to name a few. Bugzilla (Serrano and Ciordia, 2005) and GitHub Issues (Fiechter et al., 2021) are engineered to make it easy for the user to report bugs. Whenever a user reports a *bug* on GitHub Issues (Fiechter et al., 2021), the issue is posted online so anyone can see it; this allows other users to help resolve the issue if it is a common problem. Bugzilla (Serrano and Ciordia, 2005) only allows the user and developer to interact; this can help the developer get crucial information from the user to fix the issue.

2.1.1 Bug Reports

Bug reports are issues that users can report on software to a developer. The issues a user reports differ from systems and Operating systems (OS), so it is essential for *bug reports* to track any valuable information to the developer. Bugzilla (Serrano and Ciordia, 2005) was one of the first commercially standard *bug-tracking* software that added product and component sections to a *bug report*. Adding product and component makes it easier for the developer to find the source of a bug; to track a bug, you must be able to locate it (). Since the release of Bugzilla (Anvik et al., 2006), the evolution of *bug reports* from software like Jira and GitHub Issues (Fiechter et al., 2021) made the interaction between developer and user a seamless process.

GitHub Issues (Fiechter et al., 2021) is different from Bugzilla (Blei et al., 2003) and Jira because it does not have a traditional template for reporting a bug. It uses a comment-based system where the user can freely post an issue. The problem with this approach is that it allows users to make a duplicate report and requires a triager to read the report. GitHub Issues (Fiechter et al., 2021) addresses this with labels developers can create and attach to the report. The label can be any topic a developer considers appropriate for the issue.

The anatomy of a bug report is broken into product, summary, description, and comments. The product will tell the developer where the user is having issues. The summary will briefly explain the problem, and the description will go more in-depth with details and steps the user tried to fix the issue. The comment section is where the developers and users interact; for GitHub Issues (Fiechter et al., 2021), this is where other users can also comment on a report with suggestions and steps that can potentially fix the bug. The product, summary, description, and comment components of a *bug report* will give our approach the best chance at sorting a bug report into the appropriate category.


Bug: 1518 **Date Reported:** 2001-10-10 22:14 EDT
Product: JDT **Date Changed :** 2001-10-18 11:51 EDT
Component: Debug
Priority: P1
Severity: Enhancement **Resolution:** FIXED
Assignee: Joe Szurszewski
Summary:

JGS (8/8/01 5:20:19 PM)
We need enabled, disabled & hover icons for the following actions:
CopyToClipboardActionDelegate
RelaunchActionDelegate
TerminateAndRemoveActionDelegate

DW (9/24/2001 2:22:48 PM)
Use the standard "copy" icon for copy to clipboard (desktop likely exposes it).

DW (9/24/2001 2:23:05 PM)
Made requests for
Relaunch
Terminate All
Terminate & Remove

Figure 1: Eclipse bug report #1518

The figure above 1 shows an example of Eclipse bug report #1518; from the 1, we see that the product is the sub-project of Eclipse, and the component is the specific issue the bug was found. Priority and severity are vote-based and set by the trainers and developers, and the summary gives a brief description and example of how to recreate the issue. The date reported is when the bug report was issued, and the date changed is the last date any changes

were made to the report. Resolution is a tag that lets developers mark if a bug is fixed, duplicated, or invalid.

2.2 Natural Language Processing

Natural Language Processing (NLP) is a subfield of Artificial Intelligence (AI) (Joshi, 1991) that relates to how humans communicate with machines, indulging human language comprehension. NLP uses mathematics and computation to model *lingusitics*. *Lingistics is the scientific study of language significant to NLP with subdivisions including Syntax - the rules of sentence structure; Semantics - the study of linguistic development; and Pragmatics - the analysis of situational context in languages*. The theoretical side of NLP deals with Linguistics and the subdivisions of primary human language. NLP has many concepts that help with various tasks depending on the situation. The concepts we are most interested in are text classification and neural networks. The following sections will briefly introduce language models, text classification, and topic mining.

2.2.1 Language Models

The elementary level of language models is called N-grams. N-grams are probabilistic models that use the words in the sentence to predict the outcome of the next word. The equation is simple: *P* is the probability of the next word, *w* is the word, and *h* is the history (or sequence of previous words in the sentence).

$$P(w|h) \tag{1}$$

Equation 1.(Jurafsky and Martin, 2000) Probability of word From equation 1., we can predict the probability of the next word. For example, we can predict the word for the sentence, *The Earth is* by comparing the outcome of *round* or *flat*.

$$P(\text{flat}|\text{The Earth is}) = 0.2 \tag{2}$$
$$P(\text{round}|\text{The Earth is}) = 0.8 \tag{3}$$

Since the probability of *round* is higher than *flat*, the equation can predict that the following word in the sequence is *round*.

2.2.2 Topic Mining

Topic modeling algorithms such as LDA (Blei et al., 2003), topicMiner MTM (Xia et al., 2017), and BERTopic (Grootendorst, 2022) search a whole dataset to group documents into clusters. The traditional approach of topic modeling with LDA (Blei et al., 2003) treats each document as a bag of words approach. LDA (Blei et al., 2003) looks at each word in a document and assigns that word to a topic; the process is repeated until the model has converged documents of a similar nature together. TopicMiner MTM (Grootendorst, 2022) extends LDA by creating a topic modeling algorithm designed for bug triaging. MTM looks at *bug reports* features instead of summary or description as the central aspect of clustering. The combination of *Product and Component* and the filtered summary text allows MTM to cluster documents based on combinations. BERTopic (Grootendorst, 2022) is a new topic modeling algorithm; the model uses a combination of traditional and new methods to allow a fully customized model that works for whatever task it is designed for. BERTopic works well with dimensionality and clustering techniques to group documents of similarity into clusters; the model also adds a human representation of the topics to make it easier to understand a topic.

3 Related Work

This section will explain TopicMiner (Xia et al., 2017) and how it categorizes each bug report into topics assigned to a developer. We will explore LDA (Blei et al., 2003) and explain how we plan to train our model based on the workings of TopicMiner (Xia et al., 2017). The section will break down the core components of the models and explain how each section is processed. We will then explain the concept of Naive Bayes Classifier and Neural Networks. We will break down the core concept of the models and the best use cases for each model.

3.1 TopicMiner(MTM)

This model extends the Latent Dirichlet Allocation (LDA) (Blei et al., 2003) by converting the words found in bugs into phrases. LDA (Blei et al., 2003) allows duplicate reports to combine into a single topic and helps the bugs that are worded differently separate into their proper topic. TopicMiner (Xia et al., 2017) extends LDA (Blei et al., 2003) by separating each developer into a category that matches the bug reported. Xia et al (Xia et al., 2017) proposes three phases: model construction, recommendation, and model update. (Blei et al., 2003)

3.1.1 The module construction phase

During the model construction phase, Xia et al.(Xia et al., 2017) use the training setup from LDA (Blei et al., 2003). During this stage, MTM (Xia et al., 2017) initializes each word in each *bug report* and *additional features* are randomly assigned to a topic. The model will iterate this process to a maximum of 500 iterations to determine the likelihood estimation of the word belonging to a topic. The process involves three variables: *topic distribution, topic assignment vector*, and *topic word vector* (). MTM calculates the variables using Gibbs Sampling (Gelfand, 2000). Gibbs sampling is a Markov Chain Monte Carlo (MCMC) that samples from the observed word and tries to converge it to the target (Jurafsky and Martin, 2000). During the training phase, MTM (Xia et al., 2017) estimates the values of each word in a topic for every bug report by looking for the largest posterior.

3.1.2 The recommendation phase

In the recommendation phase, TopicMiner (Xia et al., 2017) recommends a list of developers for new bug reports based on reports the individual developers fixed in the past, this is found from the model construction phase. To find the recommended developer for a new bug report, TopicMiner (Xia et al., 2017) combines a bug report's product and component section into one category and compares the topics to each developer's history of bug fixes. The model then outputs the top 5 recommended developers.

3.1.3 The model update phase

After MTM (Xia et al., 2017) finds the top 5 developers with the highest chance of fixing the new bug report, the model then updates the topic probability of the developer.

3.2 Latent Dirichlet Allocation (LDA)

Latent Dirichlet Allocation (LDA) (Blei et al., 2003) is a probabilistic model that uses a large body of text such as *documents* and *corpora* to create and distribute topics. LDA (Blei et al., 2003) uses a three-level Bayesian model to classify the topic of a *document*. At the basic level, LDA (Blei et al., 2003) solves the variational inference problem using Bayesian methods. A variational inference approximates the posterior distribution of the latent variables for complex *documents* and *corpora*. Using the variational Expectation Maximization (EM) procedure, LDA (Blei et al., 2003)[] finds an approximate posterior distribution for the model parameters. EM steps are repeated until the E-step finds the optimized value of the variational parameters, and the M-step maximizes the Evidence Lower Bound(ELBO) with respect to the model parameters.

3.3 BERTopic

BERTopic (Grootendorst, 2022) is a transformer-based topic modeling algorithm, unlike traditional methods of LDA (Blei et al., 2003). BERTopic gives the user complete control over how the model clusters documents together. The algorithm combines techniques used for machine learning algorithms, such as tokenizers, dimensionality reduction, clustering, and embeddings. BERTopic starts with document embeddings; the default for this parameter is sentence transformers. The following steps are the most important for our goal. First, we will look at dimensionality reduction; this step is crucial for high-dimensional data and can improve clustering and performance depending on your chosen type. The types of dimensionality reduction we will experiment with are Principal Component Analysis PCA () and Uniform Manifold Approximation and Projection for Dimension Reduction UMAP (McInnes et al., 2020). Once we reduce the dimensions of our dataset, we can apply a clustering technique we want our model to follow; the default clustering technique is Hierarchical density-based clustering HDBSCAN (McInnes et al., 2017). This technique enables the tuning of parameters to change the metric used for clustering. Once HDBSCAN applies the specified metric, it clusters the documents based on density and metric, removing outliers from the clusters. We also will see how HDBSCAN compares to KMeans (Joshi, 1991). KMeans (Joshi, 1991) clusters documents differently; instead of using the density of documents, it sets a predefined number of topics to look for and then clusters the documents based on how far apart the document is from the topic. KMeans will be helpful since we can set a predefined number of topics. However, we might run into issues with developers with a sparse set of bug reports not being represented and developers with a few bug reports unable to cluster *bug reports*. BERTopic uses a count vectorizer for the tokenizer by default; we can play with this parameter to see if the traditional Term Frequency Inverse Document Frequency TF-IDF is a better option for tokenization. We will

compare the default settings of BERTopic and try to fine-tune our parameters to create a cluster of topics for each developer.

Figure 2: BERTopic Diagram

4 Proposed Approach

Our approach is to use BERTopic (Grootendorst, 2022) to create a model for each developer. We will then use a sorting algorithm to filter the developer based on the date of a bug report to test the accuracy and continue training our model for updates. For the models that are over a certain threshold, we will combine clusters and reduce the number of topics; if the model does not have enough valuable information, then we will increase the clustering. We will then save the model and repeat the process.

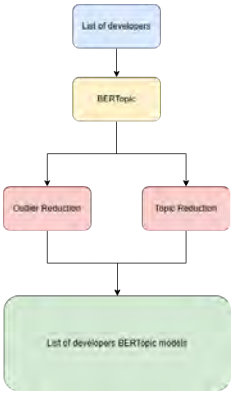


Figure 3: Our Construction Phase

BERTopic (Grootendorst, 2022) is a valuable tool that allows you to have complete control over the structure of your model. In this subsection, we will break down the creation of the model we used for this approach. The model is constructed together using six individual subsections to guide the model into creating clusters of documents. We started by using transformers to create our document embedding; this was the default for the BERTopic model, and we found that this worked better than traditional embedding methods. We will then use a dimensionality reduction to preserve the information in our data; we will compare Uniform Manifold Approximation and Projection UMAP (McInnes et al., 2020) and PCA (Wold et al., 1987) to determine the best. We will compare the use of Hierarchical density-based clustering HDBSCAN (McInnes et al., 2017) and KMeans (Krishna and Murty, 1999). Both algorithms work well in different ways. We will use a count vector, remove the stop words from, and experiment with different ngram ranges to determine how increasing or decreasing the ngram range helps with our model. 3 shows a diagram of our model construction. We first train our current developers with over a set amount of bug reports fixed on a shuffled dataset. The dataset is shuffled so that we can train the model to learn the developer's pattern of bug reports to fix. Once the models are trained, we will use our testing data to predict the type of bug report to the developer.

Once the model is trained, we will use a sorting algorithm to filter out the dates and select the active developers during a bug report. We get our inspiration from BT-RL (Liu et al., 2022) and Avik (Anvik et al., 2006). They proposed looking at active developers and recommending a new incoming bug report based on the recent activity. We plan to create our model based on the dates a developer was active to test the model's accuracy for predicting a developer based on the time they were active and the new *bug reports*. Priority and severity also play an essential role in the triaging process, so we will add those additional features to the BERTopic model so that the machine can create a profile of the developers based on their underlying topics. Once our algorithm filters developers by the date, priority, and severity level, we will test each BERTopic model that is relevant to an incoming bug report and find the highest probability a developer has of fixing a bug report. We display the *Top-k* number of developers with the best chance of fixing a bug report. If we have an even split of developers who are recommended as the best developer for fixing a *bug report*, we will force the model to choose the best developer based on the date of activity. If a developer activates the most around the time of a *bug report* and the developer has the experience with fixing the type of report, then that developer will become the *Top-1* developer. **??** shows the breakdown of the sorting algorithm.

5 Experimental Results

To measure our approach's effectiveness, we find the *Top-K* accuracy that our model predicts the correct developer. We do this to compare our results with other research (Anvik et al., 2006)(Xia et al., 2017)(Liu et al., 2022). To find the *Top-K accuracy*,

Table 1: Statistics of bug reports before filtering

Project	Period	Amount	Product	Component	Combination	Developers	Final Amount	Final Developer
Eclipse	2001/10/10 - 2013/12/30	85,156	4	21	25	335	42,414	296
Mozilla	1997/03/28 - 2013/12/31	205,069	-	130	-	-	101,500	-
Firefox	1999/07/30 - 2013/12/31	115,814	-	52	-	-	19,270	-
JDT	2001/10/10 - 2013/12/31	45,296	3	6	11	172	22,750	138
Thunderbird	2000/04/12 - 2013/12/31	32,551	1	23	23	380	5703	12

we count the times our model guesses the correct prediction over the total number of predictions.

$$\textit{Top-k} = \frac{\#ofcorrect}{Total\#ofpredictions} \tag{4}$$

We also measured the precision and recall of our model to compare our results with other research papers (Anvik et al., 2006). To measure the recall accuracy, we count the total number of correct developers over the number of correct developers plus false developers. Precision counts the number of correct developers over the number of correct developers plus the number of incorrect developers.

$$\textit{Recall} = \frac{\#\text{ of correct}}{\#\text{ of correct} + \#\text{ of False Negatives}} \tag{5}$$

$$\textit{Precision} = \frac{\#\text{ of correct}}{\#\text{ of correct} + \#\text{ of False Positives}} \tag{6}$$

5.1 Experimental Setup

We used 486,591 bug reports from the bughub GitHub repository (Yuan et al., 2024); the repository uses bug reports from Bugzilla (Serrano and Ciordia, 2005) websites. We added three additional columns to each of the bug reports, *Assignee Real Name* the name of the Developer, *Product* this is the platform the bug is occurring, *Severity* how severe a bug report is. With the addition of the new *features* of a bug report, we will create a BERTopic model (Grootendorst, 2022) for each Developer relevant to the Project. We filtered our dataset based on the previous works (Xia et al., 2017)(Anvik et al., 2006)(Liu et al., 2022) and decided a developer is relevant to a project based on the number of bugs fixed in the past. We filtered the dataset further by removing generic developer names from our list (Xia et al., 2017) shows a breakdown of the final number of developers. The *Project* is the name of the bug reports and where they were pulled from, *Amount* is the total number of bug reports in the dataset, *Product* is the total number of products in the bug report, *Component* is the total number of Component in the bug report, *Combination* is the total Combination of Product and Component combined, and *Developer* is the total number of developers. Additionally, we filtered out the developers that fixed over 100 *bug reports*; this helps the BERTopic model make enough clusters that it can be used to predict *bug reports*. We can use developers that have fixed *bugs* less than 100 if we apply different types of dimensional reduction and clustering means.

Each developer will have a BERTopic model fine-tuned to the *bug reports* marked as *fixed* by the developer; we do this to ensure that the models receive accurate information about a developer's past. We first filter our data set by the developer and then do an 80/20 split on the developer's bug list. Once the model is trained, we will reduce the number of topics for larger models to a smaller size; we tested different values and found that reducing the more prominent models to ten topics allows for the models to retain a high probability score and allows smaller models to have a chance of being a top pick. We reduced the number of outliers for smaller models, allowing the models to have more representation from the developer's documents. Once a developer's BERTopic model finishes, we save the model and repeat the process for the remaining developers.

5.2 Results

We applied a simple accuracy test to our model and compared it to TopicMiner MTM (Xia et al., 2017), BT-RL (Liu et al., 2022), LDA-SVM, LDA-KL, and BUGZIE (Tamrawi et al., 2011). Among the baseline models, TopicMiner MTM (Xia et al., 2017) performed the best compared to the other approaches. We compared Eclipse, Mozilla, Firefox, Thunderbird, and JDT bug reports. We measured the *Top-K* accuracy of our model, focusing on *Top-1* through *Top-5* accuracies. We treated the testing phase as the recommendation phase for a new bug report received, so we needed to ensure that our model was trained on the datasets in a time

sequence manner. Each developer's bug reports were split into a test and training set, meaning the models will predict new bug reports unseen by the developer up to the time report split. This approach gave us a real-world view of how accurately our model predicted the correct developer.

Table 2: Top-1 Accuracy

Projects	BERTopic	MTM	BZ	BT-RL	LDA-SVM
Eclipse	-	0.64	0.39	0.46	0.15
Mozilla	-	0.52	0.30	0.20	0.15
Firefox	-	-	-	-	-
JDT	0.97	-	-	-	-
Thunderbird	0.82	-	-	-	-
GCC	0.88	0.51	0.27	-	0.21
Average	0.89	0.55	0.32	0.33	0.17

Table 3: Top-5 Accuracy

Projects	BERTopic	MTM	BZ	BT-RL	LDA-SVM
Eclipse	-	0.90	0.71	0.78	0.35
Mozilla	-	0.78	0.72	0.68	0.32
Firefox	-	-	-	-	-
JDT	1.00	-	-	-	-
Thunderbird	0.92	-	-	-	-
GCC	0.91	0.80	0.59	-	0.49
Average	0.94	0.81	0.68	0.73	0.39

6 Discussion

6.1 Threats to Validity

6.1.1 Internal Threats

We tried to find and recreate the models we used to compare our results but could not find them. So, we compared the results of the models obtained in their research with the appropriate project to keep the measurements and compare our model similarly. Our approach can take up a lot of space depending on the number of developers since we are creating one BERTopic (Grootendorst, 2022) model per each developer. Additionally, adding a new developer is easy however for our model to best work it is recommended that the developer has fixed over a certain amount of *bug reports* before creating a model. The issue can be fixed with using a different type of clustering and dimensional reduction, we found that KMeans (Krishna and Murty, 1999) and PCA (Wold et al., 1987) are good options to start and test the difference. Our approach also relies on *bug reports* with dates as we found the best for filtering developers.

6.1.2 External Threats

The *bug reports* we used for our testing was pulled from the bughub GitHub (Yuan et al., 2024). The total amount of bug reports used from the repository totaled 486,591 reports. The projects used for the research were limited to sub-projects of Eclipse and Mozilla. The Eclipse sub-projects are Eclipse-platform and Eclipse-JDT. The Mozilla sub-projects are Mozilla-firefox, Mozilla-core, and Mozilla-Thunderbird. Since the dataset used with our model is limited, we need to apply different projects so that we can get a broader range of different types of bug reports.

7 Conclusion and Future Work

We made the model for this project open-source and encouraged the development of our project for future research and projects. In the future, we plan to improve the model's efficiency to see if using the BERTopic (Grootendorst, 2022) model merge feature significantly impacts the predictability of new bug reports to developers in the future. We would also like to improve our filtering process using a neural network or a decision tree. That way, we can train the model on the dates of a developer and allow it to filter the developers automatically. We will also add more projects and bug reports to test our model from *bug list* on Bugzilla's website, allowing us to test a broader range of developments and developers. The goal is to be able to apply the model with something like GitHub issues (Fiechter et al., 2021); as of now, we will continue to use Bugzilla for its ease of access.

We applied a traditional approach to automatic bug triaging using topic modeling. We get our motivation from the works of previous research papers (Blei et al., 2003)(Anvik et al., 2006)(Xia et al., 2017)(Liu et al., 2022) and extend the works of automatic bug triaging with a new approach of applying BERTopic (Grootendorst, 2022) and using a sorting algorithm. We differ from the previous research by creating one topic model per developer, allowing us to cluster the *bug reports* marked as *FIXED* to create a pattern of *bugs* the developer has the best chance of fixing. We also found that we can filter our developers using the dates, priority, and severity of *bug reports* to allow our model a better chance of creating a *Top-K* of developers with the best experience to fix a particular bug. From this, we can guide our model to look at developers and choose who is best at fixing a certain *bug report* based on the experience level or the number of years they actively worked on a project. With the additional features, we added to

the BERTopic models, and the sorting algorithm applied to developers once a new *bug report* is received, we were able to beat previous research papers by a significant amount. We can further improve the accuracy of our model by applying a Neural Network or some other kind of decision-based algorithm and learning the developer's level of experience based on the total number of years, bug reports, priority, and severity level bug reports they have fixed. Adding other types of models trained on years of experience with the ability to look at the type and time frame of a bug report would help the model learn how to categorize in a history-based instead of only looking at the features of a *bug report* we learned that we could help our model make correct predictions through the use a sorting algorithm. The sorting algorithm with BERTopic (Grootendorst, 2022) opens the door for future work to improve the efficiency and learning of the model.

8 Acknowledgment

Acknowledgement

All work herein reported is supported by the Nation Science Foundation under Grant No. 2349452.

Any opinion, finding, or conclusion in this study is that of the authors and does not necessarily reflect the views of the National Science Foundation.

References

John Anvik, Lyndon Hiew, and Gail C. Murphy. 2006. Who should fix this bug? In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, page 361–370, New York, NY, USA. Association for Computing Machinery.

David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022.

Aron Fiechter, Roberto Minelli, Csaba Nagy, and Michele Lanza. 2021. Visualizing github issues. In *2021 Working Conference on Software Visualization (VISSOFT)*, pages 155–159. IEEE.

Alan E Gelfand. 2000. Gibbs sampling. *Journal of the American statistical Association*, 95(452):1300–1304.

Maarten Grootendorst. 2022. Bertopic: Neural topic modeling with a class-based tf-idf procedure. *arXiv preprint arXiv:2203.05794*.

Gaeul Jeong, Sunghun Kim, and Thomas Zimmermann. 2009. Improving bug triage with bug tossing graphs. In *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC/FSE '09, page 111–120, New York, NY, USA. Association for Computing Machinery.

Aravind K Joshi. 1991. Natural language processing. *Science*, 253(5025):1242–1249.

Daniel Jurafsky and James H Martin. 2000. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall.

K Krishna and M Narasimha Murty. 1999. Genetic k-means algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 29(3):433–439.

Yong Liu, Xuexin Qi, Jiali Zhang, Hui Li, Xin Ge, and Jun Ai. 2022. Automatic bug triaging via deep reinforcement learning. *Applied Sciences*, 12(7):3565.

Leland McInnes, John Healy, and Steve Astels. 2017. hdbscan: Hierarchical density based clustering. *The Journal of Open Source Software*, 2(11):205.

Leland McInnes, John Healy, and James Melville. 2020. Umap: Uniform manifold approximation and projection for dimension reduction.

N. Serrano and I. Ciordia. 2005. Bugzilla, itracker, and other bug trackers. *IEEE Software*, 22(2):11–13.

Chengnian Sun, David Lo, Siau-Cheng Khoo, and Jing Jiang. 2011. Towards more accurate retrieval of duplicate bug reports. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, pages 253–262.

Ahmed Tamrawi, Tung Thanh Nguyen, Jafar M Al-Kofahi, and Tien N Nguyen. 2011. Fuzzy set and cache-based approach for bug triaging. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pages 365–375.

Svante Wold, Kim Esbensen, and Paul Geladi. 1987. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52.

Xin Xia, David Lo, Ying Ding, Jafar M. Al-Kofahi, Tien N. Nguyen, and Xinyu Wang. 2017. Improving automated bug triaging with specialized topic model. *IEEE Transactions on Software Engineering*, 43(3):272–297.

Deyi Yuan, Gong Luo, and Xiang Qin. 2024. Bughub: A benchmark dataset for bug report analysis. `https://github.com/logpai/bughub?tab=readme-ov-file`. Accessed: 2024-07-08.

Natural Language Processing with TinyML

Andrew Barovic
University of Colorado Colorado Springs
abarovi2@uccs.edu

Armin Moin
University of Colorado Colorado Springs
amoin@uccs.edu

Abstract

The deployment of a complex keyword model to resource-constrained or edge devices opens up a range of applications, from smart home systems to voice-activated devices. In this project, we utilize a transfer learning model, to achieve high accuracy with a small dataset. The final model boasts a 97% accuracy rate and is trained on over an hour of audio data. Edge Impulse is instrumental in this process, facilitating data collection, processing, model selection, training, data analysis, and deployment to the Arduino Nano 33 BLE Sense chip. This chip is specifically designed for IoT and AI applications, making it an ideal choice for this scenario. While most existing research focuses on a limited set of keywords, our model can process 23 different keywords/labels, enabling it to perform complex functions based on these inputs. This advancement significantly broadens the potential use cases for edge devices, demonstrating the feasibility and effectiveness of deploying sophisticated AI models on resource-constrained hardware.

1 Introduction

Natural Language Processing (NLP) is a crucial area of Artificial Intelligence (AI), which deals with enabling computers to analyze and understand human language (usually in textual form), reason on it, and respond to it. Machine Learning (ML) is also a sub-discipline of AI. ML methods and techniques are applied in NLP, among other areas.

ML models are trained on large amounts of data using ML algorithms. There exist various ML model families, such as Decision Trees, Support Vector Machines (SVMs), and various architectures for Artificial Neural Networks (ANNs). Training advanced ML models (e.g., deep ANNs) typically requires a huge amount of data, as well as intensive computational resources. Once trained, ML models can perform (e.g., make predictions) with a lower expectation of computational resources. However, their sizes are still typically too large to fit into the main memory of many resource-constrained Internet of Things (IoT) devices, such as embedded microcontrollers in sensor motes, which possess only a few Kilobytes of memory. With the advent and rapid expansion of the IoT, this has become a more interesting and relevant problem in many domains.

TinyML is a recently emerged sub-field of ML, which involves ML models that can be deployed on microcontrollers with energy consumption in the order of milli-watts (mW). They typically also possess a main memory in the order of Kilobytes (KB). The challenge is to make ML models compact enough to fit into the limited memory and consume a very low amount of energy without compromising the performance (e.g., drop in accuracy, precision, and recall) at a minimal and acceptable level. Note that compacting ML models often implies applying quantization techniques to numeric values (e.g., parameters' weights) that inevitably degrade their performance capabilities.

In this paper, we concentrate on ML models deployed in NLP and aim to fit them onto the main memory of a resource-constrained IoT platform, namely the *Arduino Nano 33 BLE Sense* (Arduino Nano 33 BLE Sense) microcontroller board with a main memory (SRAM) of 256 KB. We use the Tensor Flow Lite library (TensorFlow Lite; TensorFlow Lite for Microcontrollers) which has seen improvements in its use regarding NLPs (TensorFlow Blog, 2020). The developers behind the Tensor Flow Lite library made improvements to an existing NLP model MobileBert managing to improve the size and speed of the model while only observing a minor drop in accuracy. This process was achieved via quantization.

TinyML has strong applications in IoT-related use cases (Dutta and Bharali, 2021). This is due to the advantage of local computing as opposed

to cloud computing most evident when the size of data being processed is large. It is often preferred to have a small microcontroller handle the majority of data and then work together in an IoT environment to bring a synchronous product. Some concrete applications of this approach can be observed in smart cars or smart homes where the car sensor data (audio or distance) is processed by a local TinyML.

This paper is structured as follows: Section 2 provides some background information about TinyML, NLP, and IoT. Further, we review the literature in Section 3. In Section 4, we propose our novel approach and report on our experimental results in Section 5. Moreover, Section **??** discusses the results and points out potential threats to validity. Finally, we conclude and suggest future work in Section 6.

2 Background

2.1 TinyML

TinyML or Tiny Machine Learning is the process of training, downloading, and using machine learning on a resource-constrained device such as a microcontroller (Ray, 2022). These machine learning models are typically scaled down to fit on the main memory of the chip.

TinyML has applications in various fields often being a great alternative to processing data on internet-heavy systems. They either allow for redundancy or efficiency in already existing systems that bring a suite of benefits to those who use them. Some of these could be cost, performance, redundancy, etc. Additionally, these microcontroller devices can also be used to run sensors or other data collection devices often also being able to process the data in some way before combining it. It is via the Internet of Things that this process maintains its connection and thus the use of the TinyMLs.

From a practical approach, TinyML holds a multitude of uses. Machine Learning models could be deployed for voice or text recognition i.e. Siri or Google Assistant on mobile platforms. They can also be used to potentially process spoken or written language as a whole to perform a general series of tasks on a use-case basis; for example, listening for a keyword to turn on the lights in a smart home or creating a direct translation via an onboard speaker.

In this case, a simpler approach will be taken via a keyword analyzer that processes more complex commands. Additionally, TinyML sees uses when running any computing system that requires low power or long battery life. Ideally using a cheap, low-cost, and rapidly mass-producible model. The growth of this technology will bring benefits to a multitude of fields.

2.2 NLP

Natural Language Processing is a series of techniques and methods that are used to quantifiably interpret language (Jurafsky and Martin, 2024). The core of this process is handled by a series of probabilistic equations typically in connection with neural networks or other machine learning models. These equations help determine the likelihood of certain words appearing after a given word or set of words; it is important to note that these "words" can be quantified through any form of language with the most common applications being speech and text. When these equations are used in tandem with machine learning it provides an interesting application of the processes that can interpret any form of language as long as the training data is significant enough. These language models with significant training data are typically referred to as Large Language Models (LLMs).

NLP also includes various techniques of categorizing speech in order to improve or direct the use of NLP in ML. One such method can be explored via a grammar classification i.e. breaking up nouns, verbs, adjectives, etc. These classifications help LLMs determine and properly recognize speech heavily decreasing the training data needed to output understandable text.

One common metric for measuring a language model's performance is via its F_1 score. This score is graded on a metric of *precision* and *recall*. Precision is a measure of the number of positive items that the system correctly labeled. Recall represents the number of items that the system labeled correctly. Mathematically precision is represented by the following equation (Jurafsky and Martin, 2024):

$$Precision(P) = \frac{truepositives}{truepositives + falsepositives} \quad (1)$$

Next, the equation for Recall is represented by the following equation (Jurafsky and Martin, 2024):

$$Recall(R) = \frac{truepositives}{truepositives + falsenegatives} \quad (2)$$

The equation for the F_1 metric then uses the following formula (Jurafsky and Martin, 2024):

$$F_1 = \frac{2PR}{P+R} \tag{3}$$

Equation 3 is one of the more common ways of evaluating the performance of text classification as it provides a more useful account of performance then simply using accuracy would.

2.3 IoT

The Internet of Things is a broad classification category typically referring to the interconnection of internet devices via the internet (noa). It is an ever-expanding and ever-relevant field in the modern era. With the widespread use and popularity of AI and servers as a service the Internet of Things is even more ingrained in human culture than ever before.

One of the major downsides of the Internet of Things is the potential pitfalls that may arise when an internet connection is lost or is not stable/fast enough to process incoming data. This pitfall is where TinyML could be used in order to enhance an IoT framework or structure. One way this can be done is by handling some or all of the data processing locally on a microcontroller thus bypassing the complete reliance and cost of doing the processing via the internet. Additionally, the chip itself can act as a fallback in case an internet connection is lost thus allowing for some redundancy in the system.

In application, this convergence between TinyML and the Internet of Things is a factor that is desirable to many and has great real-world applications. For companies, it can be a cost-saving measure that can greatly reduce the amount of data that they are sending over the internet for their servers to process. For a general device user, it can bring them the added security of either redundancy or a performance increase on their device.

The real-world applications of the Internet of Things and TinyML can be seen across almost every industry in the modern world. Airplanes need to process a high amount of data every second and use it in real-time; TinyML can reduce the data being sent over the internet thus improving the system. For general mobile phone users, TinyML can benefit their voice or AI searches as well as potentially improve battery life depending on the use case.

3 Related Work

3.1 TinyML Speech and Gesture Recognition

(V et al., 2022) have applied a tiny machine learning model for both speech and hand gesture recognition onto the Arduino Nano 33 BLE Sense microcontroller. Their model listens for very specific keywords to activate the onboard LEDs focusing primarily on their color. This is a good baseline and start to the research which will then expand upon this topic.

3.2 Real Time Voice Recognition

(Patel et al., 2023) have implemented a self-trained edge impulse keyword detection system with a high degree of accuracy. This self-trained model was then transferred to and used on a resource-constrained system, the Arduino Nano 33 BLE.

3.3 Speech Anger Detection System

(Waqar et al., 2021) uses a unique method of determining speech categorization that can potentially be expanded to other NLP models or projects. It also provides various chip capabilities and limitations. This allows for an application of speech recognition that categorizes levels of anger in speech and returns a visual reading.

3.4 Design of Custom Keyword Recognition

(Nived et al., 2023) train a model to listen for custom keywords by using edge impulse. They detail the training process and the specific custom keywords selected. Additionally, this research shows a potential method of recognizing more complex keywords provided proper model training.

3.5 AI Neural Networks Inference into the IoT Embedded Devices

(Toma et al., 2020) performs an in-depth look at the various structure elements behind the use and function of neural networks in TinyML. This research covers the tools and processes behind training and deploying a model to a TinyML device. Additionally, it references the internal logic needed to be performed by the chip in order to complete a specific function.

3.6 Introduction to the special issue on TinyML

A summary of the conference proceedings for the Association of Computing Machinery (ACM) adds some key insights into the current state of the art

regarding tinyML (Theocharides et al., 2024). This introduction to the idea and brief summary of the papers submitted helps to determine the current direction of TinyML and provides some good works to follow when exploring TinyML.

3.7 TinyML: Tools, Application, challenges, and future research

(Kallimani et al., 2024) compiles a comprehensive review of various sources relating to TinyML in 2023. This work provides a good background for popular topics in TinyML and a general overview of the more in-depth aspects of TinyML. It is a good general background baseline to either compare newer advancements to or to give general knowledge relating to a specific TinyML process.

3.8 TinyTS: Memory-Efficient TinyML Model Compiler Framework on Microcontrollers

(Liu et al., 2024) create a new DNN model compiler that focuses on memory efficiency and speed called TinyTS. The concepts used to make the model and the model itself will see use when compressing a trained model onto the Arduino Nano 33 BLE sense chip.

3.9 Implementation of a Speech-command-interface on Microcontroller with TinyML

(Pham, 2024) experiments with more complex command methods. The work is able to recognize 13 total commands and process them with an average accuracy of 80%. This research is closely related to applying more complex commands as it focuses on recognizing a larger command suit and provides some good potential direction for possible machine learning models and training data. Additionally, it explains the process mathematical and otherwise behind speech recognition. Published in 2024 this is a good direction for the research to follow and it provides a good testing standard to compare data to.

3.10 On-Device Domain Learning for Keyword Spotting on Low-Power Extreme Edge Embedded Systems

(Cioflan et al., 2024) explores a unique method of handling noise featuring on-device machine learning meant to adapt to background noise thus increasing accuracy over time. The noise handling method could potentially be adapted in order to increase the accuracy of predictions and readings.

3.11 Towards Contactless Elevators with TinyML using CNN-based Person Detection and Keyword Spotting

(Pimpalkar and Niture, 2024) explore an application of TinyML that involves person and keyword detection in an elevator use case. This research shows the different states of the elevator transferring from a non-listening mode to a listening mode after a given condition is met (a person is detected). Additionally, this offers some potential solutions to differentiating between states and can provide some insights into further training a custom speech model.

3.12 Voice Activated Edge Devices Using Tiny Machine Learning Enabled Microcontroller

(Uddin et al., 2024) explore another application of keyword detection, that being a smart fan. This research innovates by accommodating for noise specifically from the fan itself therefore increasing the accuracy of detection. This method of listening specifically for noise is a good potential adaptation to this research. The fan use case is also another applicable IoT use case for the results of the research.

3.13 TinySV: Speaker Verification in TinyML with On-device Learning

(Pavan et al., 2024) uses a speaker verification procedure along with keyword detection. This two-step process offers some insight into potentially syncing the two chips together so that they operate as one coherent unit. Additionally, it offers some unique approaches to model training that can contribute to the research.

3.14 Efficient Deep Learning Computing: From TinyML to LargeLM

(Lin, 2024) offers a more efficient approach to TinyML both training and inference in the thesis. Additionally, some common quantization and efficiency techniques are discussed in regards to TinyML that manages to maximize the efficiency and accuracy while also maximizing the available space on a TinyML device. This thesis manages to explain and provide a suite of tools that can contribute to the implementation of TinyML in a memory-efficient way.

3.15 Audio–visual keyword transformer for unconstrained sentence-level keyword spotting

(Li et al., 2024) provide a unique approach to speech based NLP. This research also provides some good potential test cases and scenarios to test the language models on in order to determine their accuracy. Some of these test cases will be used in order to identify the validity of the research.

3.16 Edge Impulse: An MLOps Platform for Tiny Machine Learning

(Hymel et al., 2023) explore Edge Impulse a software-as-a-service online tool that allows developers to simplify the machine learning pipeline for edge devices. This paper details a simplified process behind what edge impulse does as well as explain its uses in both academia and industry. Additionally, it goes into detail about the general process behind some of Edge Impulse's more important tools and processes. This paper provides a good baseline for Edge Impulse development, recreation, and use in industry and research.

3.17 1D convolutional neural networks and applications: A survey

(Kiranyaz et al., 2021) explores the concept and application of 1D convolutional neural networks. This paper details the process behind the network and its advantages those being that these networks have a much smaller complexity when compared to 2D CNNs and that they provide great results on relatively small datasets. This paper also discusses the need to use a DSP step before processing the data with the CNN as it helps the neural network extract features from the audio file more consistently and easily.

3.18 SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition

(Park et al., 2019) propose a state-of-the-art method of handling speech/audio data. In the context of machine learning, this involves doing some audio preprocessing in order to make it easier for a model to pull features out of an audio input. This method makes the model more accurate and drastically reduces the required dataset size for the model to function properly with a high accuracy. This is also the technique that edge impulse uses in order to preprocess data before sending it into a neural network.

4 Proposed Approach

An Arduino Nano 33 BLE sense chip will be used in order to process complex tasks and commands. The general logic of this process will be as follows. The chip will initially be in a 'sleep' mode and will be awoken with the input of the keyword 'Wake Up'. Once awoken the chip will be in a listening mode where it will wait and listen for a user's spoken commands. If nothing is said from this state the chip will transition back to the 'sleep' mode.

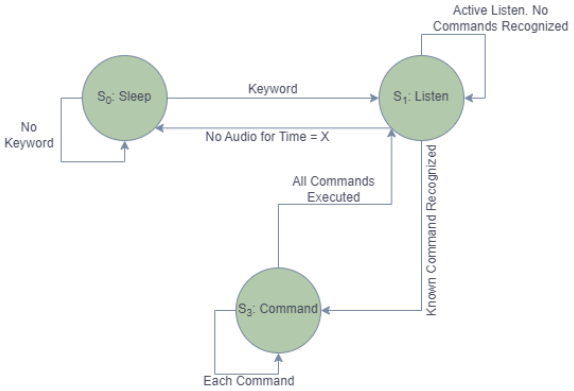


Figure 1: Chip State Logic

The chip and its functionality will be tied to the usage of the on-device LEDs and thus the recognized commands of the machine-learning models will relate to the LEDs and their colors. In order to process more complex commands with the chip the command interpretation will be split into a series of smaller keywords that the model is programmed to recognize. These keywords will then act as 'flags' for the program to recognize with some being required before a desired function is observed. The model and its training will be split in two for the initial creation of the machine learning model. One of these models will be trained on the LEDs colors and another will be trained on a desired list of commands that the chip should follow. For their initial creation, the models will each have their own custom dataset, and will also be tested separately to one another. This is done in order to have smaller subsets to work with that allow for the potential of using two chips that each individually host their respective model; as well as to allow for the option to retrieve only one of the models for reuse. For example, one may want to use the command model because it is relevant to their application, but they are not using a system that has the specific colors that the color model listens for.

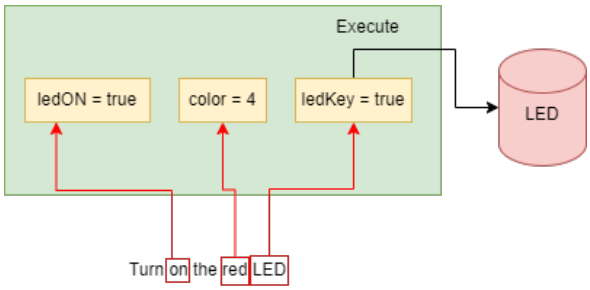


Figure 2: Text Interpretation Breakup

In order for an action to be taken by the chip the chip needs to hear keywords that activate the correct flags with the keyword 'LED' functioning as an execute command. When 'LED' is said the flags will be processed and if the correct flags are set an action will execute. These flags will each have their own priority of consideration during the execution phase.

The *Edge Impulse* (edg) platform will be used in order to train and deploy the model onto the chip. First, a model will be trained, deployed, and tested for the colors, and then a model will be trained, deployed, and tested for the commands. Afterward, the two datasets will be combined and the model will be retrained before deploying the final model to the chip. The edge impulse tool/framework has predefined optimizations for the Arduino Nano 33 BLE Sense chip as well as a variety of other hardware devices, allowing for cloud testing of the models before deployment, and for the potential to port the model to another chip.

5 Experimental Results

5.1 Experiment & Results

The initial starting point of the implementation of the proposed approach begins with a basic hardware test to ensure that the chips are functioning properly. To do this some basic tests of the Bluetooth functionality of the chip as well as some tests for the LEDs on the chip are needed. Additionally, this gives an indication of the projected speed of the Bluetooth connection between the two chips. This test is performed by following a basic guide given by Arduino (Bagur). The results of this test concluded with a confirmation that the low-energy Bluetooth connection is a viable and fast method to communicate between the chips and that two of the chip LEDs are programmable and easily usable. This showcases the IoT potential of the chip. Additionally, the system appears to be active LOW and the LED colors are slightly different than what is initially programmed into the chips header files for the color definitions. Finally, this step yielded the inference that the Micro USB cord connected to the chip needed to meet a standard that requires data transfer.

Next, a test of building the color keyword identifier model was attempted using *Edge Impulse*. This tool allows for easily creating test data, training the model, and compiling the model. Additionally, it allows for model retraining. Each of these features were explored tested and used in order to create a basic model that recognizes two color keywords. Each keyword is given around 33 one-second audio files containing the keyword which is split between training and testing 80-20. Edge Impulse automatically added its own general noise data in order to categorize background noise. From this, the basic functionality and capability of the Edge Impulse software environment were explored for its functionality, use, and features.

The model created on and by Edge Impulse was also tested for deployment. The basic test model was ported to an Arduino Library Zip file and executed as an 'example' in the Arduino IDE after importing it. This endeavor proved successful after properly selecting a quantized model to use on the chip. The functionality of this model consisted of constantly reading from the microphone on the chip and displaying the probability that the sound heard falls under one of four categories. Some experimentation was also done in regard to trying to modify the test model in order to change the functionality slightly. This concluded with the realization that the model should be trained on speech data pulled from the chip itself and that the response time of the current model is within an acceptable parameter.

The guides on Edge Impulse were ported over in order for the software to read directly from the chip. The first guide (Edg, a) followed a general overview of the process needed in order to connect to the chip. There were some issues found with the *flash_windows.bat* file that was provided in the guide which mainly centered around the *Arduino CLI 1.0* which the script was incorrectly labeling as an incorrect version. Additionally, the file was also trying to use *COM6* instead of *COM5* where the chip was actually connected. The *.bat* file was modified by using notebook++ in order to ignore the *Arduino CLI* version and select *COM5*. With these modifications, the first guide was able to be followed to completion without further issues. The *Edge Impulse CLI* tutorial (Edg, b) was also fol-

lowed as a prerequisite step for the previous guide. The only issues encountered in this step related to the length of the program's installation. The download of *node.js* and its required files is a long process thus a significant amount of time should be allocated before terminating the process.

The second chip test was performed using a combination of the knowledge base of the research to date. The previous model was modified in order to replace the audio keyword data shifting from the original audio (computer microphone) to the new audio source (Arduino Nano 33 BLE sense microphone). This model was kept simple thus only the keyword label "Red" was used for the new model. The training method differed from the previous model as the audio files had to manually be recorded and moved to either the training or testing datasets. The chip microphone recordings were made at 16000 Hz at an interval of 20 seconds before being split up with the assistance of tools. The new model has a total of 35 seconds of training data for the keyword red. Despite this data point increase over the previous model, the accuracy of the model suffers slightly from the drop in microphone quality and the manual audio split. When the model was uploaded to the chip and modified to use the logic of the previous test it was shown that the keyword 'Red' was being accurately listened to and processed. From this test it was concluded that this model only had successful readings when the keyword was spoken at the distance that it was trained at.

The third chip test involves the functionality of the chip itself rather than the machine learning models deployed on them. This experiment watches the function of the LEDs on the board itself and their associated colors. In this step it was found that the blue and green pins are reversed from the chip's regular assumed values from the Bluetooth example (Bagur). Additionally, the colors are active low as opposed to active high. So a value of 0 activates the color and a value of 255 deactivates the color. From this experiment it was found that there are 8 easily discernable colors that can be made from the LEDs which will serve as good keywords for the color model. There are more than 8 possible colors that the LED can display, but those colors will be adding unneeded complexity. After confirming the functionality of the on-board LEDs, the next process to be tested should be a two-way Bluetooth connection.

Developing a simple working prototype was the fifth test. For this test, the culmination of the results of the previous tests was used. The model was trained using a much larger collection of data for each keyword. Approximately one minute and thirty seconds for each keyword, and the training was done both at a moderate distance and a close distance to the speaker. The accuracy of this model has been inconsistent so far, showing poor results for red and good results for green. This suggests that even more thorough training data should be used for the process. Also, it was noted that there have been no issues with porting a model of this size to the Arduino Nano 33 BLE sense microcontroller which means that a larger audio dataset can acceptably be used to train the model for the purpose of improving accuracy.

The sixth test was an expansion of the previous prototype into a model that was closer to the desired use case. Eight color keywords were trained for the model before importing it to the chip. For this process, the previous method of recording training and testing audio at differing distances was used, and the testing and training split was done automatically by the software in order to attempt to reduce any human bias that might be observed via the split. It was observed from the previous tests that the microphone on the chip had its own static/background noise that was constantly present on each of the audio files. It was suspected that this was a major cause of the model's practical inaccuracy and the primary cause of the model incorrectly predicting certain keywords as a different keyword. In order to alleviate this potential pitfall the color model used in this test was trained on background noise that was present in some of the prior recordings. It was observed that this made a dramatic increase in the model's accuracy both in theory and practically. Additionally, there were some difficulties that were met when the initial model with the highest accuracy was selected as the digital signal processing method that was used for that model was one that excelled in non-human voices/sounds. A different DSP method was then selected that had a lower accuracy score of 95% but used the DSP model that was specifically designed for voices. When this new model was ported over to the device it saw a stronger keyword reading and accuracy in a practical sense than the previous test had managed to achieve. The model did at some points wrongfully categorize general speech unrelated to the keywords as one of the keywords, but for the majority of the words that the model was trained on an

LED relating to the keyword would turn on when the keyword was said. From this process, it was seen that training for the noise specifically generated by the chip's microphone drastically increased the accuracy of the model, the DSP (digital signal processing) method that was used also improved speech recognition and the model prototype would likely need some more tweaking and training in order to reach a more usable state.

The seventh test involved the exploration of the chip's low-power settings and its local interrupts. From this experiment it was determined that the chip currently does not have any libraries that support low power, and thus only has access to the sleep function. During this process, the method of creating a timer and using it for an interrupt was derived and used successfully. Unfortunately, the team was unable to identify a method of allowing for interrupts via the chip's microphone. Thus the actual use of the sleep function for the intended effect in the chip's logic was not able to be implemented. Instead, a process to simulate the sleep process will be used instead. Unlike the regular form of sleep this process will not have any power benefits but will instead process through an infinite loop that will look for a specific keyword before unlocking the continued function of the model.

Following this another test was performed changing the logic of the original color prototype; this new logic checks for a 60% or greater certainty with the model's classification before making the classification. This process observed a slight decrease in the improper classification of random speech, but it should be noted that the model also performed slightly worse at general classification. It is believed that this tradeoff between accuracy and correct classification is worth the implementation of such a system.

Using the previous methods learned and devised from the prior tests a dataset was created for the desired command keywords and combined with the color dataset in order to create a cohesive model that has strong functionality. The accuracy and performance of this model exceeded the individual accuracy of each dataset when kept separate. Using this new combined dataset the model will then need to be adapted in the code in order to handle the proposed processes and functionality.

For the final release of the model, the behavior of the chip using the predictions of the keyword model was implemented using a specific hierarchical structure where certain keywords took precedent before others in order to properly implement the desired functionality. In order to achieve this some of the problems with the model were exploited in order to provide a reliable way of recognizing when someone is speaking to the model. This is done by making use of the incorrect classification of general human speech as one of the non-noise keywords.

The results of this final model showed a high accuracy, and high F_1 scores across every keyword/label that the model can assign. With the worst F_1 score being 0.93, observed in the classification of the noise label that was autogenerated by edge impulse. The average F_1 score was: 0.98, the average precision was: 0.97, the average recall was: 0.98, and the accuracy was: 97.26%.

The dataset was created off of 1 hour 3 minutes and 31 seconds of data with each file taking up a 1-second interval. Each label has its own train and test split but they average out to an 80/20 split.

Table 1: F_1 scores per label

	F1-SCORE	PRECISION	RECALL
BLUE	0.96	0.96	0.96
CYAN	0.95	0.95	0.95
GREEN	1.00	1.00	1.00
LED	1.00	1.00	1.00
MAGENTA	0.98	1.00	0.95
OFF	0.98	1.00	0.96
ON	1.00	1.00	1.00
RED	1.00	1.00	1.00
WAKE UP	0.98	0.96	1.00
WHITE	0.95	1.00	0.91
YELLOW	1.00	1.00	1.00
AND	0.98	0.96	1.00
BLINK	1.00	1.00	1.00
CANCEL	1.00	1.00	1.00
FAST	1.00	1.00	1.00
FLASH	1.00	1.00	1.00
NOISE	0.93	0.91	0.96
NOISE2	0.94	0.97	0.92
PLUS	1.00	1.00	1.00
QUICK	1.00	1.00	1.00
SLOW	0.98	0.96	1.00
TOGGLE	1.00	1.00	1.00
UNKNOWN	0.98	1.00	0.96

6 Conclusion and Future Work

6.1 Conclusion

The chip performs well in a simulated environment, but its real-world application reveals room for improvement due to miscategorization issues. Despite this, the chip boasts high accuracy, recall, precision, and F1 scores. The functionality of the complex command model is desirable and useful on a real-world scale, showing promise for both specific use cases and general smart home or smart system applications. It enhances clarity for potentially ambiguous commands by generalizing certain keywords while restricting others. For example, "On" can refer to the state of many systems, but "LED" specifies a particular system to target. Additionally, the model requires minimal resources, making it suitable for deployment on more resource-constrained platforms than the Arduino Nano 33 BLE Sense.

7 Acknowledgement

All work herein reported is supported by the Nation Science Foundation under Grant No. 2349452.

Any opinion, finding, or conclusion in this study is that of the authors and does not necessarily reflect the views of the National Science Foundation.

References

a. [link].

b. [link].

Internet of Things - an overview | ScienceDirect Topics.

The leading edge ai platform.

Arduino Nano 33 BLE Sense. Arduino Nano 33 BLE Sense. https://store.arduino.cc/products/arduino-nano-33-ble-sense. Accessed: 2024-05-29.

José Bagur. Connecting nano 33 ble devices over bluetooth®.

Cristian Cioflan, Lukas Cavigelli, Manuele Rusci, Miguel de Prado, and Luca Benini. 2024. On-Device Domain Learning for Keyword Spotting on Low-Power Extreme Edge Embedded Systems. *arXiv preprint*. ArXiv:2403.10549 [cs, eess].

Lachit Dutta and Swapna Bharali. 2021. Tinyml meets iot: A comprehensive survey. *Internet of Things*, 16:100461.

Shawn Hymel, Colby Banbury, Daniel Situnayake, Alex Elium, Carl Ward, Mat Kelcey, Mathijs Baaijens, Mateusz Majchrzycki, Jenny Plunkett, David Tischler, Alessandro Grande, Louis Moreau, Dmitry Maslov, Artie Beavis, Jan Jongboom, and Vijay Janapa Reddi. 2023. Edge Impulse: An MLOps Platform for Tiny Machine Learning. *arXiv preprint*. ArXiv:2212.03332 [cs].

Dan Jurafsky and James H. Martin. 2024. Speech and language processing (3rd ed. draft) dan jurafsky and james h. martin.

Rakhee Kallimani, Krishna Pai, Prasoon Raghuwanshi, Sridhar Iyer, and Onel L. A. López. 2024. TinyML: Tools, applications, challenges, and future research directions. *Multimedia Tools and Applications*, 83(10):29015–29045.

Serkan Kiranyaz, Onur Avci, Osama Abdeljaber, Turker Ince, Moncef Gabbouj, and Daniel J. Inman. 2021. 1D convolutional neural networks and applications: A survey. *Mechanical Systems and Signal Processing*, 151:107398.

Yidi Li, Jiale Ren, Yawei Wang, Guoquan Wang, Xia Li, and Hong Liu. 2024. Audio–visual keyword transformer for unconstrained sentence-level keyword spotting. *CAAI Transactions on Intelligence Technology*, 9(1):142–152. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1049/cit2.12212.

Ji Lin. 2024. *Efficient Deep Learning Computing: From TinyML to LargeLM*. Thesis, Massachusetts Institute of Technology. Accepted: 2024-03-21T19:09:19Z.

Yu-Yuan Liu, Hong-Sheng Zheng, Yu Fang Hu, Chen-Fong Hsu, and Tsung Tai Yeh. 2024. TinyTS: Memory-Efficient TinyML Model Compiler Framework on Microcontrollers. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 848–860. ISSN: 2378-203X.

Boreda Vamshi Nived, K. Jamal, Guguloth Mahesh, and Rathod Mukesh Kumar. 2023. Design of custom keyword recognition using edge impulse on arduino nano 33 ble sense. In *2023 2nd International Conference on Applied Artificial Intelligence and Computing (ICAAIC)*, pages 1522–1529.

Daniel S. Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin D. Cubuk, and Quoc V. Le. 2019. SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition. In *Interspeech 2019*, pages 2613–2617. ArXiv:1904.08779 [cs, eess, stat].

Parth Patel, Nikhil Gupta, and Sachin Gajjar. 2023. Real Time Voice Recognition System using Tiny ML on Arduino Nano 33 BLE *. In *2023 IEEE International Symposium on Smart Electronic Systems (iSES)*, pages 385–388. ISSN: 2832-3602.

Massimo Pavan, Gioele Mombelli, Francesco Sinacori, and Manuel Roveri. 2024. TinySV: Speaker Verification in TinyML with On-device Learning. *arXiv preprint*. ArXiv:2406.01655 [cs, eess].

Duy Anh Pham. 2024. *Implementation of a Speech-command-interface on Microcontroller with TinyML*. Thesis, Hochschule für Angewandte Wissenschaften Hamburg. Accepted: 2024-04-05T08:15:22Z.

Anway S. Pimpalkar and Deeplaxmi V. Niture. 2024. Towards Contactless Elevators with TinyML using CNN-based Person Detection and Keyword Spotting. *arXiv preprint*. ArXiv:2405.13051 [cs].

Partha Pratim Ray. 2022. A review on TinyML: State-of-the-art and prospects. *Journal of King Saud University - Computer and Information Sciences*, 34(4):1595–1623.

TensorFlow Blog. 2020. What's new in TensorFlow Lite for NLP. https://blog.tensorflow.org/2020/09/whats-new-in-tensorflow-lite-for-nlp.html. Accessed: 2024-05-29.

TensorFlow Lite. TensorFlow Lite. https://www.tensorflow.org/lite/guide. Accessed: 2024-05-29.

TensorFlow Lite for Microcontrollers. TensorFlow Lite for Microcontrollers. https://www.tensorflow.org/lite/microcontrollers. Accessed: 2024-05-29.

Theocharis Theocharides, Charlotte Frenkel, and Lukas Cavigelli. 2024. Introduction to the Special Issue on tinyML. *ACM Transactions on Embedded Computing Systems*, 23(3):40:1–40:5.

Cristian Toma, Marius Popa, and Mihai Doinea. 2020. Ai neural networks inference into the iot embedded devices using tinyml for pattern detection within a security system. In *International Conference on Informatics in Economy Education, Research and Business Technologies*, pages 14–22.

Md Raihan Uddin, Abu Asaduzzaman, Khanh Le, and Rohit R. Medarametla. 2024. Voice Activated Edge Devices Using Tiny Machine Learning Enabled Microcontroller. In *2024 IEEE Green Technologies Conference (GreenTech)*, pages 38–42. ISSN: 2166-5478.

Viswanatha V, Ramachandra A. C, Raghavendra Prasanna R, Prem Chowdary Kakarla, Viveka Simha PJ, and Nishant Mohan. 2022. Implementation of Tiny Machine Learning Models on Arduino 33 – BLE for Gesture and Speech Recognition.

Dania Maryam Waqar, Teddy Surya Gunawan, Malik Arman Morshidi, and Mira Kartiwi. 2021. Design of a Speech Anger Recognition System on Arduino Nano 33 BLE Sense. In *2021 IEEE 7th International Conference on Smart Instrumentation, Measurement and Applications (ICSIMA)*, pages 64–69. ISSN: 2640-6535.

Exploring Perplexity Defense on Adversarial Language Model Inputs

Ainslee Archibald
Pitzer College
aarchib@pitzer.edu

Jeno Saghi & Jugal Kalita
University of Colorado Colorado Springs
{esaghi, jkalita}@uccs.edu

Abstract

Robustness in language models, specifically in defense against adversarial textual attacks, is an open area of research. Identifying and mitigating adversarial text inputs requires innovative approaches and flexibility to handle different attack settings. Perplexity-based masking and substitution schemes have been an active frontier in defense tactics, but more can be done to adjust and add to the systems in order to counter more contemporary attacks. This study investigates a novel adversarial defense method through perplexity-based identification techniques with mask-and-refill approaches and other concepts from contemporary defense frameworks. Our study tests this omnibus method on classification tasks for the SST-2 and AG News datasets. Despite theoretical backing, this approach did not reliably yield success in empirical testing. We analyze why our perplexity-based approach faced challenges and what can be done to guide future research.

Code, datasets, and full results are available on GitHub at: https://github.com/ains-arch/ppl-defense

1 Introduction

Natural language processing aims to teach computers to understand and use language. Unfortunately, computers can't really "understand" anything beyond mathematical representations of language. Thus, it is deceptively easy to fool a unprotected language model by making small changes to its inputs. Strategies to intentionally perturb the input data of an AI model to cause incorrect or unintended outputs are called adversarial attacks (Liu and Hu, 2024; Goyal et al., 2023).

With recent increased interest in adversarial attacks on language models, there has also been much development in adversarial defense, which aims to protect language models and increase their robustness to a wide array of attacks. Current approaches include additional training on adversarial inputs, changing model architecture to increase robustness, and adding spell-checking (Goyal et al., 2023).

Perplexity-based approaches leverage the assumption that manipulated text will be distinguishable from benign text through analyzing its perplexity, allowing the suspicious inputs to be replaced or removed altogether.

Perplexity is an inverse measure of probability of a sequence, normalized by the number of words (Jelinek et al., 2005). It is defined for a sequence $w_1 w_2 \dots w_N$ as

$$\begin{aligned}\text{perplexity}(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}}.\end{aligned}$$

Perplexity is a very common metric in measuring a model's ability to generate human-like text. Despite its prevalence, some studies investigate other ways of evaluating language models, implying that other avenues for sanitizing text inputs may be fruitful (Meister and Cotterell, 2021). However, perplexity remains a common statistic in both model evaluation and adversarial defense. Despite this, current perplexity-based approaches in adversarial defense have either not handled replacement (Qi et al., 2021) or have used other metrics to select the tokens for replacement (Ge et al., 2023).

Our approach uses perplexity on both parts of the pipeline: one, to select suspicious tokens in input data for removal, and two, to select suspicious tokens in input data for replacement. Our results unexpectedly showed that this omnibus method was not reliably more effective than simply using perplexity as a measure for removal, providing insight into potential pitfalls in over-reliance on perplexity as a metric.

2 Related Works

There has been much research in natural language processing with the intention of misleading language models through adversarial attacks. Many approaches, including perplexity-based methods, have been developed to defend against these attacks by enhancing model robustness.

2.1 Adversarial attacks

The distorted inputs used in adversarial attacks are designed to affect how the model reacts without rising to the level of human detection (Goyal et al., 2023). Attacks on computer vision systems often involve adjustments to images that would be difficult for a human to notice without further analysis. In natural language processing, adversarial attacks typically alter text data at the word, character, or sentence levels (Goyal et al., 2023).

The literature on adversarial attacks can also be subdivided between backdoor attacks and data poisoning (Liu and Hu, 2024). Backdoor attacks in language models involve introducing triggers that cause malicious behavior while allowing the model to perform normally otherwise (Nguyen et al., 2024). This can be done during training to cause systemic failures, or during inference to cause targeted failures in responses to specific prompts. Data poisoning manipulates the training data to introduce decision-making errors, and thus takes place in the training phase (Liu and Hu, 2024). This is especially a concern for large language models that are pretrained on huge datasets of unverified data from public sources.

Attacks across character, word, and sentence levels are implemented in the OpenAttack framework (Zeng et al., 2021).

2.2 Adversarial defense

Progress has been made in defense using adversarial training, changing model architecture, and spell-checking add-ons (Goyal et al., 2023). However, approaches that first identify the adversarial input and then mitigate it through replacement or outright removal can avoid costly retraining or significant change to models.

Inspired by methods to increase robustness in computer vision settings, randomness can be used to catch the adversarial inputs or decrease their effectiveness (Swenor and Kalita, 2022; Zeng et al., 2023; Gietz and Kalita, 2024).

A more common approach is to identify suspicious tokens through analysis. Qi et al. (2021) introduced a method to handle textual backdoor attacks called ONION, which selects trigger words by analyzing the perplexity of sentences with and without each word, and removes any words with a high enough difference. This idea was also developed by Ge et al. (2023) as part of a two step process called the UMPS defense to first fill in out-of-vocabulary words (to handle character-level attacks) and then use perplexity sampling to find synonyms (to replace word-level attacks) (Ge et al., 2023).

Other approaches include certifying robustness through mathematical means (Jia et al., 2019; Levine and Feizi, 2019; Zeng et al., 2023; Gietz and Kalita, 2024), merging multiple models to decrease the effect of a single backdoor (Arora et al., 2024), and developing weighted suites of defense paradigms (Xiang et al., 2024).

3 Problem Formulation and Algorithm Design

While research has been done on using perplexity calculations as a means of adversarial input identification and removal for character-level attacks and, separately, on using perplexity as a way to select synonyms to combat word substitution attacks, no approach has tried to combine both perplexity measures into a single approach that can be robust against both character perturbation and word substitution. As perplexity provides a rigorous way to quantify the unlikeliness of a certain sequence, this study applies it as much as possible in character- and word-level adversarial defense.

Key to our approach is avoiding costly adversarial training and maintaining no knowledge of the attack type in the target language model. Instead, all the attacks our pipeline is tested against are score-based. Score-based attacks craft adversarial perturbations based on repeatedly querying the target model. They require no knowledge of the dataset or additional model training. They only require the outputs of the classifier to be probabilities (Wang et al., 2022). Our pipeline requires no information about the attack it is defending against in order to run.

We use BERT as the target model for our classification tasks (Devlin et al., 2019). Bi-directional models like BERT do not have a well-defined measure of perplexity over sequences. One approach

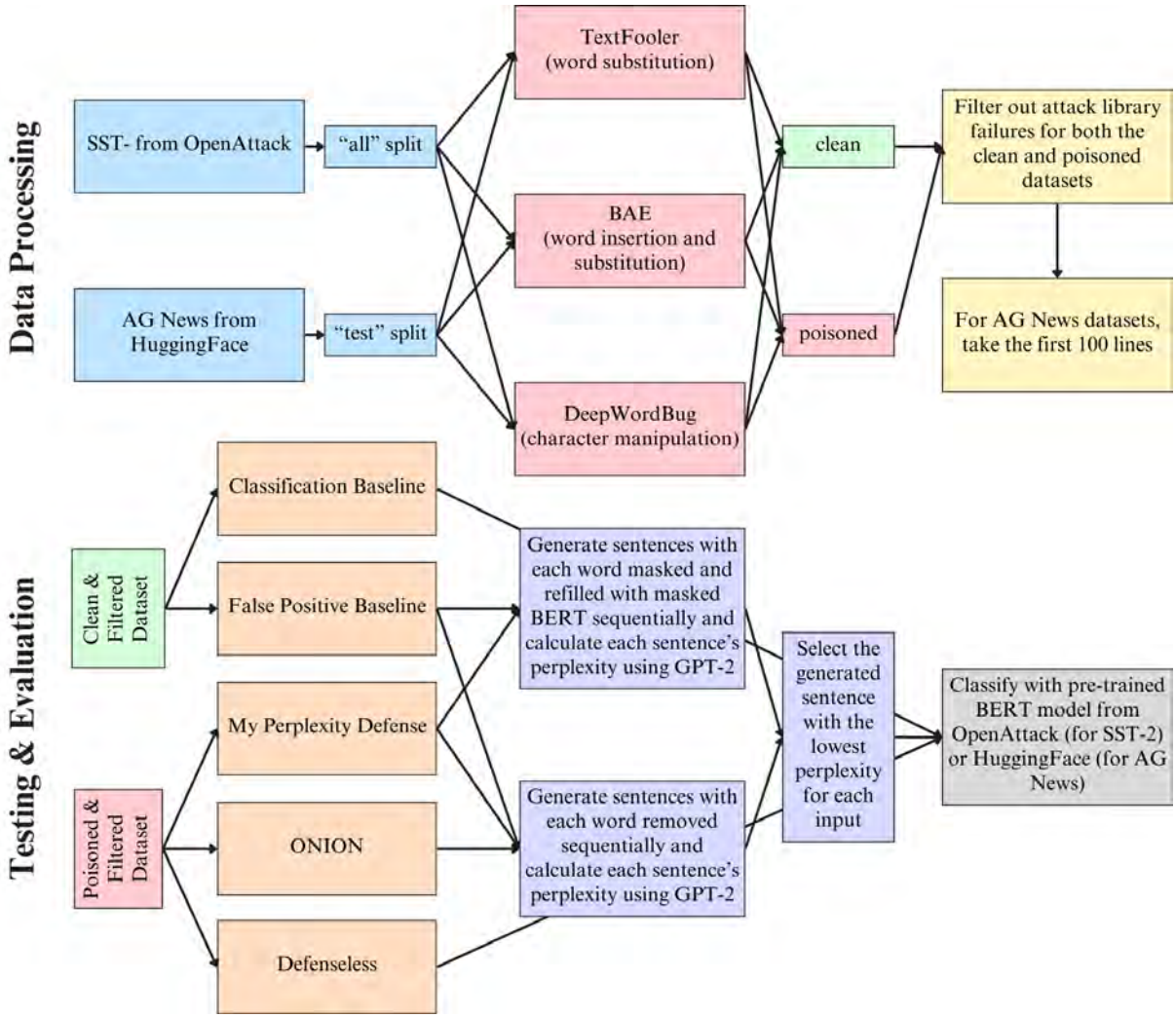


Figure 1: Data processing and defense pipeline

used by Miaschi et al. (2021) masks each word and calculate the probability

$$p(W) \approx p(w_i \mid \text{context}),$$

where the context is

$$w_1, \ldots, w_{i-1}, w_{i+1}, \ldots, w_k$$

to then calculate a version of perplexity

$$PPL_W = e^{p(W)}.$$

However, we instead relied on a GPT-2 model to calculate perplexity. While this abstracts the meaning of perplexity away somewhat, as it outsources it to another model rather than pulling it from the classifier model, both GPT-2 and BERT are sufficiently large models to contain understanding of the English language such that the perplexity calculation from one is applicable to classification tasks in the other (Miaschi et al., 2021).

3.1 Algorithm

Our method is focused on inference-stage adversarial input. Thus, it operates through sitting in between the BERT classification model and the poisoned data, sanitizing inputs so that the BERT model can more accurately classify the sentences. For the full explanation, see Algorithm 1 and Figure 1.

4 Experiments

To implement attacks and test our framework, we use the OpenAttack framework (Zeng et al., 2021). Specifically, we use its implementation of TextFooler, BAE, and DeepWordBug. TextFooler is a word-level substitution attack (Jin et al., 2020). BAE, or BERT-based Adversarial Examples, is a word substitution and insertion attack (Garg and Ramakrishnan, 2020). DeepWordBug is a character substitution attack (Gao et al., 2018).

Algorithm 1 Adversarial Defense Algorithm

Input: Original sentence S
Output: Classification of the sentence
Generate an adversarial attack on S
Calculate perplexities for word removal as inspired by ONION (Qi et al., 2021):
for each word w_i in S **do**
    Remove w_i from S to create S_i
    Calculate perplexity of S_i using GPT-2
end for
Calculate perplexities for word mask-and-replace as inspired by UMPS (Ge et al., 2023):
for each word w_j in S **do**
    Replace w_j with `[MASK]` in S to create $S_{j\text{-}masked}$
    Refill `[MASK]` with $w_{j\text{-}BERT}$ using BERT base model (uncased) to create $S_{j\text{-}refilled}$
    Calculate perplexity of $S_{j\text{-}refilled}$ using GPT-2
end for
Classify the original input:
Choose $S_{\text{lowest-ppl}}$ with the lowest perplexity from either method
Classify $S_{\text{lowest-ppl}}$ using BERT model pre-trained on dataset
Return Classification result

As a victim model for our classification task for the SST dataset, we use a pre-trained BERT model that comes packaged with OpenAttack. As a victim model for our classification task for the AG News dataset, we use a pre-trained BERT model from HuggingFace.

Additionally, we use GPT-2 for perplexity calculations and the BERT base model for the masking and re-filling tasks. GPT-2 and the BERT base model are both loaded from the HuggingFace transformers library.

Also packaged with the OpenAttack library is an easy way to load the SST-2 dataset (Socher et al., 2013), which is the dataset we use for sentiment analysis. We loaded the AG News dataset from HuggingFace, which we used for topic modeling.

The accuracy for this method was calculated on the entire SST-2 dataset and the first 100 entries in the correctly attacked test split of the AG News dataset for time consideration reasons. However, because both of these pre-trained BERT models were presumably trained on the data we test on, the true accuracy of the models on data they haven't seen may be lower. Thus, it is more important to compare the defenses to each other rather than looking at a single accuracy percent alone.

5 Results

Our perplexity pipeline did worse than the ONION defense on the SST-2 dataset, but better than ONION on AG News. In most cases, the defenses both did better than classification on poisoned data without a defense, but worse than the accuracy without an attack or a defense. Additionally, the perplexity pipeline did not significantly decrease accuracy between classification under no attack or defense when run on a clean dataset, implying it doesn't overly sanitize input. See Table 1 for the full results.

The results come from one run of the pipelines on the dataset. Multiple runs are not necessary for a setup like this, as there is no randomness involved or model training. The pipelines are deterministic and would return the same accuracy.

The classification baselines are different for each attack because each attack in the OpenAttack library fails on some percentage of the input sentences and returns nothing. This was mitigated in our pipelines by running the attack on the dataset and then filtering out sentences that didn't return an attack from both datasets. As this created three separate datasets, the classification baseline that would otherwise be the same varies somewhat between attacks.

We evaluate the results (Table 1) of our model's performance through comparing its robustness under attack with the ONION defense (Qi et al.,

Attacks	Defense	Datasets	
		SST-2	AG News
TextFooler	Perplexity pipeline	.4498	**.96**
	ONION	**.5239**	.89
	Attack without defense	.3012*	.96
	Classification baseline	.9033*	.97
	Pipeline on unpoisoned dataset	.8931***	.95
BAE	Perplexity pipeline	.4061	**.97**
	ONION	**.4822**	.96
	Attack without defense	.1954	.97
	Classification baseline	.8046	.97
	Pipeline on unpoisoned dataset	.7837	.96
DeepWordBug	Perplexity pipeline	.1398**	**.96**
	ONION	**.2876**	.89
	Attack without defense	.1326	.96
	Classification baseline	.8674	.97
	Pipeline on unpoisoned dataset	.9116**	.95

Table 1: Comparison of Different Attacks on Various Datasets

2021). As baselines, we also compare against the BERT model's accuracy under attack with no defense, its accuracy with no attack or defense, and to test the pipeline's tendency toward false positives, the defense on unpoisoned data.

Across the board, the ONION defense does better on the SST-2 dataset, and our perplexity defense does better on the AG News dataset. The main difference between the two datasets is the length of each input sentence, which may affect model accuracy as discussed below. Because the defense were only tested on a small portion of the AG News dataset, further testing and investigation is necessary in order to more concretely determine why the defenses' accuracies differ.

5.1 Discussion

The difference in accuracy between our perplexity pipeline and the ONION defense was greater in the TextFooler and DeepWordBug attacks than in the BAE attack on the AG News dataset. This follows our intuition that mask-and-replace approaches would be better suited to word substitution and character manipulation attacks than just removal. BAE does both word substitution and word insertion, and DeepWordBug does character manipulation, and ONION was less reliably able to remove the substituted words and words with substituted characters while maintaining the classification of the sentence.

However, the pipeline did worse than ONION on the SST-2 dataset. One possible explanation for this is the length of inputs in SST-2 are shorter on average than the AG News dataset. Our pipeline generates a new sentence for each word in the input for each of the removal and replacement approaches. Conversely, ONION cycles through 100 settings of a hyperparameter and generates new sentences for all of them. Thus, in a dataset with shorter inputs, the ONION approach may have an advantage because it has 100 "chances" to get the best and lowest perplexity sentence. It may be possible to correct for this by limiting ONION to only choose the hyperparameter as many times as our pipeline generates sentences, or through adjusting the voting functions to limit this advantage.

Perplexity-based defense is a fruitful area of research, and this work shows other potential avenues to investigate how to better handle more sophisticated adversarial attacks. The most important future research directions involve solidifying our results through more extensive testing on larger and more diverse datasets, and if necessary, iterating on this pipeline in order to more reliably bring the results above the baselines. There are many possible avenues for making improvements, and we detail some below.

The pipeline uses GPT-2 to calculate the perplexity of the sanitized sentences. However, we

* slightly different codebase used
** preliminary results on $\tilde{4}0\%$ of the dataset
*** preliminary results on $\tilde{8}8\%$ of the dataset

are using a BERT model to classify the sentences, and to perform the mask and refill task. Therefore, it would be sensible to investigate using something other than GPT-2 to calculate perplexity, including possibly an implementation of the BERT perplexity concept mentioned previously (Miaschi et al., 2021).

Once the perplexity calculations are made, another possible improvement could come in investigating other ways to weight the perplexity-based removals and replacements. We tested taking a majority vote over the five lowest-perplexity sentences. However, we found that using the classification of just the one top lowest-perplexity sentence was more accurate. There may be a better choice for the hyperparameter than 1 or 5, and testing that would be beneficial. Another next step could be to consider the effectiveness of instead taking a 50/50 split of the two approaches and a straight perplexity weight. In the long term, a more rigorous approach to this weight could come in the form of substituting our voting function for a classifier trained on which approach to prioritize given different inputs.

To further justify our results and to provide greater context, more datasets and attacks could be used with the pipeline. Most relevant to perplexity-based approaches is the attack introduced by Du et al. (2024) that employs natural word substitution (NWS). By creating a diverse synonym thesaurus, NWS avoids perplexity-based defenses that detect insertion attacks. A next step could be to implement a version of the NWS attack to test our defense against (Du et al., 2024), and to implement other more contemporary attacks, as the OpenAttack library only includes attacks up through 2020. TextAttack is one possible option for a different attack library, as it is more actively maintained and has more recent attacks (Morris et al., 2020).

For more datasets, we suggest testing on the Twitter dataset for toxicity detection (Founta et al., 2018). The Twitter dataset, together with SST-2 and AG News, would match the set used to test the NWS attack, which both suggests that they're reliable choices to test adversarial attacks and defense against, and if the NWS attack was implemented, would make an evaluation of the effectiveness of our defense most direct (Du et al., 2024).

To have more defenses to compare our approach to, it would be ideal to implement a version of the UMPS defense (Ge et al., 2023). This would allow us to understand if our merged perplexity defense improves on either of its two main inspirations, or if it under-performs both. Additional comparisons can come from a state-of-the-art approach that doesn't involve perplexity, and an approach previously implemented at this lab.

More mundane improvements to the research would come in the form of improving the codebase to make use of parallel processing where feasible, and to narrow down exactly what is causing pervasive errors in the attack library implementations in OpenAttack. Depending on the dataset and attack, around 30% of the input sentences fail to return an adversarial attack sentence. For our results, this was handled by simply not removing these sentences from the dataset. However, more investigation is needed on the root cause. Other extensions include investigating certified robustness and generalizing to sentence level attacks.

6 Conclusion

Perplexity-based identification, removal, and replacement schemes have potential in mitigating adversarial attacks in NLP settings. However, our method of combining ways of considering perplexity at as many stages of the process as possible was ultimately unable to reliably improve upon existing approaches in its current form. These results, while inconclusive, suggest that there is some merit to our approach, and can help guide future research by narrowing down how perplexity should and should not be applied in adversarial defense.

7 Acknowledgements

All work herein reported is supported by the Nation Science Foundation under Grant No. 2349452.

Any opinion, finding, or conclusion in this study is that of the authors and does not necessarily reflect the views of the National Science Foundation.

The authors would like to thank Harrison Gietz, Ali Al Shami, Melhamu Mersha, Ahmed Bensaoud, and Joe Worsham for contributing ideas and Ethan Capehart and Elise Clark for reviewing code.

References

Ansh Arora, Xuanli He, Maximilian Mozes, Srinibas Swain, Mark Dras, and Qiongkai Xu. 2024. Here's a Free Lunch: Sanitizing Backdoored Models with Model Merge. ArXiv:2402.19334 [cs].

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Wei Du, TongXin Yuan, HaoDong Zhao, and Gong-Shen Liu. 2024. NWS: Natural Textual Backdoor Attacks Via Word Substitution. In *ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4680–4684. ISSN: 2379-190X.

Antigoni Founta, Constantinos Djouvas, Despoina Chatzakou, Ilias Leontiadis, Jeremy Blackburn, Gianluca Stringhini, Athena Vakali, Michael Sirivianos, and Nicolas Kourtellis. 2018. Large Scale Crowdsourcing and Characterization of Twitter Abusive Behavior. *Proceedings of the International AAAI Conference on Web and Social Media*, 12(1). Number: 1.

Ji Gao, Jack Lanchantin, Mary Lou Soffa, and Yanjun Qi. 2018. Black-box Generation of Adversarial Text Sequences to Evade Deep Learning Classifiers. ArXiv:1801.04354 [cs].

Siddhant Garg and Goutham Ramakrishnan. 2020. BAE: BERT-based Adversarial Examples for Text Classification. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6174–6181. ArXiv:2004.01970 [cs].

Zhaocheng Ge, Hanping Hu, and Tengfei Zhao. 2023. Towards Trustworthy NLP: An Adversarial Robustness Enhancement Based on Perplexity Difference. In *European Conference on Artificial Intelligence 2023*, pages 803–810. IOS Press.

Harrison Gietz and Jugal Kalita. 2024. MaskPure: Improving Defense Against Text Adversaries with Stochastic Purification. In *The 29th International Conference on Natural Language & Information Systems*.

Shreya Goyal, Sumanth Doddapaneni, Mitesh M. Khapra, and Balaraman Ravindran. 2023. A Survey of Adversarial Defenses and Robustness in NLP. *ACM Computing Surveys*, 55(14s):332:1–332:39.

F. Jelinek, R. L. Mercer, L. R. Bahl, and J. K. Baker. 2005. Perplexity—a measure of the difficulty of speech recognition tasks. *The Journal of the Acoustical Society of America*, 62(S1):S63.

Robin Jia, Aditi Raghunathan, Kerem Göksel, and Percy Liang. 2019. Certified Robustness to Adversarial Word Substitutions. ArXiv:1909.00986 [cs].

Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. 2020. Is BERT Really Robust? A Strong Baseline for Natural Language Attack on Text Classification and Entailment. ArXiv:1907.11932 [cs].

Alexander Levine and Soheil Feizi. 2019. Robustness Certificates for Sparse Adversarial Attacks by Randomized Ablation. ArXiv:1911.09272 [cs, stat].

Frank Weizhen Liu and Chenhui Hu. 2024. Exploring Vulnerabilities and Protections in Large Language Models: A Survey. ArXiv:2406.00240 [cs].

Clara Meister and Ryan Cotterell. 2021. Language Model Evaluation Beyond Perplexity. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5328–5339, Online. Association for Computational Linguistics.

Alessio Miaschi, Dominique Brunato, Felice Dell'Orletta, and Giulia Venturi. 2021. What Makes My Model Perplexed? A Linguistic Investigation on Neural Language Models Perplexity. In *Proceedings of Deep Learning Inside Out (DeeLIO): The 2nd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, pages 40–47, Online. Association for Computational Linguistics.

John X. Morris, Eli Lifland, Jin Yong Yoo, Jake Grigsby, Di Jin, and Yanjun Qi. 2020. TextAttack: A Framework for Adversarial Attacks, Data Augmentation, and Adversarial Training in NLP. ArXiv:2005.05909 [cs].

Thuy Dung Nguyen, Tuan Nguyen, Phi Le Nguyen, Hieu H. Pham, Khoa D. Doan, and Kok-Seng Wong. 2024. Backdoor attacks and defenses in federated learning: Survey, challenges and future research directions. *Engineering Applications of Artificial Intelligence*, 127:107166.

Fanchao Qi, Yangyi Chen, Mukai Li, Yuan Yao, Zhiyuan Liu, and Maosong Sun. 2021. ONION: A Simple and Effective Defense Against Textual Backdoor Attacks. ArXiv:2011.10369 [cs].

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642.

Abigail Swenor and Jugal Kalita. 2022. Using Random Perturbations to Mitigate Adversarial Attacks on Sentiment Analysis Models. ArXiv:2202.05758 [cs].

Chenxu Wang, Ming Zhang, Jinjing Zhao, and Xiaohui Kuang. 2022. Black-Box Adversarial Attacks on Deep Neural Networks: A Survey. In *2022 4th International Conference on Data Intelligence and Security (ICDIS)*, pages 88–93.

Tao Xiang, Fei Ouyang, Di Zhang, Chunlong Xie, and Hao Wang. 2024. NLPSweep: A comprehensive defense scheme for mitigating NLP backdoor attacks. *Information Sciences*, 661:120176.

Guoyang Zeng, Fanchao Qi, Qianrui Zhou, Tingji Zhang, Zixian Ma, Bairu Hou, Yuan Zang, Zhiyuan Liu, and Maosong Sun. 2021. OpenAttack: An Open-source Textual Adversarial Attack Toolkit. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*, pages 363–371, Online. Association for Computational Linguistics.

Jiehang Zeng, Jianhan Xu, Xiaoqing Zheng, and Xuanjing Huang. 2023. Certified Robustness to Text Adversarial Attacks by Randomized [MASK]. *Computational Linguistics*, 49(2):395–427.

Linear Decoding of Morphology Relations in Language Models

Eric Xia
Brown University
eric_xia@brown.edu

Jugal Kalita
University of Colorado Colorado Springs
jkalita@uccs.edu

Abstract

The recent success of transformer language models owes much to their conversational fluency and productivity in linguistic and morphological aspects. An affine Taylor approximation has been found to be a good approximation for transformer computations over certain factual and encyclopedic relations. We show that the truly linear approximation $W\mathbf{s}$, where $\mathbf{s}$ is a middle layer representation of the base form and W is a local model derivative, is necessary and sufficient to approximate *morphological derivations*. This approach achieves above 80% faithfulness across most morphological tasks in the Bigger Analogy Test Set. We argue that morphological forms in transformer models are likely to be encoded by linear transformations, with implications for how entities are represented.

1 Introduction

Large language models display impressive capabilities for factual recall, which commonly involve relations between entities (Brown et al., 2020). Recent work has shown that affine transformations on subject representations can faithfully approximate model outputs for certain subject-object relations (Hernandez et al., 2023). Identifying the contexts in which approximations perform well is an important area of study, with applications in interpretability and model editing.

Work to date around relational representation in LMs have primarily focused on relations in the context of factual subjects and objects (Meng et al., 2022), (Hernandez et al., 2023), (Chanin et al., 2023). However, relations in natural language encompass a much broader range of subject and object relations. Much of the mainstream success of LLMs has been due to the conversational nature of chat-oriented language models. The impressive conversational ability of LLMs depends on their linguistic competency, including lexical and morphological productivity, and uncovering how models are able to achieve this is an important aspect of model interpretability.

We demonstrate that for morphological relations, a transformation from the Jacobian alone is able to approximate object computations from an enriched subject exceptionally well. This suggests that transformers encode morphology linearly, in an even simpler fashion than the affine LRE discovered by Hernandez (2023). Linearly approximating the computation from an early hidden state of the base form to the final state of the derived form, we find that approximable relations include pluralization, nominalization, changes in tense, and resultative forms. These derivations range over different parts of speech, including noun to adjective **[noun+less]**, adjective to noun **[adj+ness]**, verb to noun **[verb+er]**, and verb to adjective **[verb+able]**, and involve diverse subjects and objects.

By linearly decoding linguistic relations in this manner, we offer an interpretation which reveals how internal ontological atomic representations, such as those espoused in the Linear Representational Hypothesis and concept theory (Park et al., 2023), (Wang et al., 2023), (Park et al., 2024), could be unfolded by a LM to encompass a range of morphological variations. We show that relational approximation in LMs can be applied to a broad range of linguistic phenomena, opening avenues for further research in model representations. To the best of our knowledge,

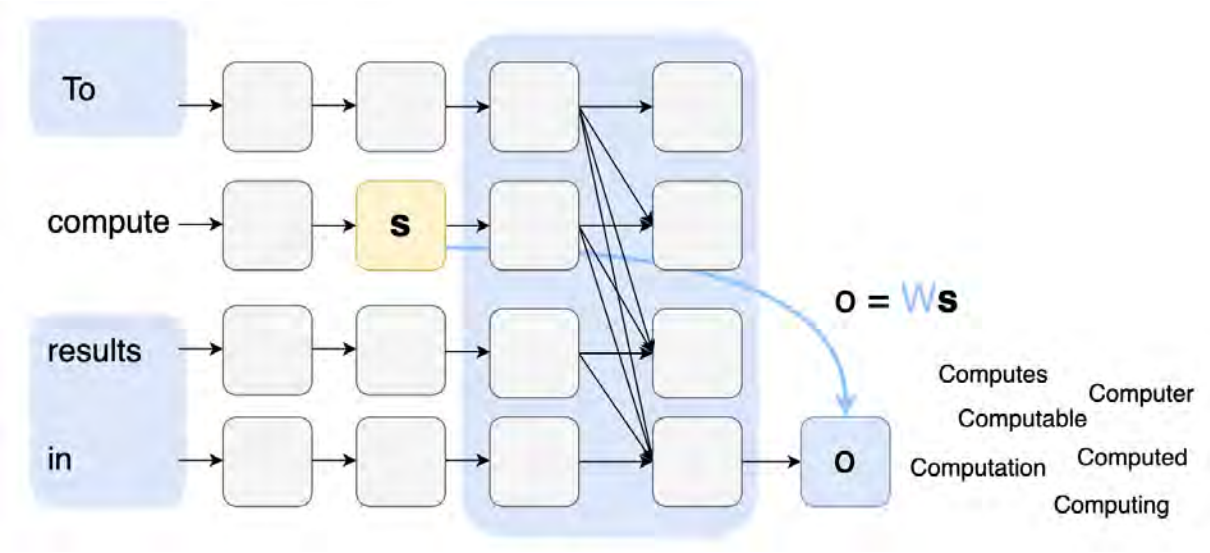


Figure 1: Adapting morphological analogies from the Bigger Analogy Test Set to relational contexts reveals that many are genuinely linearly approximable, such as **[verb+tion]**, **[verb+able]**, **[noun - plural]**, **[verb_inf]**, and **[verb+er].**

we are the first to provide empirical evidence of a linear transformation acting as an effective LM approximator over a broad range of inputs.

To understand affine approximation better, we also analyze linear projections of the LRE. We find that the Jacobian increases the geometric similarity of the subject and object spaces, while the bias contributes the majority of the movement. We confirm the previous hypothesis that a beta parameter is necessary to reproduce the output space. We find that approximation accuracy can be measured by embedding variance, and that linear representation of final layer embeddings is an effective way to diagnose relations which are not affinely approximable.

2 Background & Related Work

2.1 Transformer Computation

In autoregressive transformer language models, input text is converted to a sequence of tokens $t_1 \dots t_n$, which are subsequently embedded as $x_1 \dots x_n \in \mathbb{R}^d$ by an embedding matrix. The hidden states $x_1 \dots x_n$ are then passed through L transformer layers, each composed of a self-attention layer a^l and an multi-layer perceptron (MLP) layer m^l, and then decoded by the decoder head D to a probability distribution over tokens. The representation state x_i^l of the i^{th} token at layer l is obtained as:

$$x_i^l = x_i^{l-1} + a_i^l + m_i^l$$

Where a_i^l and m_i^l are

$$a_i^l = \text{attn}^l\left(x_1^{l-1}, x_2^{l-1}, \dots, x_i^{l-1}\right)$$
$$m_i^l = W_{out}^l\, \sigma\left(W_{in}^l(a_i^l + x_i^{l-1})\right)$$

Here, attn^l is multiheaded Key-Value Query attention as described in Vaswani et al. (2017), W_{in}^l and W_{out}^l are projection matrices, and σ is a nonlinear activation function. In GPT-J, the output of the l^{th} MLP sublayer for the i^{th} representation depends only on x_i^{l-1} rather than $a_i^l + x_i^{l-1}$, so the attention and MLP modules operate in parallel (Wang and Komatsuzaki, 2021).

Following the insights of Meng (2022) and Geva (2023) that the last subject token state in middle layers are strongly casual, we are primarily interested in utilizing the Jacobian (derivative) between the hidden state at the last subject token x_s^i, and the last token position overall, the object prediction x_o^L. The LM computation between these two states is clearly highly nonlinear, but within certain relational contexts this derivative can yield faithful approximations (Hernandez et al., 2023).

2.2 Internal Relational Representation

Relations can be encoded as $n \times n$ matrices, or linear transformations. In transformer models, positional encodings are designed to linearly encode relative positions through a range of methods (Vaswani et al., 2017; Su et al., 2024).

Through linear probing and embedding analysis, transformer models have been found to encode high-level linguistic features in internal embedding representations, such as syntactic dependencies and thematic categories (Kann et al., 2018; Tenney et al., 2019; Wilson et al., 2023).

Meng et. al. (2022) found that factual statement predictions exhibit strongly causal states in middle layers at last subject token, supporting the idea that an enriched subject representation exists prior to output. Geva (2023) demonstrated that attribute extraction is often performed by specific attention heads in later layers, and takes the form of a query on the enriched representation. The AttributeLens directly applies this notion to extract encoded attributes from hidden states (Hernandez et al., 2023).

We directly build off of work by Hernandez et al. (2023), who present a an affine linear approximator, known by the corresponding internal hypothesis of the Linear Relational Encoding. With **s** denoting the hidden subject state and **o** denoting the final object state, they treat object-retrieval within a relational context as linearly approximable: $\mathbf{o} = F(\mathbf{s})$. They model o with a first-order Taylor approximation

$$o = F(\mathbf{s}) \approx W\mathbf{s} + b$$

using the transformer Jacobian $\frac{\partial F}{\partial s}$ from relation examples to approximate W, and utilizing a subject representation **s** from an intermediate layer. By doing so, they achieve over 60% faithfulness for LM predictions across certain factual, commonsense, linguistic, and bias relations.

2.3 Linear Embedding Spaces

Paccanaro and Hinton (2001) introduced the concept of the linear relational embedding for learning relational knowledge from triples (a, R, b). Along with prior work (Hinton, 1986), they were able to solve a family tree problem where data is given in relational triples (Colin, *child*, Victoria), where vector components captured implicit semantics such as generation. Concepts such as a and b are represented as n-length vectors, while relations such as R are represented as $n \times n$ matrices, akin to Coecke's vector semantics (2010) and similar to the hidden state representation described above.

Mikolov et al. (2013) used linear operations in word vector space derived from context-predictive neural nets, demonstrating a correspondence between directional binary relations (male-female, country-capital, verb tense) and the addition of certain embedding vectors. Subsequent work found inflection relations (*comparative*, strong:stronger) are better captured than derivation relations (*lacking*, life:lifeless), and that encyclopedic relations (*capital-of*, Greece:Athens) are better captured than lexicographic relations (*member-of*, player:team) (Gladkova et al., 2016; Vylomova et al., 2016).

Park et al. (2023) formalize the compositional representation of concepts in embedding spaces. Extending prior work (Wang et al., 2023), they define a set of counterfactual outputs Y for a directional binary concept W. Letting W = male⇒female, the space of outputs comprise:

$$(Y(W{=}0), Y(W{=}1)) \in \{(\text{"}man\text{"}, \text{"}woman\text{"}), (\text{"}king\text{"}, \text{"}queen\text{"}), \ldots\}$$

They formalize concept intervention as adding an embedding representation $\bar{\lambda}_W$ to change the probability of an output reflecting a concept W. For any concept Z linearly separable from W, an output word $Y(W, \ldots, Z)$, and concept embedding λ, an intervention is effective if it changes the probability of W but not Z.

Subsequent work (Park et al., 2024) illustrated concepts were linearly encoded in the final embedding layer, through noun projections with particular binary characteristics against estimated feature vectors.

2.4 In-context learning

Our work utilizes in-context learning (ICL) for both training and testing purposes, where input-label pairs are provided as demonstration for a novel task. Min (2022)'s in-depth empirical study

of ICL finds that ground truth demonstrations are not necessary, and suggests that more important to ICL is the identification of a label and output space, while Wei et al. (2024) proposes that learning input-label mappings is an emergent ability of large LLMs. Yan (2023) also performs an in-depth study on what they term the token reinforcement loop, providing empirical evidence of $n = 8$ as an optimal number of examples for the LRE. During the testing process, we reproduce Hernandez et. al. (2023)'s findings that approximations without relation-specific contexts generally perform significantly worse than relation-specific contexts. Further work in this area is important for understanding how concepts are represented in transformers.

3 Problem Statement

The LRE is well motivated mathematically, under the assumption the transformer computation is linearly approximable for a specific contextual relation. The object retrieval function from a subject with a fixed relational context, $o = F(s)$, is hypothesized by Hernandez et al. (2023) to be modeled by a first-order Taylor approximation of F about a number of examples $s_1 \dots s_n$. For $i = 1 \dots n$:

$$\begin{aligned} F(s) &\approx F(s_i) + W(s - s_i) \\ &= F(s_i) + Ws - Ws_i \\ &= Ws + b, \\ \text{where } b &= F(s_i) - Ws_i \end{aligned}$$

Note that we get a W and b for each s_i. This motivates the following definition for W and b over a relation. They can be calculated as the mean Jacobian and bias between n enriched subjects $s_1 \dots s_n$ and outputs $F(s_1) \dots F(s_n)$ for a fixed relation:

$$W = \mathbb{E}_{s_i}\left[\left.\frac{\partial F}{\partial s}\right|_{s_i}\right]$$

$$b = \mathbb{E}_{s_i}\left[\left.F(s) - \frac{\partial F}{\partial s}\, s\right|_{s_i}\right]$$

The LRE diverges from its namesake, the linear relational embedding introduced by Hinton (1986), by introducing the bias b and scaling β terms:

$$\mathbf{o} \approx \beta W\mathbf{s} + b$$

They claim the LRE is limited by layer normalization: the $\mathbf{s}$ representation is normalized before contributing to $\mathbf{o}$, and $\mathbf{o}$ is normalized before token prediction by the LM head, resulting in a mismatch in the scale of the output approximation. We find that this conclusion is supported by empirical evidence from linear projections.

However, while linearity is assumed by Hernandez by calculating W and b from as $\mathbb{E}_{s_i}$ over $i = 1 \dots n$, defining the approximation as a Taylor series implicitly makes a weaker assumption. Under the assumption that the relation is not only linearly approximable, but truly linear, we would expect the following approximation to be valid:

$$\mathbf{o} \approx F'(s_i)\mathbf{s}$$

This motivates the definition for a linear approximation over $s_1 \dots s_n$ within the same relation to be simply the mean Jacobian [1]:

$$W = \mathbb{E}_{s_i}\left[\left.\frac{\partial F}{\partial s}\right|_{s_i}\right]$$

$$F(s) = Ws$$

This is the form given in the original linear relational embedding (Paccanaro and Hinton, 2001). We will test this approximation against the LRE; for truly linear relations, we anticipate equivalent performance.

4 Approach

4.1 Introducing New Relations

The Bigger Analogy Test Set, was originally introduced to explore linguistic regularities in word embeddings (Gladkova et al., 2016). It provides forty different categories, ten each in inflectional morphology, derivational morphology, encyclopedic knowledge, and lexical semantics. Each category consists of 50 unique word pairs; the dataset contains 2000 samples total. The data is compiled from various sources, including WordNet, SemEval2012-Task2, Wikipedia, the Google Analogy Test Set, and a color dataset built for evaluating multimodal models (Fellbaum, 1998; Jurgens et al., 2012; Mikolov et al., 2013; Bruni et al., 2012).

[1] Note that because the scale of the hidden state does not contribute to an output prediction, β is irrelevant:

$$\underset{t}{\operatorname{argmax}}\, D(Ws) = \underset{t}{\operatorname{argmax}}\, D(\beta Ws)$$

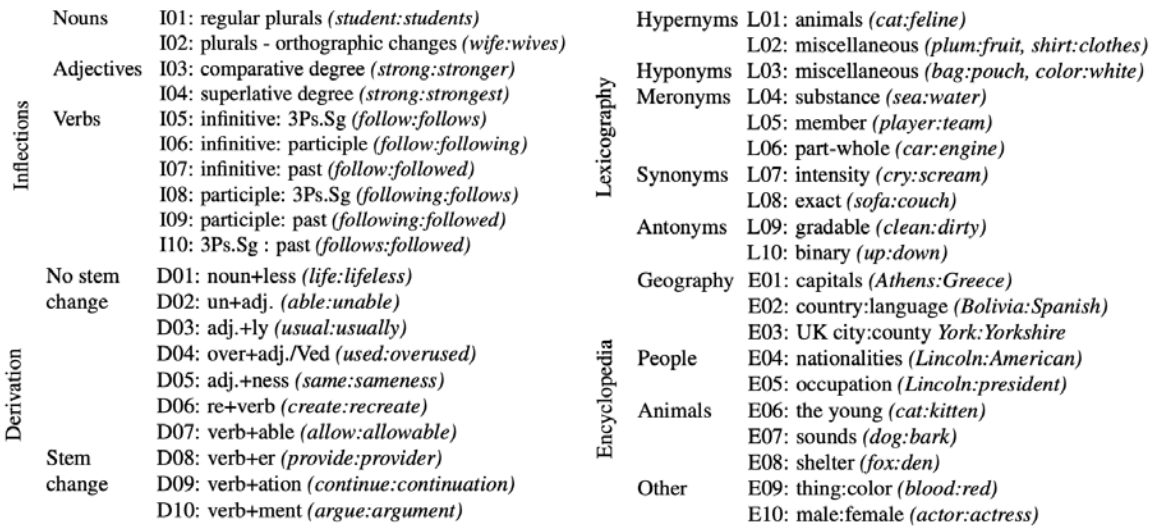


Figure 2: The BATS dataset structure from Gladkova et al. (2016)

We adapt the Bigger Analogy Test Set to a relational dataset by introducing relation-specific prompts for each analogy dataset. The derivational morphology dataset **[verb+ment]** uses the clozed prompt *"To {} results in a"*, which is filled in with subjects to obtain the Jacobians used. For example, one corresponding prompt would be *"To fulfill results in a"*, eliciting the object *"fulfillment"*. We use the first item as the subject and the second as the object, except in **[verb_inf - Ving]**, where the reverse was used.

4.2 Utilizing ICL

Following the testing standards established by Hernandez (2023), we use 8 ICL examples for 8 different subject-object pairs to create an approximator for each relation. For instance, we might approximate **E06 [animal - youth]** with the pairs $\{(dog, puppy), (sheep, lamb), \ldots\}$, prepending the 7 other examples before each pair.

We restrict evaluation to the pairs for which the LM computation is successful in reproducing the actual object in question: for both GPT-J and Llama-7b, this is all or nearly all of the examples provided in BATS.

4.3 Evaluating the Jacobian

We primarily work with the six-billion parameter model GPT-J. Following Hernandez, we measure approximator faithfulness over a relation by the top-one token match rate for the approximation and the LM. Let the enriched subject state be $\mathbf{s}$ and the relation-expressing context be r. Let the transformer computation be $o = F(\mathbf{s})$ and the relational approximator be $\tilde{F}$. Then for token t and decoder head D, we say an approximator is faithful if the top token approximation matches the LM:

$$\underset{t}{\operatorname{argmax}}\, D(F(\mathbf{s}))_t \stackrel{?}{=} \underset{t}{\operatorname{argmax}}\, D(\tilde{F}(\mathbf{s}))_t$$

In the original LRE,

$$\tilde{F} = \beta W \mathbf{s} + b$$

We primarily test two variants of the LRE. First, the Jacobian approximator:

$$\tilde{F} = W\mathbf{s}$$

Second, the Bias approximator:

$$\tilde{F} = \mathbf{s} + b$$

We would like our approximations to generalize over new subjects. In order to do so, we omit the subject-object pairs used to build the approximator from the testing pool.

5 Results

5.1 The Jacobian Faithfully Approximates Morphological Relations

We build approximators for likely subject hidden states (layers 3-9) and the final object state (layer

27) through the process outlined above. We then evaluate the approximators four times, with randomized test prompts each iteration, and average the best performing approximation from each. For the LRE, we use $\beta = 7$, which was found to be optimal for BATS. We find that the Jacobian reproduces derivational and inflectional morphology particularly well [2]. In most other morphology categories, the LRE performance does not improve significantly past the Jacobian, suggesting that the object representation is well captured by the Jacobian alone.

The high faithfulness of the Jacobian shows that it is sufficient to approximate most morphological relations, but not that it is necessary. To show that the Jacobian is also necessary, we compare against the Bias approximation $\mathbf{s} + b$, (equivalent to $\mathbf{s} + \mathbb{E}(o - W\mathbf{s})$). We also compare against the TRANSLATION approximator, where the bias is formulated as $b = \mathbb{E}(o - \mathbf{s})$.[3] We find that without the Jacobian, bias approximations fail to approximate nearly all morphological relations, while successfully capturing some semantic and encyclopedic relations: the bias approximator achieves 67% faithfulness on **[things - color]**, while the TRANSLATION estimator attains 50% and 52% faithfulness on **[animal - shelter]** and **[hypernyms - misc]** respectively.

5.2 Llama-7b Results

While these results show that the Jacobian is a faithful approximator for morphological relations in GPT-J, it is possible that the unique architecture decisions have contributed to the observed linearity. We repeat the process for Llama-7b, which like most popular LLMs, utilizes sequential attention and feedforward layers (Touvron et al., 2023). As seen in Figure 4, we obtain very similar results. [4]

We can conclude that morphological relations can be decoded linearly in LMs, and that they are likely to be linearly encoded.

In general, the Jacobian does poorly on semantic and encyclopedic relations, highlighting the complementary role of the bias term.

5.3 Underlying Mechanisms

Due to the layer normalization within the decoder head, the scale of the hidden state does not contribute to an output prediction. We have

$$\underset{t}{\operatorname{argmax}}\, D(\beta W\mathbf{s} + b)_t = \underset{t}{\operatorname{argmax}}\, D(W\mathbf{s} + \frac{b}{\beta})_t$$

In other words, the scale factor β on $W\mathbf{s}$ is equivalent to $\frac{1}{\beta}$ on b, and that the LRE approaches $W\mathbf{s}$ asymptotically for high β. In practice, we observed that for $\beta > 100$, the performance of the LRE becomes negligibly different from $W\mathbf{s}$.

We would like to interpret W and b. The approximation results in Figure 3 and 4 give credence to the idea that W and b play complementary roles in the approximation. One potential interpretation is that the weight provides variation in the embedding space, while the bias is a concept shift. With this interpretation, b can be compared to the vectors used by Mikolov and many others, and the concept vector subsequently formalized by Park. However, it's important to note the bias vector and the concept vector are not exactly analogous. The bias term describes an offset from the transformed subject to the object: $b = \mathbb{E}(o - W\mathbf{s})$, not $b = \mathbb{E}(o - s)$. We observe the bias vector does not generally lie in the same direction as $\mathbb{E}(o - s)$, suggesting it may play a different role in transforming the subject.

We find that it is possible to get interpretable subject representations through linear projection onto the span $\{b, \perp\}$, where $\perp$ is a vector orthogonalized through Gram–Schmidt to b [5]. They suggest W is primarily responsible for transforming the underlying distribution to be geometrically similar to the output, while b contributes the majority of movement in vector space. Figures 5 and 6 both display embeddings projected to $\{b, \perp\}$.

Note that the shapes of the transformed subject spaces **Ws** and **Ws + b** are both similar to the object space. Note that the b scale is much larger than the $\perp$ scale.

[2] The exceptions are the prefix relations **[re+verb_reg]** and **[over+adj_reg]**, and present participle relations. We will address these further below.

[3] This approximator, from Hernandez et. al. (2023), calculates the direct offset of the subject and object hidden states, and is inspired by Merullo et al. (2023) and Word2Vec arithmetic. See the appendix for the results.

[4] This diagram will be updated in the final version to look like the GPT-J one: with the best approximations after layer sweeping, and with β optimized for Llama-7b.

[5] There are many options for this orthogonal vector; we chose the first weight column vector W_0.

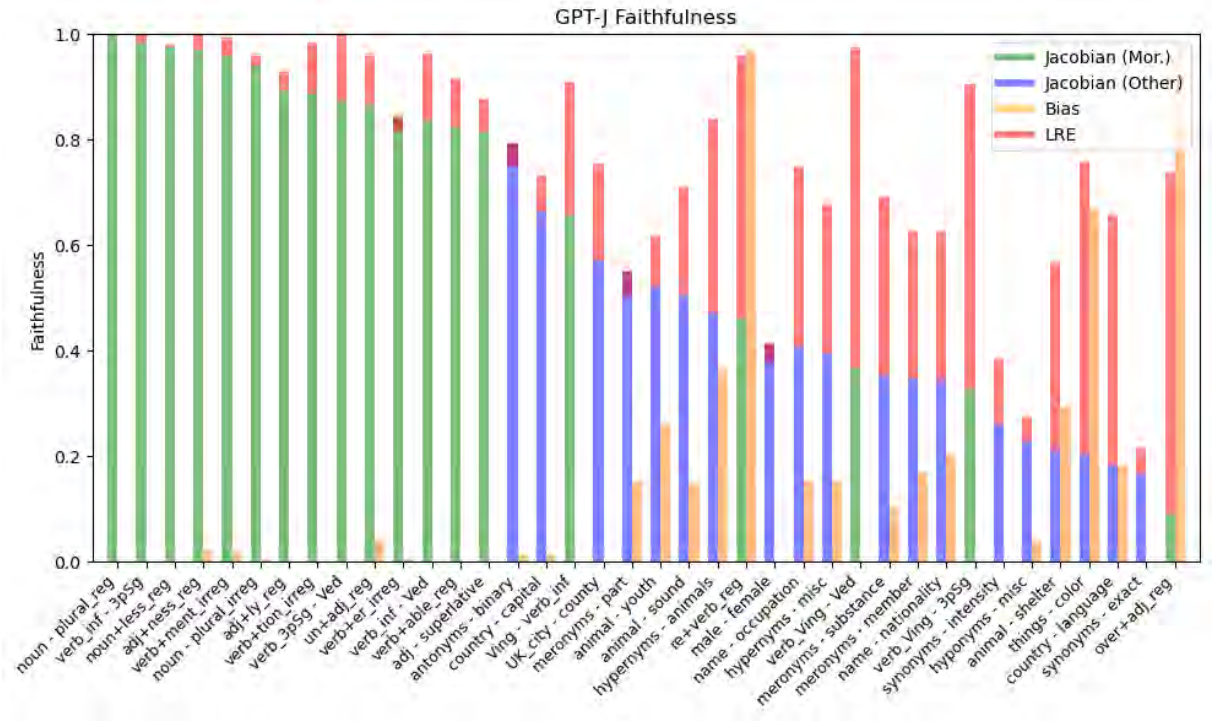


Figure 3: Breaking down the affine LRE into Jacobian ($W\mathbf{s}$) and Bias ($\mathbf{s}+b$) approximators suggests that W and b play complementary roles: the Jacobian is responsible for approximating morphology, while the bias is responsible for conceptual shifts.

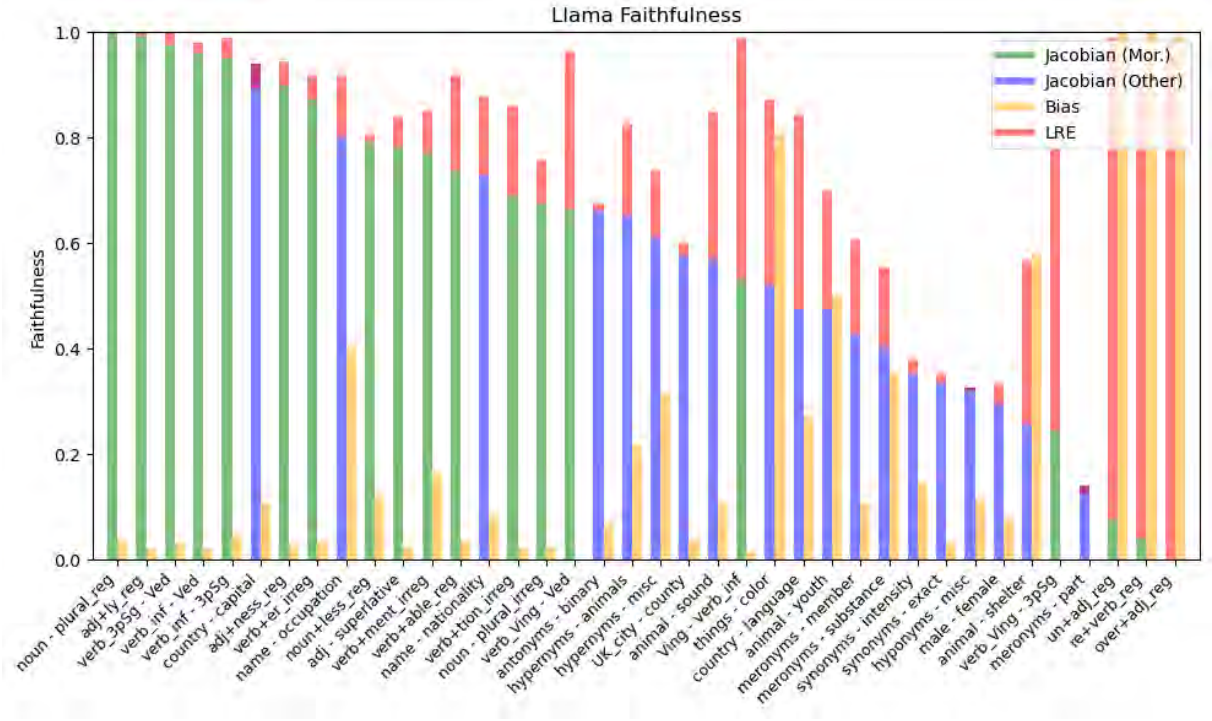


Figure 4: Comparing the LRE and Jacobian for Llama-7b reproduces the results seen above for GPT-J, suggesting the high faithfulness of the Jacobian for morphological relations is widespread among LMs.

Projection also aids in interpreting β in the LRE, which scales the output approximation $\beta W\mathbf{s} + b$. When this approximation approaches the output embeddings $\mathbf{o}$, the performance of the

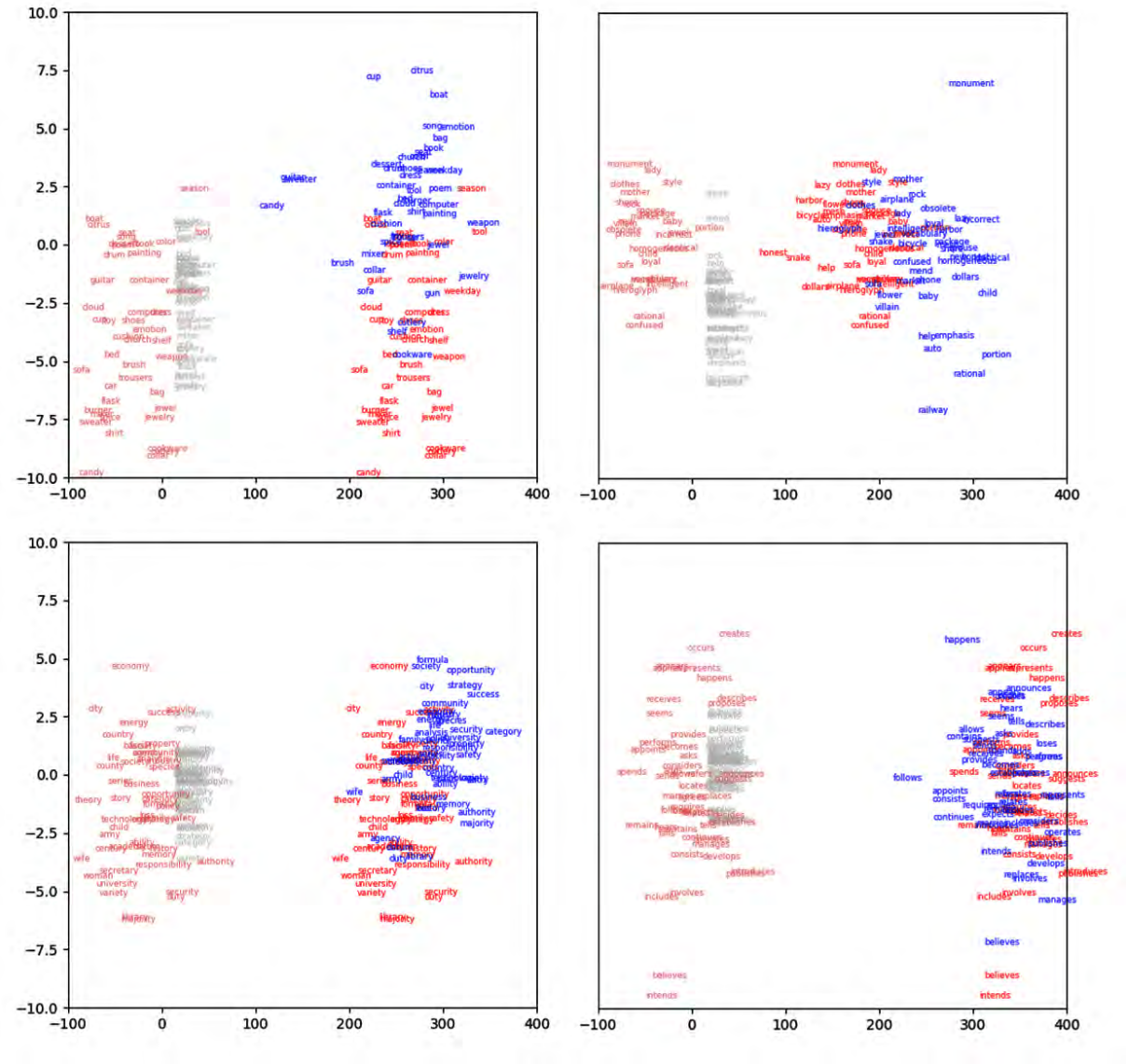


Figure 5: Output space projections of $\mathbf{s}$, $\beta W\mathbf{s}$, $\beta W\mathbf{s}+b$ and $\mathbf{o}$ can be used to diagnose nonapproximable relations. Above, the ineffective **[hyponyms - misc]** and **[synonyms - exact]** approximations do not resemble their corresponding outputs, despite high cosine similarity scores. Below, the effective **[noun - plural_irreg]** and **[verb_3pSg - Ved]** approximations closely resemble their outputs.

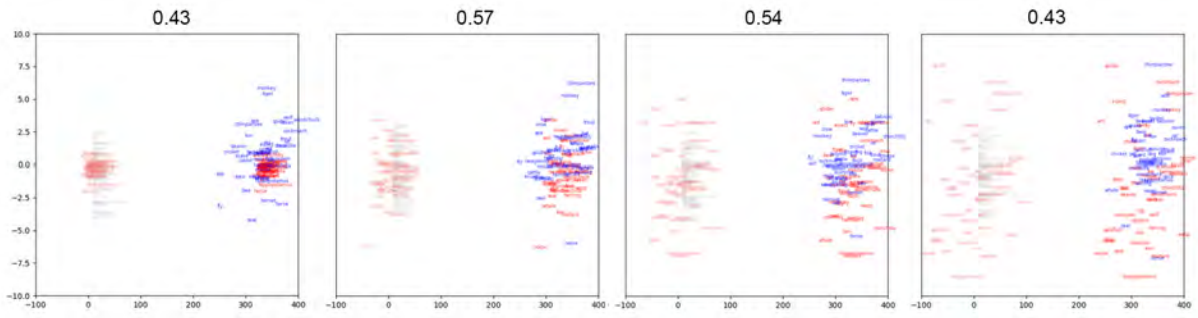


Figure 6: A projection of $\mathbf{s}$, $\beta W\mathbf{s}$, $\beta W\mathbf{s}+b$ and $\mathbf{o}$ for **[animal - shelter]** for $\beta = 1, 3, 5, 7$ with faithfulness scores demonstrates that embedding distances corresponds to the accuracy of the approximation.

approximator improves. In Figure 6, we see empirical evidence that β restores the magnitude of change that was lost through layer normalization, as conjectured by Hernandez et. al. (2023).

Discussion

5.4 Counterarguments

We have shown empirically that it is possible to linearly decode morphological relations in LMs with a high degree of faithfulness. However, several potential issues must be addressed prior to considering the theoretical implications.

Subject	Jacobian Top-3
society	societies, Soc, soc
child	children, children, Children
success	successes, success, Success
series	series, Series, Series
woman	women, women, Women
manage	manager, managers, manager
teach	teacher, teachers, teach
compose	compos, composer, composing
borrow	borrower, lender, debtor
announce	announcer, announ, ann
righteous	righteousness, righteous, ...
conscious	consciousness, conscious, ...
serious	seriousness, serious, serious
happy	happiness, happy, happy
mad	madness, mad, being
invest	investment, invest, investing
amuse	amusement, amuse, amusing
accomplish	accomplishment, accomplish, ...
displace	displacement, displ, dis
reimburse	reimbursement, reimburse, reimb
globalize	globalization, global, international
install	installation, install, Installation
continue	continuation, continu, contin
authorize	authorization, Authorization, ...
restore	restoration, restitution, re

Table 1: **[noun_plural], [verb+er], [verb+ment], [adj+ness], [verb+tion]** Selected examples from GPT-J show that relational Jacobian approximation captures irregular morphology effectively, and does not only reproduce stemmed subject forms.

What if the Jacobian is just modeling syntax? One argument against linear encoding is that the Jacobian is not learning morphology, but instead some regular syntax in the relation. Then, the high faithfulness reported merely reflects some orthographic change from the base, and not a true morphological relation.

What if the Jacobian is replicating the subject? Another argument against linear encoding is that the faithfulness metric is a bad choice for measuring morphological faithfulness. High faithfulness scores on many reported inflectional tasks can be achieved simply by reproducing a substring of the subject token.

To provide a counterargument against the two possibilities above, we provide specific approximation token predictions in Table 1. While generally the fact that approximation outputs tend to be stemmed forms is not a cause for concern, we observe there exist many tokens such as #25303 ' sadness' and #24659 ' continuation' which faithfully replicate morphology.

There are two inflectional relationships the Jacobian failed to approximate as well over the tests performed, **[Ving - 3psg]** and **[Ving - Ved]**.It's possible that transformations from the verb active form make the LM computation non-linear. For the majority of the relations on which the Jacobian achieves high faithfulness, the subject is the unmarked form, such as the verb infinitive or third person singular.

There are two derivational prefix tasks for which the LRE, but not the Jacobian, faithfully approximates, **[re+verb]** and **[over+adj]**. The Jacobian does achieves a high faithfulness on the prefix relation **[un+adj]**, so the notion that prefix relations are distinctive from other morphology is not supported. A partial explanation for this phenomenon may be that the object tokens "over" and "re" are idiosyncratically related by an offset to the subjects, unlike other relations. With fewer correct object tokens than average, a linear subject transformation without any bias may fail to model the relation effectively.

5.5 Implications for Concept Theory

Geometrically, the findings suggests that morphological relations between words do not involve additional concept vectors. Above, we have demonstrated that the bias term is not necessary for morphological terms, and even results in incorrect approximation for low values of β. This is compatible with the Linear Relational Hypothesis, which

Relation	# Unique
un+adj	**7**
over+adj	**4**
re+verb	**15**
name - nationality	13
animal - shelter	18
synonyms - intensity	35
verb+able	47
noun - plural	47

Table 2: The number of unique starting object tokens for selected BATS relations.

posits that subspace distances in LMs are fundamentally about semantics. If morphology is encoded as linear transformation, vectors can retain their semantic interpretations.

Conclusion

In this work, we have adapted the Bigger Analogy Test Set to create a large novel testing dataset for relations, covering forty relations over morphological, factual, and semantic relations. We find Jacobian approximation models morphological relations well. We hypothesise that the Jacobian serves the role of *extending* a subject entity to alternative forms (including morphological derivations), and the bias term serves the role of *shifting* underlying concepts. We validate this hypothesis through embedding projections of model transformations.

Through linear approximation of a language model, we arrive at a better understanding of its internal structure, which is crucial for controlling its outputs effectively. This ultimately has implications for many downstream applications of transformer language models, including as knowledge bases, dialogue agents, and as robust tools for inference and reasoning.

Reproducibility statement

The code is based on the LRE repository, and loads GPT-J in half-precision. The code and the dataset are available at https://github.com/rkique/linear-morphology. Experiments were run remotely on a workstation with 24GB NVIDIA RTX 3090 GPUs using HuggingFace Transformers.

Acknowledgments

All work herein reported is supported by the Nation Science Foundation under Grant No. 2349452. Any opinion, finding, or conclusion in this study is that of the authors and does not necessarily reflect the views of the National Science Foundation.

References

[Brown et al.2020] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

[Bruni et al.2012] Elia Bruni, Gemma Boleda, Marco Baroni, and Nam-Khanh Tran. 2012. Distributional semantics in technicolor. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 136–145.

[Chanin et al.2023] David Chanin, Anthony Hunter, and Oana-Maria Camburu. 2023. Identifying linear relational concepts in large language models. *arXiv preprint arXiv:2311.08968*.

[Coecke et al.2010] Bob Coecke, Mehrnoosh Sadrzadeh, and Stephen Clark. 2010. Mathematical foundations for a compositional distributional model of meaning. *arXiv preprint arXiv:1003.4394*.

[Fellbaum1998] Christiane Fellbaum. 1998. *WordNet: An Electronic Lexical Database*. Bradford Books.

[Geva et al.2023] Mor Geva, Jasmijn Bastings, Katja Filippova, and Amir Globerson. 2023. Dissecting Recall of Factual Associations in Auto-Regressive Language Models, October. arXiv:2304.14767 [cs].

[Gladkova et al.2016] Anna Gladkova, Aleksandr Drozd, and Satoshi Matsuoka. 2016. Analogy-based detection of morphological and semantic relations with word embeddings: what works and what doesn't. In Jacob Andreas, Eunsol Choi, and Angeliki Lazaridou, editors, *Proceedings of the NAACL Student Research Workshop*, pages 8–15, San Diego, California, June. Association for Computational Linguistics.

[Hernandez et al.2023] Evan Hernandez, Arnab Sen Sharma, Tal Haklay, Kevin Meng, Martin Wattenberg, Jacob Andreas, Yonatan Belinkov, and David Bau. 2023. Linearity of relation decoding in transformer language models. *arXiv preprint arXiv:2308.09124*.

[Hinton1986] Geoffrey E Hinton. 1986. Learning distributed representations of concepts. In *Proceedings*

of the Eighth Annual Conference of the Cognitive Science Society, volume 1, page 12. Amherst, MA.

[Jurgens et al.2012] David Jurgens, Saif Mohammad, Peter Turney, and Keith Holyoak. 2012. SemEval-2012 task 2: Measuring degrees of relational similarity. In Eneko Agirre, Johan Bos, Mona Diab, Suresh Manandhar, Yuval Marton, and Deniz Yuret, editors, **SEM 2012: The First Joint Conference on Lexical and Computational Semantics – Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*, pages 356–364, Montréal, Canada, 7-8 June. Association for Computational Linguistics.

[Kann et al.2018] Katharina Kann, Alex Warstadt, Adina Williams, and Samuel R Bowman. 2018. Verb argument structure alternations in word and sentence embeddings. *arXiv preprint arXiv:1811.10773*.

[Meng et al.2022] Kevin Meng, Arnab Sen Sharma, Alex J Andonian, Yonatan Belinkov, and David Bau. 2022. Mass-editing memory in a transformer. In *The Eleventh International Conference on Learning Representations*.

[Merullo et al.2023] Jack Merullo, Carsten Eickhoff, and Ellie Pavlick. 2023. Language models implement simple word2vec-style vector arithmetic. *arXiv preprint arXiv:2305.16130*.

[Mikolov et al.2013] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

[Min et al.2022] Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2022. Rethinking the Role of Demonstrations: What Makes In-Context Learning Work?, October. arXiv:2202.12837 [cs].

[Paccanaro and Hinton2001] Alberto Paccanaro and Geoffrey E. Hinton. 2001. Learning distributed representations of concepts using linear relational embedding. *IEEE Transactions on Knowledge and Data Engineering*, 13(2):232–244.

[Park et al.2023] Kiho Park, Yo Joong Choe, and Victor Veitch. 2023. The linear representation hypothesis and the geometry of large language models.

[Park et al.2024] Kiho Park, Yo Joong Choe, Yibo Jiang, and Victor Veitch. 2024. The geometry of categorical and hierarchical concepts in large language models.

[Su et al.2024] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. RoFormer: Enhanced transformer with Rotary Position Embedding. *Neurocomputing*, 568:127063.

[Tenney et al.2019] Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. Bert rediscovers the classical nlp pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4593–4601.

[Touvron et al.2023] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

[Vaswani et al.2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

[Vylomova et al.2016] Ekaterina Vylomova, Laura Rimell, Trevor Cohn, and Timothy Baldwin. 2016. Take and took, gaggle and goose, book and read: Evaluating the utility of vector differences for lexical relation learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1671–1682.

[Wang and Komatsuzaki2021] Ben Wang and Aran Komatsuzaki. 2021. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. `https://github.com/kingoflolz/mesh-transformer-jax`, May.

[Wang et al.2023] Zihao Wang, Lin Gui, Jeffrey Negrea, and Victor Veitch. 2023. Concept algebra for score-based conditional model. In *ICML 2023 Workshop on Structured Probabilistic Inference & Generative Modeling*.

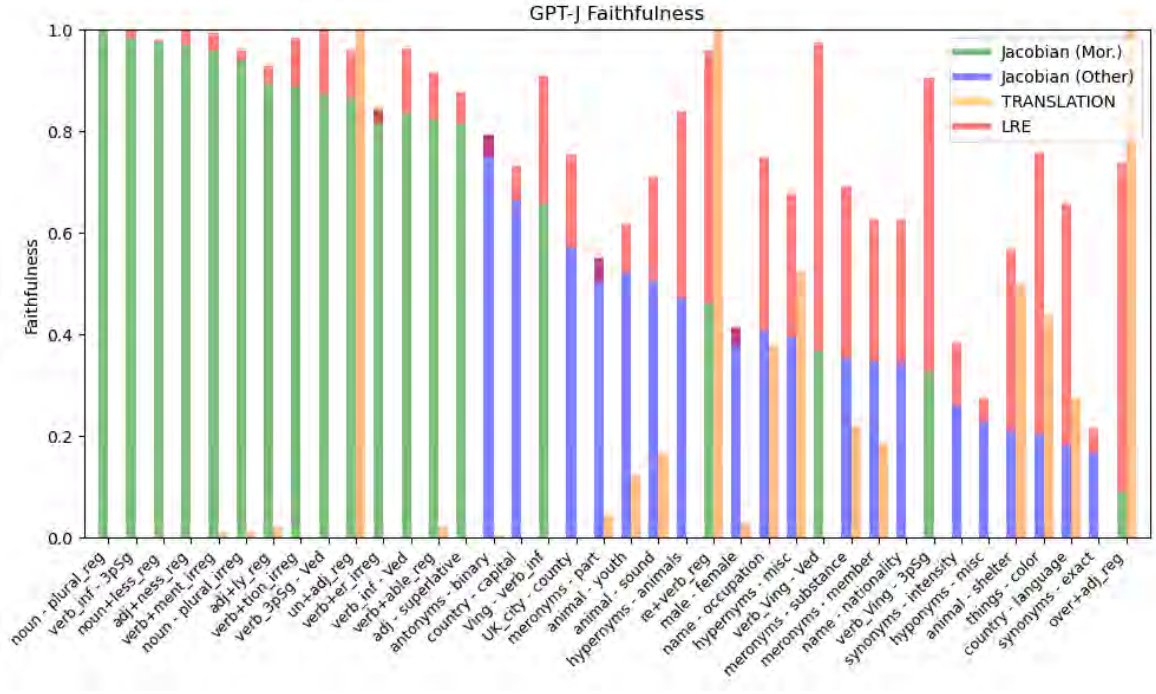
[Wei et al.2024] Jerry Wei, Jason Wei, Yi Tay, Dustin Tran, Albert Webson, Yifeng Lu, Xinyun Chen, Hanxiao Liu, Da Huang, Denny Zhou, and Tengyu Ma. 2024. Larger language models do in-context learning differently.

[Wilson et al.2023] Michael Wilson, Jackson Petty, and Robert Frank. 2023. How abstract is linguistic generalization in large language models? experiments with argument structure. *Transactions of the Association for Computational Linguistics*, 11:1377–1395.

[Yan et al.2023] Jianhao Yan, Jin Xu, Chiyu Song, Chenming Wu, Yafu Li, and Yue Zhang. 2023. Understanding in-context learning from repetitions. In *The Twelfth International Conference on Learning Representations*.

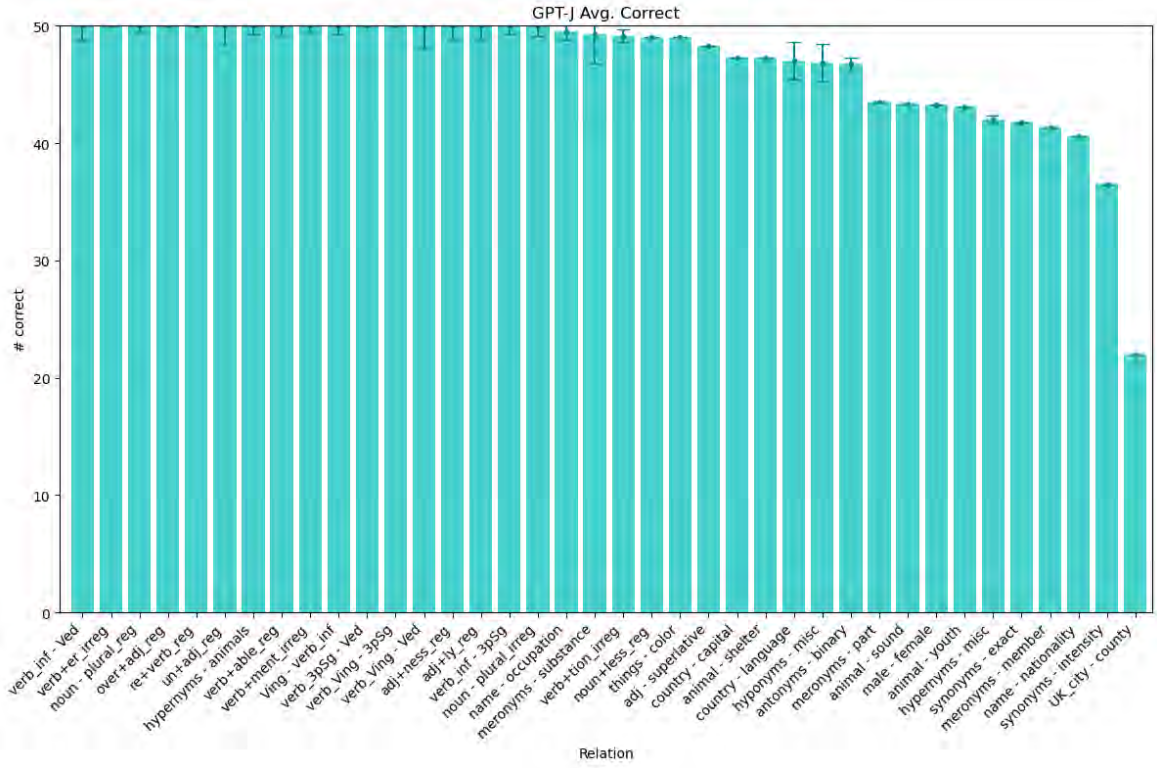
A The Jacobian with TRANSLATION

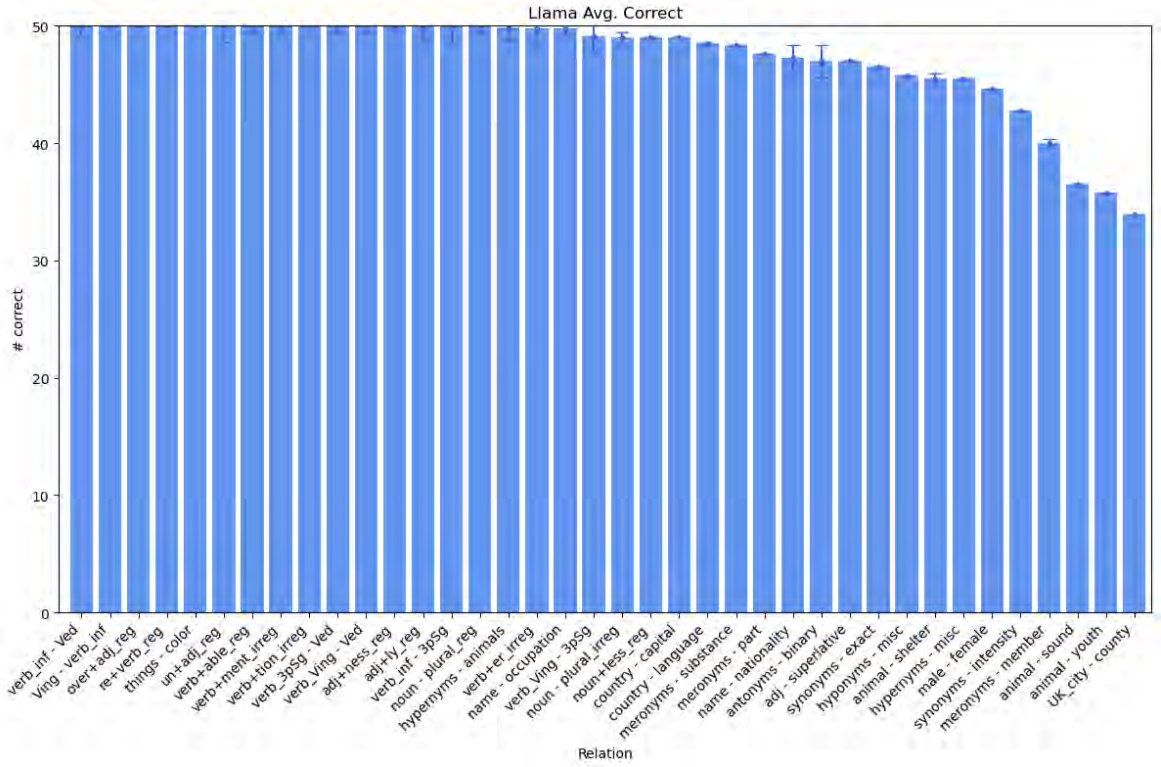
Figure 7: Comparing the affine LRE with the Jacobian ($W\mathbf{s}$) and TRANSLATION ($\mathbb{E}(\mathbf{o} - \mathbf{s})$) approximators yields similar results to above, suggesting that W and b play complementary roles.



B GPT-J and Llama-7b Ability

Figure 8: Both GPT-J and Llama-7b demonstrate mastery of most relations.




Llama Avg. Correct
correct
Relation
0
10
20
30
40
50
verb_inf - Ved
Ving - verb_inf
over+adj_reg
re+verb_reg
things - color
un+adj_reg
verb+able_reg
verb+ment_irreg
verb+tion_irreg
verb_3pSg - Ved
verb_Ving - Ved
adj+ness_reg
adj+ly_reg
verb_inf - 3pSg
noun - plural_reg
hypernyms - animals
verb+er_irreg
name - occupation
verb_Ving - 3pSg
noun - plural_irreg
noun+less_reg
country - capital
country - language
meronyms - substance
meronyms - part
name - nationality
antonyms - binary
adj - superlative
synonyms - exact
hyponyms - misc
animal - shelter
hypernyms - misc
male - female
synonyms - intensity
meronyms - member
animal - sound
animal - youth
UK_city - county

Automatic Summarization of Long Documents

Naman Chhibbar
IIT Hyderabad
Kandi, Sangareddy
Telangana, India 502285
ma21btech11011@iith.ac.in

Jugal Kalita
University of Colorado, Colorado Springs
1420 Austin Bluffs Pkwy
Colorado Springs CO 80918
jkalita@uccs.edu

Abstract

A vast amount of textual data is added to the internet daily, making utilizing and interpreting textual data difficult and cumbersome. Therefore, text summarization is crucial for efficiently extracting relevant information. Although many Transformer models excel in summarization, they are constrained by the input size, preventing them from processing texts longer than their context size. This study introduces several novel algorithms to efficiently overcome the input size limitations, utilizing the full potential of LLMs without any architectural changes. We test our algorithms on texts with more than **70,000 words**, and our experiments show a significant increase in BERTScore with competitive ROUGE scores.

1 Introduction

Due to the abundance of online text data, document summarization has become crucial for efficient and accurate extraction of relevant information from a piece of text. Over the past few years, Large Language Models (LLMs) based on the Transformer model (Vaswani et al., 2017) have shown groundbreaking abilities for several NLP tasks, including document summarization (Yadav et al., 2023). Recent developments have demonstrated remarkable improvements in the relevancy and coherence of summaries generated by such LLMs.

Long document summarization is more crucial than ever since it involves removing redundancies from a piece of text, making reading the text concise and efficient. One of the major limitations of Transformers is limited context size, stemming from the quadratic memory and computational complexity of the attention mechanism in their architecture (Du et al., 2023). This constraint hinders the extraction of relevant information from extensive texts where summarization is valuable to overcome the interpretative challenges posed by complex relations on such a large scale.

We experiment with various novel approaches to address the input size limitations of LLMs. The methods introduced do not include any architectural modifications to the model used for summarization and can be incorporated into any existing summarization pipeline. We believe that these methods can effectively utilize the full potential of any existing LLM. Though our experiments only include the task of summarization, our methods can be applied to any NLP task (sequence classification, question-answering, etc.) that requires processing long texts.

We start by stating the problem statement (2) and discussing some related works (3) to gain insights into the problem and the state-of-the-art solutions. We then introduce the datasets (4) and methodology (5) used in our experiments. For evaluating our results, we present some common metrics (6) used in text summarization. We end the report by discussing our experimental findings (7) and potential future work (8), and concluding the study (9).

2 Problem Statement

Our goal is to distill a long document of theoretically indefinite length, such that it fits within the context size of the model while retaining important information. Practically, the document length may be up to **ten times** the context size of the model used. For our experiments, we aim to reduce the summary length to about 400 words, preserving maximal information and coherence.

3 Related Works

Golia and Kalita (2024) take a "Divide and Conquer" based approach to address sequence length limitations in summarizing long meeting transcripts. They begin by segmenting the transcript and then use the BART (Bidirectional and Auto-Regressive Transformer) model to summarize each segment individually. These segment summaries

are then recursively combined and summarized until a single summary remains. This method performs well with long documents but may take a considerable amount of time to converge due to repeated calls to the summarizer.

There have also been efforts to improve the efficiency of attention mechanisms in Transformers. Beltagy et al. (2020) introduce Longformer, which replaces the quadratic self-attention mechanism in the Transformer architecture with a sliding window self-attention, resulting in a linear complexity with respect to the input size. To capture long-ranged dependencies, they include global attention at specific token positions. Huang et al. (2021) modify the encoder-decoder attention mechanism such that each attention head in the decoder attends to n/s_h tokens in the input sequence, where n is the input length and s_h is the number of heads. This method has a complexity of $O(mn/s_h)$, where m is the length of the output sequence. Bertsch et al. (2023) introduce Unlimiformer, which also modifies the encoder-decoder attention in a Transformer. The attention heads in the decoder only attend to the tokens picked by their k-Nearest-Neighbor (kNN) algorithm. The kNN indices between the input tokens are created by the hidden states generated in the encoder. Phang et al. (2022) introduce the staggered block-local attention mechanism. In the block-local attention mechanism, the input sequence is divided into multiple non-overlapping blocks. Tokens in a block attend only to the tokens in the same block. In staggered block-local attention, the blocks are staggered such that each token is in a different block in each head.

Other unique approaches include VideoAgent, introduced by Wang et al. (2024), an AI agent designed to answer a given question based on a long video. They achieve this by generating captions from multiple uniformly sampled frames from the video. These captions are used to answer the user's question. Chen et al. (2022) describe a novel algorithm to classify long Chinese news into a set of predefined categories. They form multiple groups of sentences based on a maximum token threshold in each group. These groups are then encoded using BERT (Bidirectional Encoder Representations from Transformers) and passed through a 1D convolution layer for local feature extraction. What makes this method special is that the attention mechanism is replaced by a 1D convolution layer, which has linear complexity. Chen et al. (2023) use positional interpolation to extend the context size of a pre-trained model. Instead of the usual extrapolation of the positional embeddings, they downscale the positional embeddings to force them into a range the model is trained on, hence interpolating in the pre-trained range. They claim that it is better for the model to use the positional embeddings on which it is trained.

4 Datasets

We have used datasets containing documents with a maximum word count exceeding 70,000. We briefly discuss and analyze the word counts of the datasets below.

GovReport

Introduced by Huang et al. (2021), this dataset consists of reports written by government research agencies, including the Congressional Research Service (CRS) and the U.S. Government Accountability Office (GAO). Exact word count information is given in Table 1. Figure 1 shows the word count distribution of the dataset.

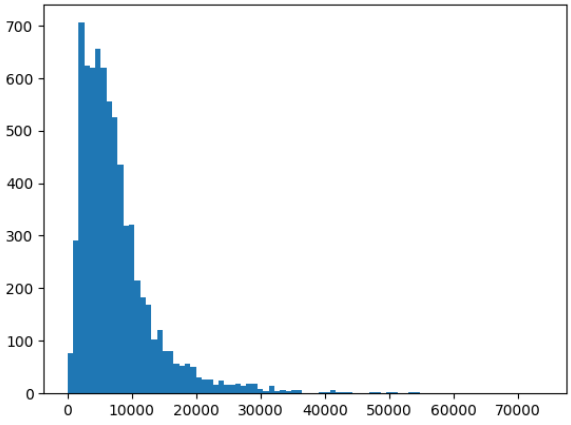


Figure 1: GovReport word counts

BigPatent

Introduced in Sharma et al. (2019), this dataset consists of over 1.3 million records of U.S. patent documents with human-written abstractive summaries. Exact word count information is given in Table 1. Figure 2 shows the word count distribution of the dataset.

5 Methodology

In this section, we discuss the algorithms used in our experiments. Most of our algorithms start by segmenting the document into smaller, contiguous,

Dataset	Avg. Word Count	Max Word Count	No. of Documents
GovReport	7,700.71	**73,815**	7,238
BigPatent	3,055.72	**71,027**	1,341,362

Table 1: Dataset information

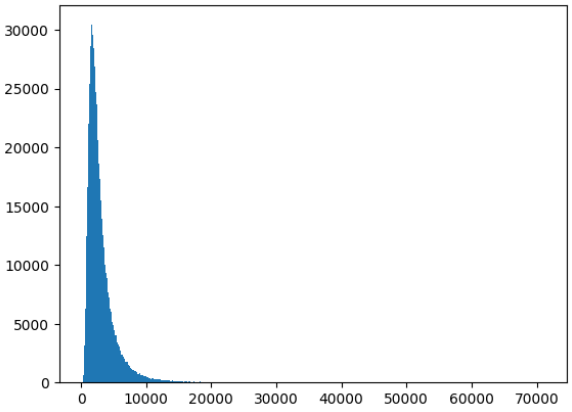


Figure 2: BigPatent word counts

and exhaustive parts. The segmentation algorithm is controlled by a threshold on the minimum number of words in a segment, which is a hyperparameter in each of the respective methods.

5.1 Central Truncation

Truncation is the most common and straightforward approach used to handle long texts that exceed the context size of an LLM. It is done in three main ways:

- **Retaining Head**: Keeping tokens from the start.
- **Retaining Tail**: Keeping tokens from the end.
- **Head and Tail**: Keeping tokens from both start and end.

Worsham and Kalita (2018) also employ "retaining head" and "retaining tail" strategies on long texts and find promising results. Though the "retaining head" method is often used, keeping the initial tokens allowed by the LLM, Sun et al. (2019) find that keeping both head and tail produces better results than both the "retaining head" and the "retaining tail" methods. Their research also shows that truncating the middle is even better than the more complicated hierarchical methods, displaying superiority with simplicity.

The fraction of tokens to be taken from the head is controlled by the hyperparameter $head\_size \in [0, 1]$. Setting $head\_size = 1$ results in taking tokens only from the head, whereas setting $head\_size = 0$ results in taking tokens only from the tail. The truncated text is then sent to the summarizer.

5.2 Document Skimming

One way to process long texts is by employing a speed reading strategy called "skimming". Skimming is done by reading the whole text in a go while selectively skipping some parts of the text for quicker reading. The reader usually omits the portions that seem redundant or irrelevant in the text, minimizing information loss.

This method starts by segmenting the text using a segmenter with the hyperparameter min_words. The segmenter uses a sentence tokenizer to separate sentences from the text, and then merges the sentences that have fewer words than min_words. We then sample segments uniformly, with each segment having probability p to be picked. The sampled segments are then merged to form a single text and sent to the summarizer. This ensures we sample a segment from each part of the text.

To address the removal of redundancy in the document, we experiment with and without removing redundant segments before and after sampling. This is done by computing the cosine similarity between the mean segment embeddings and the current segment embedding. This sentence transformer is used to generate the segment embeddings. This sentence transformer is based on MiniLM (Wang et al., 2020), which is a distilled version of larger encoder-only transformer models. If the cosine similarity is greater than a threshold, the current segment is removed. The mean embedding is updated if the current segment is retained.

When removing segments after sampling, we increase the probability of choosing the segments before sampling to compensate for the removed segments. This fraction is controlled by the hyperparameter $prob\_boost$. The updated probability is calculated as $p_{new} = (1 + prob\_boost) \cdot p_{optimal}$. Even though removing redundant segments before sampling is less efficient due to the whole docu-

ment being processed, it ensures better utilization of the LLM's context size. The mean embedding is initialised as a zero vector and updated after each segment is processed. Doing so introduces a hyperparameter $threshold$ to control the similarity threshold between the mean embedding and the current segment embedding.

This approach is a modification to the methodology Wang et al. (2024) used for question-answering on long videos. Worsham and Kalita (2018) also use random sampling for genre identification.

We will now discuss calculating the optimal value of p here. Let X denote the total number of tokens in the sampled segments. Since segments are sampled randomly, X is a random variable. If the context size of the model is $model\_size$, we want $\mathrm{E}[X] = model\_size$, where $\mathrm{E}[X]$ denotes the expectation of X.

Suppose we have $n \in \mathbb{N}$ segments and $X_i \sim \mathrm{Bernoulli}(p)$ denotes if segment i is chosen, $i \in \{1, 2, \ldots, n\}$. If len_i denotes the number of tokens in segment i, we can write:

$$X = \sum_{i=1}^{n} X_i \cdot len_i$$

$$\Rightarrow \mathrm{E}[X] = \mathrm{E}[\sum_{i=1}^{n} X_i \cdot len_i]$$

$$\Rightarrow \mathrm{E}[X] = \sum_{i=1}^{n} \mathrm{E}[X_i \cdot len_i]$$

$$\Rightarrow \mathrm{E}[X] = \sum_{i=1}^{n} \mathrm{E}[X_i] \cdot len_i$$

Since $X_i \sim \mathrm{Bernoulli}(p)\ \forall i \in \{1, 2, \ldots, n\}$, we have $\mathrm{E}[X_i] = p\ \forall i \in \{1, 2, \ldots, n\}$.

$$\Rightarrow \mathrm{E}[X] = \sum_{i=1}^{n} p \cdot len_i$$

$$\Rightarrow \mathrm{E}[X] = p \cdot \sum_{i=1}^{n} len_i$$

Let $total\_len$ be the total number of tokens in the text, then $total\_len = \sum_{i=1}^{n} len_i$.

$$\therefore \mathrm{E}[X] = p \cdot total\_len = model\_size$$

$$\Rightarrow p \cdot total\_len = model\_size$$

$$\therefore p = model\_size / total\_len$$

5.3 Summarization with Keyword Extraction

Document skimming (5.2) involves a very intuitive and simple approach of sampling segments randomly. Instead of that, we can use an efficient keyword extraction algorithm to get important keywords from the document. These keywords help us sample segments intelligently, ensuring we get the most important segments from the document.

We use Latent Dirichlet Allocation (LDA) (Blei et al., 2003) with a single topic to get topic words (keywords) from the document. These keywords are concatenated using a delimiter (a space is used in our experiments) to form a single sentence. This sentence is then embedded and compared with other segment embeddings using cosine similarity. Maximum number of segments with the highest similarity are retained. The keyword sentence and document segments are embedded using the same sentence transformer used in the previous method.

This approach is similar to the way Golia and Kalita (2024) use action items to pick segments of text (a neighbourhood of 2 sentences around the action item) for summarization.

6 Evaluation Metrics

The best way to evaluate generated natural language is by humans, but conducting human trials are expensive and time-consuming. Hence, we use automatic evaluation metrics to evaluate the quality of the generated summary, given some reference summaries. Fabbri et al. (2021) review many such open-source and state-of-the-art metrics. Some of them that we use in our experiments are discussed below.

ROUGE metrics: Lin (2004) introduces the Recall-Oriented Understudy for Gisting Evaluation (ROUGE) metrics. The basic ROUGE-N metric is based on the fraction of overlaps of some ideal summaries with the candidate summary, hence being recall-oriented. His study concludes that ROUGE-N with N = 2, ROUGE-L, ROUGE-W, and ROUGE-S work well for the summarization task.

BERTScore: Zhang et al. (2019) introduce the BERTScore, an automatic evaluation metric for text generation. BERTScore is calculated by comparing the contextual embeddings of tokens in the candidate and reference summaries, which are generated using BERT (Devlin et al., 2018). BERTScore excels at capturing semantic similarities between sentences since it uses contextual embeddings of

tokens instead of using N-gram frequencies to calculate similarity.

7 Experimental Findings

We test our pipelines with the following summarizers: **BART** (Bidirectional and Autoregressive Transformer) (Lewis et al., 2020) fine-tuned on the CNN/Daily Mail dataset (Nallapati et al., 2016) with a context size of 1024, **LongT5** (Guo et al., 2021), a variant of T5 (Text-to-Text Transfer Transformer) (Raffel et al., 2020), fine-tuned on the BookSum dataset with a context size of 4096, and **GPT-3.5 Turbo** (Brown et al., 2020) with a context size of 4096.

We compare our results with the state-of-the-art summarization models on the GovReport dataset, including Unlimiformer (Bertsch et al., 2023) integrated with BART (Lewis et al., 2020) and PRIMERA (Beltagy et al., 2020), Hepos (Huang et al., 2021), PEGASUS-X with staggered block-local attention (Phang et al., 2022), extended LLaMA-7B with positional interpolation. Refer to Table 2 for the baseline results and our results.

We also compare our results with BigBird-Pegasus (Zaheer et al., 2020) on the BigPatent dataset. Refer to Table 3 for the results.

Time complexity analysis

We evaluate the time complexity of our methods by measuring the mean time taken to process a document (excluding the time taken by the summarizer to generate the summaries). We find that Central Truncation (5.1) and Document Skimming (5.2) take approximately the same time. Skimming with post-sampling removal takes slightly more time than the other two methods. We can see a significant increase in time taken by Skimming with pre-sampling removal and Summarization with Keyword Extraction (5.3) due to the additional computations required. Figure 3 illustrates the average time taken by our methods. Check Table 4 for exact values rounded off to two decimal places.

8 Future Work

We have used a very basic sentence tokenizer (nltk.sent_tokenize), with some modifications to control the minimum number of words in a segment, to segment the document. In our experiments, we find that segmentation is a crucial step in the pipeline and can influence the output summary significantly. Ensuring the uniformity of the length of the segments while preserving coherence is important for better utilization of the context size of the summarizer while sampling. We encourage future work to experiment with more complex segmenters.

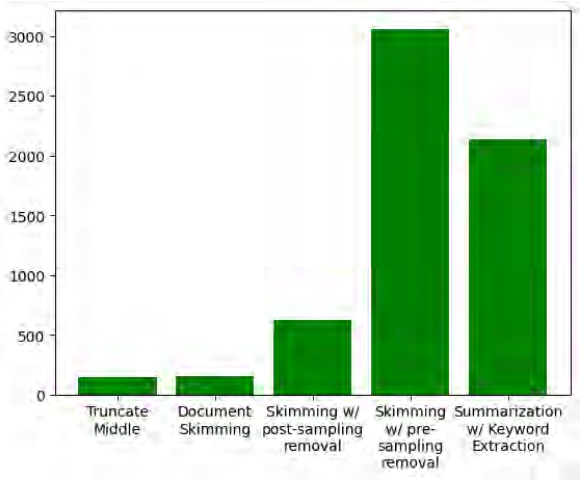


Figure 3: Mean time taken per document using BART tokenizer on BigPatent dataset

Future work can also be focused on extending the Summarization with Keyword Extraction (5.3) method. Experimenting with different keyword extraction algorithms and use cases is encouraged.

9 Conclusion

Our experiments show that Document Skimming with post-sampling removal (5.2) performs well while being efficient. The Central Truncation method (5.1) also shows good results, which shows that simple methods can also be effective in long document summarization. The last two methods, Skimming with pre-sampling removal and Summarization with Keyword Extraction (5.3), show the best results but are computationally expensive.

Our experiments show a huge jump in BERTScore compared to Unlimiformer on documents with word counts up to 70,000. This shows that our pipelines are able to utilize details in long texts efficiently. Even though our ROUGE-2 scores are lower than the baselines, ROUGE-1 and ROUGE-L scores are competitive. Since BERTScore is a better metric for capturing semantic similarity, we hypothesize our pipelines can generate better summaries than the baselines with higher ROUGE scores.

Model	ROUGE-1	ROUGE-2	ROUGE-L	BERTScore
BART w/ Unlimiformer (1,024)	53.4	22.5	22.5	66.0
PRIMERA w/ Unlimiformer (4,096)	56.5	24.8	26.3	67.7
Hepos (10,240)	51.34	19.09	**48.73**	-
PEGASUS-X w/ Staggered Block-Local Attention (16k)	60.3	**30.0**	31.5	-
LLaMA-7B w/ Positional Interpolation (15k)	60.0	28.0	29.5	-
Summarization w/ Extraction + GPT-3.5 Turbo (4,096)	**61.99**	18.52	38.46	**86.20**
Central truncation + LongT5 (4,096)	46.20	4.38	38.27	**82.19**
Skimming w/ post-sampling removal + LongT5 (4,096)	46.76	4.56	39.61	**81.96**

Table 2: Automatic evaluation results on the GovReport dataset. Context size of the models are mentioned in parenthesis. The best score in each metric category is highlighted in **bold**.

Model	ROUGE-1	ROUGE-2	ROUGE-L	BERTScore
BigBird-Pegasus (16k)	**60.64**	**42.46**	**50.01**	-
Skimming w/ pre-sampling removal + GPT-3.5 Turbo (4,096)	27.40	3.31	21.25	**82.62**
Central truncation + GPT-3.5 Turbo (4,096)	27.77	3.09	20.56	**82.57**
Skimming w/ post-sampling removal + GPT-3.5 Turbo (4,096)	26.16	2.13	20.21	**82.40**

Table 3: Automatic evaluation results on the BigPatent dataset. Context size of the models are mentioned in parenthesis. The best score in each metric category is highlighted in **bold**.

Method	Mean time taken
Central Truncation	142.50 ms
Document Skimming	155.42 ms
Skimming w/ post-sampling removal	625.17 ms
Skimming w/ pre-sampling removal	3059.63 ms
Summarization w/ Extraction	2131.40 ms

Table 4: Mean time taken per document using BART tokenizer on BigPatent dataset

Acknowledgement

All work herein reported is supported by the Nation Science Foundation under Grant No. 2349452.

Any opinion, finding, or conclusion in this study is that of the authors and does not necessarily reflect the views of the National Science Foundation.

References

Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

Amanda Bertsch, Uri Alon, Graham Neubig, and Matthew Gormley. 2023. Unlimiformer: Long-range transformers with unlimited length input. In *Advances in Neural Information Processing Systems*, volume 36, pages 35522–35543. Curran Associates, Inc.

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3(null):993–1022.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. 2023. Extending context window of large language models via positional interpolation. *arXiv preprint arXiv:2306.15595*.

Xinying Chen, Peimin Cong, and Shuo Lv. 2022. A long-text classification method of chinese news based on bert and cnn. *IEEE Access*, 10:34046–34057.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep

bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Jiangsu Du, Jiazhi Jiang, Jiang Zheng, Hongbin Zhang, Dan Huang, and Yutong Lu. 2023. Improving computation and memory efficiency for real-world transformer inference on gpus. *ACM Trans. Archit. Code Optim.*, 20(4).

Alexander R Fabbri, Wojciech Kryściński, Bryan McCann, Caiming Xiong, Richard Socher, and Dragomir Radev. 2021. Summeval: Re-evaluating summarization evaluation. *Transactions of the Association for Computational Linguistics*, 9:391–409.

Logan Golia and Jugal Kalita. 2024. Action-item-driven summarization of long meeting transcripts. In *Proceedings of the 2023 7th International Conference on Natural Language Processing and Information Retrieval*, NLPIR '23, page 91–98, New York, NY, USA. Association for Computing Machinery.

Mandy Guo, Joshua Ainslie, David Uthus, Santiago Ontanon, Jianmo Ni, Yun-Hsuan Sung, and Yinfei Yang. 2021. Longt5: Efficient text-to-text transformer for long sequences. *arXiv preprint arXiv:2112.07916*.

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. 2021. Efficient attentions for long document summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1419–1436, Online. Association for Computational Linguistics.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Ramesh Nallapati, Bowen Zhou, Caglar Gulcehre, Bing Xiang, et al. 2016. Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023*.

Jason Phang, Yao Zhao, and Peter J Liu. 2022. Investigating efficiently extending transformers for long input summarization. *arXiv preprint arXiv:2208.04347*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

Eva Sharma, Chen Li, and Lu Wang. 2019. BIGPATENT: A large-scale dataset for abstractive and coherent summarization. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2204–2213, Florence, Italy. Association for Computational Linguistics.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2019. How to fine-tune bert for text classification? In *Chinese computational linguistics: 18th China national conference, CCL 2019, Kunming, China, October 18–20, 2019, proceedings 18*, pages 194–206. Springer.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in Neural Information Processing Systems*, 33:5776–5788.

Xiaohan Wang, Yuhui Zhang, Orr Zohar, and Serena Yeung-Levy. 2024. Videoagent: Long-form video understanding with large language model as agent. *arXiv preprint arXiv:2403.10517*.

Joseph Worsham and Jugal Kalita. 2018. Genre identification and the compositional effect of genre in literature. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1963–1973, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Avaneesh Kumar Yadav, Ranvijay, Rama Shankar Yadav, and Ashish Kumar Maurya. 2023. State-of-the-art approach to extractive text summarization: a comprehensive review. *Multimedia Tools and Applications*, 82(19):29135–29197.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. 2020. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2019. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*.

Text Summarization using Rhetorical Structure Trees

Edward Trenk
Hamilton College
eddietrenk1@gmail.com

Melkamu Mersha
University of Colorado
Colorado Springs
mmersha@uccs.edu

Jugal Kalita
University of Colorado
Colorado Springs
jkalita@uccs.edu

Abstract

The vast volume of media being created each day has necessitated research into automatic text summarization. The hypothesis guiding this work is that a well-written summary uses rhetorical devices to convey the main content of a longer text. To investigate, this work uses Rhetorical Strucutre Theory (RST) to better understand the structure of text and create summaries. We present an RST-aided graph pruning approach to select important sentences for extractive summarization, which compares well in performance to popular baseline methods. Additionally, we propose a novel multi-step method for short text summarization by employing RST tags to train the BART language model and produce abstractive summaries. Finally, we prompt GPT 3.5 Turbo with and without RST tags to compare results. We conclude that RST is not useful for general news summarization tasks.

1 Introduction

With the tremendous rise in the quantity of available textual documents, people have become reliant on short summaries to peruse and consume content. This creates a need for summarization of all types of texts. Summarization involves different techniques depending on the document length — short or long. There is a wide range of approaches to short text summarization using techniques such as rule-based methods, recurrent neural networks, and large language models.

Summaries may belong to one of two categories: extractive or abstractive. Extractive summaries consist of the most important pieces lifted directly from the source text. This captures the author's distinctive voice and maintains content accuracy, but can be difficult to comprehend since the extracted phrases are pieced together, often without transition and referential words. In contrast, abstractive summaries are original wordings, much like those of a human who creates a more coherent summary but may suffer from factual inaccuracy (Bleiweiss, 2023).

This paper posits that a summary conveys the most salient information in a longer document, and hence how a summary is structured is likely to have an impact on its quality. We consider structure of a summary in terms of the rhetorical components — in terms of related clauses and sentences — that comprise it.

This paper introduces a novel approach for text summarization, providing both extractive and abstractive summaries. Our graph pruning extractive approach leverages the ability of RST to identify important parts of a source text.

We also develop deep learning approaches for summarization that take into account the rhetorical structures present within the original articles and reference summaries while training. One such approach inserts rhetorical labels both within the articles and reference summaries during training, with a view to learning effects of rhetorical structures. Finally, we explore ways to instruct a large language model, in particular ChatGPT, to incorporate rhetorical structures in producing document summaries. We compare results of all the approaches and conclude that our hypothesis fails, and understanding rhetorical structure is not necessary for summarization. We believe that this finding is surprising and deserves attention.

Section 2 of the paper provides an overview of related research motivating this work, Section 3 details the research problem, Section 4 gives the motivation and discusses implementation of our approaches, Section 5 presents our experiments

and the results as compared with previous work, Section 6 acknowledges the associated risks, and Section 7 discusses concluding thoughts.

2 Related Work

The following sections highlight popular summarization strategies and the models supporting this work.

2.1 Rhetorical Structure Theory

Rhetorical Structure Theory (Mann and Thompson, 1988), or RST, creates binary trees connecting the Elementary Discourse Units (EDUs) of a text by their rhetorical relations. Single EDUs or groupings of them are classified into either Nucleus or Satellite in which the relationship from the Satellite to the Nucleus takes on one of the defined relations such as Background, Elaboration, or Contrast. Hence, at each split in the tree there is one relation defined from the Satellite subtree to the Nucleus subtree. In some cases, a split consists of two Nuclei, implying that the two subtrees are independent of each other. A simple example can be seen in Figure 1. In this example, EDU2 provides background for EDU1. Additionally, the second sentence (EDU3-EDU5) elaborates on the first (EDU1-EDU2).

2.2 Summarization

Automatic text summarization has a large research base and has been attempted using myriad approaches.

2.2.1 Extractive Methods

Extractive methods often employ a three-step process to split up the source text, assign each piece a score, and select the most important ones. These algorithms consist of supervised and unsupervised learning methods. Supervised methods need more information to train on such as human written summaries. Implementations of supervised methods often use neural networks such as RankNet (Burges et al., 2005). Meanwhile, unsupervised approaches produce a summary while only accessing the input text making them more computationally efficient and requiring less human labor (Mridha et al., 2021). Extractive approaches have applications in everyday life including Google News summaries (Shakil et al., 2024).

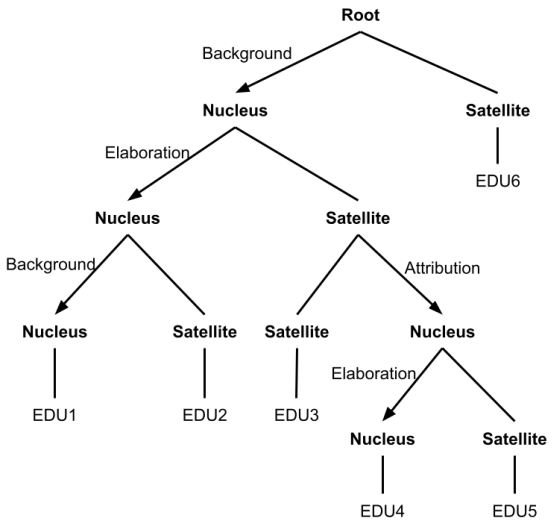


Figure 1: An example RS-tree of the following input text: (EDU1) *Harry Potter star Daniel Radcliffe gets £20M fortune* (EDU2) *as he turns 18 Monday.* (EDU3) *Young actor says* (EDU4) *he has no plans* (EDU5) *to fritter his cash away.* (EDU6) *Radcliffe's earnings from first five Potter films have been held in trust fund.*

2.2.2 Abstractive Methods

Abstractive methods generally can generally be classified as struture-based or semantic-based methods. Strucure-based summaries implement data strucutes such as templates (Oya et al., 2014) or trees (Kurisinkel et al., 2017) but are limited in contextual information and diversity. Although semantic-based methods largely avoid these issues, they struggle with sentence cohesion and have limited use cases. Remaining abstractive summarization algorithms include deep learning (Song et al., 2019) and graph-based approaches (Mridha et al., 2021). A hybrid approach of extractive and abstractive methods first separates the important sentences and then uses a Large Language Model (LLM) to rephrase the summary (Koh et al., 2022).

2.2.3 Rhetorical Structure Theory

RST has been used in the process of creating extractive summaries by taking sentences from near the roots of the parsing tree (Zhang et al., 2023). Hirao et al. (2015) summarize Wall Street Journal articles by pruning rhetorical structure trees for important EDUs using a translation to depen-

dency trees and dynamic programming selection algorithm. Thus, dependency relations have been shown to be useful for identifying important parts of a text; rather than just dependency relationships, this paper explores whether any benefits accrue from incorporating full RST knowledge into summarization approaches.

2.2.4 Large Language Models

Transformer architectures (Vaswani et al., 2017) are effective for summarizing small inputs, and it has even been shown that humans prefer news summaries generated by LLMs to those written by humans (Goyal et al., 2022). An advantage of LLMs is their ability to work with zero-shot or few-shot prompting, so they do not require fine-tuning (Zhang et al., 2024). Prompt engineering is a more resource efficient approach to text summarization which uses techniques such as template engineering to directly obtain a summary or chain of thought to break the problem up (Jin et al., 2024).

3 Statement of Problem

Since RST gives insight into the structure of a text, a goal of this work is to determine whether RST is a helpful tool for summarization. Better understanding of the rhetorical structures of source texts and reference summaries may allow for improved summarization results and could generalize to other NLP tasks. Thus, exploring the benefits and disadvantages of RST is worth studying.

4 Approach

4.1 Build Rhetorical Structure Trees

The first step to summarizing using rhetorical relations is to create a RST parse tree for the source texts. Our method uses the DMRST[1] algorithm (Liu et al., 2021) to find the rhetorical structure tree (RS-tree) of the source text and the associated summaries. The output of the DMRST code is a string which we use to recursively construct a custom tree for ease of use on future tasks.

4.2 Tree Pruning

Once the tree is constructed, we create baseline summaries by pruning the tree for important EDUs. Hirao et al. (2015), present an effective way to find important sentences from a RS-tree by translating to a dependency tree which only considers Nucleus and Satellite relationships. By following the path of nucleus dependencies, we arrive at an EDU which everything else builds off of. This is the root of the dependency tree. In the simple example from Figure 1, following Nucleus dependencies leads to EDU1. This strategy, finds an important fragment to include in a summary but is not sufficient for covering an entire article. Hirao et al. then use dynamic programming to select additional EDUs for their summary by traversing the dependency tree from the root. Our approach diverges from theirs at this point by finding the root but otherwise disregarding the dependency tree.

Our approach centers around selecting EDU's from throughout the source text. Algorithm 1 displays the pseudo code for this approach. Function Get-Top-Sentence(·) traverses the tree following Nucleus nodes only until it reaches the leaf EDU. For each EDU we select, we include the entire sentence that it is a part of. This is necessary to provide context for parts of the EDU, especially as related to understanding pronouns.

The threshold that we are using to travel down a tree is based on dividing the total number of EDU's by a length parameter, α, and rounding down to the nearest integer. Hence, we search down approximately α subtrees, representing α distinct parts of the source text, with the goal of getting α sentences for the final summary. For our final testing, we use $\alpha = 3$ since three sentences is the typical summary length in our dataset.

Algorithm 1 RS-Tree-Graph-Pruning

```
procedure EXTRACTIVE-SUMMARY(article)
    rs_tree = Build-RS-Tree(article)
    sent_list = []
    sent_list = Rec-Search-Tree(rs_tree, sent_list)
    return List-To-String(sentence_list)
end procedure

procedure REC-SEARCH-TREE(root, sent_list)
    top_sent = Get-Top-Sentence(root)
    Append top_sent to sent_list
    if root.left_subtree_size > EDU_COUNT / α then
        Rec-Search-Tree(root.left_child, sent_list)
    end if
    if root.right_subtree_size > EDU_COUNT / α then
        Rec-Search-Tree(root.right_child, sent_list)
    end if
    return
end procedure
```

In order to improve the pruned results, we use sentence similarity[2] to avoid including redundant

[1] https://github.com/seq-to-mind/DMRST_Parser

[2] https://huggingface.co/sentence-transformers

Reference summary	Motorised surfboards weigh just 15kg and travel at speeds of up to 33mph . Capable of thrusting riders into the air off waves and the wake from boats . Contraption, made in Czech Republic, is powered by a two-stroke engine .
Lead3 extractive summary	With a surfboard strapped to his feet, a thrill-seeker soars through the air as he showcases a new extreme sport. This action shot was taken at Estavayer-le-Lac in Switzerland where the latest water sports craze of Jet Surfing has taken off. The contraption can be seen leaping up above the surface, almost resembling scenes from the film Back to the Future where Michael J Fox's character Marty McFly climbs on to a futuristic hoverboard.
Pruned summary	With a surfboard strapped to his feet, a thrill-seeker soars through the air as he showcases a new extreme sport. But unlike in the 1985 classic, this Jet Surf board, made in the Czeck Republic, is designed for travelling across water - and is so powerful it can help propel riders into the air off waves or the wake from passing boats. Extreme: Jet surf boards have a top speed of 33mph and weigh just 15kg. A bit more traditional: The surfers can get down to the water if they fancy something a little more sedate .
GPT 3.5 Turbo Summary	Jet Surfing, a new extreme water sport, allows riders to soar through the air on powerful, lightweight boards. With top speeds of 33mph, the sport is easy to master and offers an adrenaline rush while minimizing risks.
BART model trained on raw articles and raw summaries	Jet surf boards have a top speed of 33mph and weigh just 15kg. It is equipped with a specially-designed two-stroke engine which uses up to two litres of fuel an hour depending on the model and conditions.

Table 1: A comparison between a reference summary and the ones we generate for a hand-picked example from the CNN/DailyNews dataset.

sentences. We iterate through the sentences of the pruned summary and add them to a final summary if and only if their similarity score is less than β with every sentence already in the final summary. The optimal β value found was 0.4, and we also adjusted this value in the case where the new summary had fewer than 30 words.

4.3 Deep Learning Summarizer with Rhetorical Structure

To generate better summaries, we train the BART (Lewis et al., 2020) Large Language Model on our custom tagged data. We use the RS-tree representation of each article and summary to add dependency (Nucleus or Satellite) and relationship (e.g., Elaboration, Contrast) tags to the original text. We also employ the NLTK sentence tokenizer (Bird and Loper, 2004) to add sentence beginning and end tags to the text.

We create and train our model with a few different types of tags as well as untagged text. The two final tag types are: tagging just the nucleus EDUs, and tagging all EDUs and interior relations. In the latter case, we have transformed the RS-tree into a string without losing any of the information. This allows us to train the BART sequence to sequence model without needing to work with less resourced Graph Neural Networks. We train the model with every combination of tagged and untagged text to compare the results.

4.4 Prompting Large Language Model

In addition to fine-tuning the BART model, we use the chatGPT 3.5 Turbo API with one-shot prompting to generate summaries. This approach is tested first by simply providing an article and asking for a summary. In a second variation, we provide a tagged article and describe the basics of RST before asking for a summary. In both cases, one example article-summary pair is given to guide the model. After testing many prompts, the best two were selected and can be seen in Table 2. Further explanation of our design choices can be seen in Appendix A.Following our experimental results and the work of Zhang et al. (2024), we use a temperature value of 0.3 to control the random-

ness of the model. We also use the LLM to paraphrase our pruned summaries so that they match the style of the reference summaries. This secondary approach uses two-shot prompting which is made possible by the smaller inputs.

Situation	Prompt
Untagged Articles	Summarize this article into approximately 25 to 35 words which should consist of a few simple sentences.
Tagged Articles	Summarize this article into approximately 25 to 35 words using the provided relation classification tags. The beginning of a Nucleus EDU is marked with: `<Nucleus>` The end is marked with: `</Nucleus>`. The Satellite relations follow a similar tagging structure. Use these tags to create a summary which includes mostly information from Nucleus tags.

Table 2: Prompts used for GPT 3.5 Turbo

5 Experiments

The sections below outline our testing process, state-of-the-art comparisons, and initial results.

5.1 Datasets

We test our method on the CNN/DailyMail[3] dataset which contains over 300,000 news articles with associated summaries of about three sentences (Hermann et al., 2015). The articles are taken from 2007 to 2015 with the validation and testing subsets coming chronologically from the end. On average, the articles contain 781 tokens and the summaries contain 58 tokens. Thus, the average reduction in text length is to 7% of the original length. This dataset is ubiquitous in the leading literature for short text summarization.

5.2 Evaluation

We evaluate our results using the Rouge-1, Rouge-2, and Rouge-L evaluation metrics (Lin, 2004) which are ubiquitous for this task. Rouge scores look for overlap of words, so they can be misleading for texts with similar meaning portrayed with different words. We also test our results using Bertscore (Zhang et al., 2019) which focuses on the meaning of the words in the summary rather than the words present. Bertscore provides an advantage over Rouge because it will give a high score for similar words such as "good" and "great" by using dense word embeddings. A critical flaw in both of these metrics is their reliance on words rather than sentence-level meaning. Due to the popularity of the data and evaluation used, we can compare our results to many other state-of-the-art approaches.

[3] https://huggingface.co/datasets/abisee/cnn_dailymail

5.3 Baselines

We compare our results with two extractive baselines as well as two state-of-the-art approaches. Lead3 summarization (Zhu et al., 2021) simply takes the first three sentences from the article to use as a summary. The second extractive baseline is our graph pruning strategy as outlined in Section 4.2. SimCLS (Liu and Liu, 2021) is a reference-less technique which works on top of existing sequence-to-sequence summarization models to improve their results. Alternatively, Zhao et al. (2022) introduce sequence likelihood calibration (SLiC) which works by calibrating sequence generation to reduce loss compared to reference summaries. SLiC and SimCLS demonstrate current state-of-the-art results as a comparison.

5.4 Results

In order to help the reader understand the style of summaries, Table 1 provides example summaries from our different approaches.

We have obtained results across the approaches discussed earlier. Our results are displayed in Table 3. These scores are calculated on the 11,490 article-summary pairs from the test set of the CNN/DailyMail data. The trained BART model is trained on over 244,000 article-summary pairs from the training set.

Our lead3 and tree pruning baseline approaches perform comparably with the slightly unexpected result that lead3 scores better in all metrics used. This is likely due to the nature of news articles which provide summary-like paragraphs to begin. Paraphrasing our pruned summaries with GPT decreased all three Rouge metrics but increased Bertscore. This suggests that the resulting summaries, which were much shorter, lost the original voice of the article but captured more of the summary content in a shorter space. Adding sentence similarity to the pruned summaries short-

Method	Rouge-1	Rouge-2	Rouge-L	BertScore
Lead3	40.05	17.48	25.02	86.92
Tree Pruning	33.33	12.71	21.16	86.17
Tree Pruning + GPT paraphrasing	29.36	8.75	19.37	86.48
SimCLS (Liu and Liu, 2021)	46.67	22.15	43.54	**88.85**
SLiC (Zhao et al., 2022)	**47.97**	**24.18**	**44.88**	_
BART model trained on raw articles and raw summaries	44.52	21.33	30.87	88.13
BART model trained on tagged articles and raw summaries	44.33	21.23	30.71	88.11
BART model trained on tagged articles and tagged summaries	40.75	18.56	27.78	87.30
BART model trained on nucleus tagged articles and raw summaries	44.38	21.35	30.84	88.12
GPT 3.5 Turbo with raw articles	34.53	11.72	23.28	87.36
GPT 3.5 Turbo with tagged articles	32.11	10.01	21.65	87.03

Table 3: Results from assorted summarization methods on the CNN/DailyNews dataset. Best results are shown in bold. All scores are f1 measures which use a combination of precision and recall scores.

ened the length as expected but consistently lowered the scores across all our metrics.

The two GPT prompting methods achieved only marginally better Bertscore results than the baselines. In all four evaluation metrics, the LLM performed better without attempting to use rhetorical structure than it did with the added instructions and tagged information about RST.

Our best results across all our evaluation metrics come from training the BART model. The model trained with no RST knowledge and the two models trained with RST knowledge about just the articles perform at a very similar level and fall slightly short of the state of the art approaches. The final model, trained on article and reference summaries with rhetorical structure tags, is significantly worse than the others and compares more closely with the GPT prompting results.

5.5 Reproducibility Statement

Our experiments were conducted on machine with four NVIDIA GeForce RTX 2080 Ti GPUs each containing 11264 MiB of RAM and running CUDA version 12.2.

6 Potential Risks and Downfalls

Even with breakthroughs in summarization, no method is perfect, and it is possible that automatically generated summaries have factual errors or do not tell the whole story. In the application of new articles, small errors and relying on summaries could facilitate the spread of misinformation.

7 Conclusion

This work supplies evidence that Rhetorical Structure Theory is not useful for general short text summarization.

The performance of our tree pruning strategy implies that rhetorical structure information is at least somewhat useful at identifying important pieces of an input text. However, the tree pruning results falling short of the much simpler lead3 design which does not use RST at all suggests that the knowledge of RST is unnecessary.

Our trained sequence-to-sequence model demonstrated nearly identical results across training with and without knowledge of rhetorical relations. In fact, the best performance we

achieved came from the model trained with no relation tags. The models where rhetorical structure from only the article was included show that although RST did not improve scores, the added information was also not detrimental. In the case of adding tags to both articles and reference summaries, our evaluation indicates that the added information of RST impeded the model's ability to create a good summary.

Our use of the GPT 3.5 Turbo LLM also conveyed the lack of importance of RST. The model performed better without relations than with them, implying that the added understanding of textual structure was more confusing than it was helpful for summarization.

8 Future Work

Despite results suggesting that RST is not necessary for summarization, we are interested in exploring Graph Neural Networks to learn a translation from rhetorical structure trees to summaries.

We believe that RST could be useful for controlled summarization instead of general summarization. Harnessing relations may allow for query based summaries such as providing mainly background as well as stance based summarization which has applications in legal documents and political news.

Acknowledgements

All work herein reported is supported by the National Science Foundation under Grant No. 2349452. Any opinion, finding, or conclusion in this study is that of the authors and does not necessarily reflect the views of the National Science Foundation. We thank Hassan Shakil for contributing to the proposed research approach and conducting important background research.

References

Steven Bird and Edward Loper. 2004. NLTK: The natural language toolkit. In *Proceedings of the ACL Interactive Poster and Demonstration Sessions*, pages 214–217, Barcelona, Spain, July. Association for Computational Linguistics.

Avi Bleiweiss. 2023. Two-step text summarization for long-form biographical narrative genre. In Michael Strube, Chloe Braud, Christian Hardmeier, Junyi Jessy Li, Sharid Loaiciga, and Amir Zeldes, editors, *Proceedings of the 4th Workshop on Computational Approaches to Discourse (CODI 2023)*, pages 145–155, Toronto, Canada, July. Association for Computational Linguistics.

Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, pages 89–96.

Tanya Goyal, Junyi Jessy Li, and Greg Durrett. 2022. News summarization and evaluation in the era of gpt-3. *arXiv preprint arXiv:2209.12356*.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. *Advances in neural information processing systems*, 28.

Tsutomu Hirao, Masaaki Nishino, Yasuhisa Yoshida, Jun Suzuki, Norihito Yasuda, and Masaaki Nagata. 2015. Summarizing a document by trimming the discourse tree. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(11):2081–2092.

Hanlei Jin, Yang Zhang, Dan Meng, Jun Wang, and Jinghua Tan. 2024. A comprehensive survey on process-oriented automatic text summarization with exploration of llm-based methods. *arXiv preprint arXiv:2403.02901*.

Huan Yee Koh, Jiaxin Ju, Ming Liu, and Shirui Pan. 2022. An empirical survey on long document summarization: Datasets, models, and metrics. *ACM computing surveys*, 55(8):1–35.

Litton J Kurisinkel, Yue Zhang, and Vasudeva Varma. 2017. Abstractive multi-document summarization by partial tree extraction, recombination and linearization. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 812–821.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online, July. Association for Computational Linguistics.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July. Association for Computational Linguistics.

Yixin Liu and Pengfei Liu. 2021. Simcls: A simple framework for contrastive learning of abstractive summarization. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 1065–1072.

Zhengyuan Liu, Ke Shi, and Nancy F Chen. 2021. Dmrst: A joint framework for document-level multilingual rst discourse segmentation and parsing. *arXiv preprint arXiv:2110.04518*.

William C Mann and Sandra A Thompson. 1988. Rhetorical structure theory: Toward a functional theory of text organization. *Text-interdisciplinary Journal for the Study of Discourse*, 8(3):243–281.

Muhammad Firoz Mridha, Aklima Akter Lima, Kamruddin Nur, Sujoy Chandra Das, Mahmud Hasan, and Muhammad Mohsin Kabir. 2021. A survey of automatic text summarization: Progress, process and challenges. *IEEE Access*, 9:156043–156070.

Tatsuro Oya, Yashar Mehdad, Giuseppe Carenini, and Raymond Ng. 2014. A template-based abstractive meeting summarization: Leveraging summary and source text relationships. In *Proceedings of the 8th International Natural Language Generation Conference (INLG)*, pages 45–53.

Hassan Shakil, Ahmad Farooq, and Jugal Kalita. 2024. Abstractive text summarization: State of the art, challenges, and improvements. *Neurocomputing*, page 128255.

Shengli Song, Haitao Huang, and Tongxiao Ruan. 2019. Abstractive text summarization using lstm-cnn based deep learning. *Multimedia Tools and Applications*, 78(1):857–875.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2019. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*.

Xin Zhang, Qiyi Wei, Qing Song, and Pengzhou Zhang. 2023. An extractive text summarization model based on rhetorical structure theory. In *2023 26th ACIS International Winter Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD-Winter)*, pages 74–78. IEEE.

Tianyi Zhang, Faisal Ladhak, Esin Durmus, Percy Liang, Kathleen McKeown, and Tatsunori B Hashimoto. 2024. Benchmarking large language models for news summarization. *Transactions of the Association for Computational Linguistics*, 12:39–57.

Yao Zhao, Mikhail Khalman, Rishabh Joshi, Shashi Narayan, Mohammad Saleh, and Peter J Liu. 2022. Calibrating sequence likelihood improves conditional language generation. In *The eleventh international conference on learning representations*.

Chenguang Zhu, Ziyi Yang, Robert Gmyr, Michael Zeng, and Xuedong Huang. 2021. Leveraging lead bias for zero-shot abstractive news summarization. In *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval*, pages 1462–1471.

A Experiments with LLM Prompting

In addition to the two prompts used for generating summaries with the GPT 3.5 Turbo API, we experimented with other prompts which varied in detail. Our goal was to teach the LLM what RST was, so it could be used for summarization. Below is an example of the prompt we wrote with the most detail:

> Summarize this article into approximately 25 to 35 words using Rhetorical Structure Theory. Rhetorical Structure Theory breaks a text into Elementary Discourse Units (EDUs) and organizes them in a binary tree by how they are related. Relations go from a Satellite EDU to a Nucleus EDU meaning that the Satellite builds off of the Nucleus. Segments of multiple EDUs also have relations between each other. There are many relations, and the two most important ones are Attribution and Elaboration. The article input has tags which show the classification of each EDU. The beginning of a Nucleus EDU is marked with: `<Nucleus>`. The end is marked with: `</Nucleus>`. The Satellite relations follow a similar tagging structure. Use these tags to create a summary which includes about 50-80% Nucleus tags and 20-50% Attribution and Elaboration tags. There are also the [CLS] and [SEP] tags which mark the start and end of each sentence respectively. The summary should be short and consist of the most important parts of the text. Make sure the summary produced is coherent and does not end in the middle of a sentence.

Using this detailed prompt with one-shot prompting resulted in the same average Bertscore on the validation set as the prompt we selected for final testing. Moreover, we tested three other prompts with gradually shorter explanations and the Bertscore for each one was within 0.03 of the

selected prompt. Hence, the amount of instruction given to the LLM in regards to RST had an insignificant effect on resulting summary quality.

Four prompts were also used for generating summaries without rhetorical strucutre tags and the best was selected. We use this method as a comparison to the tagged version.

Solving MWPs Using Estimation Verification and Equation Generation

Mitchell Piehl
University of St. Thomas
(Minnesota)
mitchellpiehl@gmail.com

Dillon Wilson
University of Colorado
Colorado Springs
dwilso21@uccs.edu

Jugal Kalita
University of Colorado
Colorado Springs
jkalita@uccs.edu

Abstract

Large Language Models (LLMs) excel at various tasks, including problem-solving and question-answering. However, LLMs often find Math Word Problems (MWPs) challenging because solving them requires a range of reasoning and mathematical abilities with which LLMs seem to struggle. Recent efforts have helped LLMs solve more complex MWPs with improved prompts. This study proposes a methodology that initially prompts an LLM to create equations from a question decomposition, followed by using an external symbolic equation solver to produce an answer. To ensure the accuracy of the obtained answer, inspired by an established recommendation of math teachers, the LLM is further instructed to solve the MWP a second time, but this time with the objective of estimating the correct answer instead of solving it exactly. The estimation is compared to the generated answer to verify the original answer. This approach achieves new state-of-the-art results on datasets used by prior published research on simple numeric and algebraic MWPs. In addition, the approach obtains satisfactory results on trigonometric MWPs, a feat never attempted before to the author's best knowledge. This study also introduces two new datasets, SVAMPClean and Trig300, to help advance the testing of the reasoning abilities of LLMs.

1 Introduction

Solving MWPs has been an area of interest in natural language processing since Bobrow (1964). In recent years, researchers have found many different ways of solving MWPs, including using Seq2Seq methods (Wang et al., 2017), Seq2Tree methods (Wang et al., 2019; Xie and Sun, 2019), Graph2Tree methods (Zhang et al., 2020) or using transformers to directly translate from text to algebraic equations (Griffith and Kalita, 2020; Griffith and Kalita, 2021) and many more. Recently, there has been a noticed surge in the use of LLMs to solve MWPs to achieve high accuracy (Brown et al., 2020), which has been difficult to obtain despite the perceived simplicity of MWPs solved. LLMs often get confused when solving MWPs unless assisted in reasoning steps via appropriate prompts (Shi et al., 2023). To address this challenge, researchers have aided plain LLMs with Chain-of-Thought prompting, which allows the LLM to solve it as a multi-step reasoning task (Wei et al., 2022; Kojima et al., 2022). Among the various state-of-the-art prompt engineering techniques, Progressive Rectification Prompting combined with Chain-of-Thought (Wu et al., 2024a) stands out. This method allows an LLM to rectify its answer by generating responses until the answer passes verification. Another notable technique, BRIDGE, erases the interrogative parts of an MWP, decomposes the rest of the MWP into simpler statements, translates them into equations, and finally solves to get the answer (Wang et al., 2024).

This paper hypothesizes that current methods for solving MWPs, in spite of being aided by designed prompts, improperly rely on precise mathematical accuracy during individual steps of solving. For example, while solving a multiple-step problem, there is a high chance of occurrence of a single mistake along the way, leading to a final incorrect answer. Additionally, very few prompting techniques check for errors as a problem is being solved, and when they do, they rely on precise accuracy again, which is unnecessary and may even

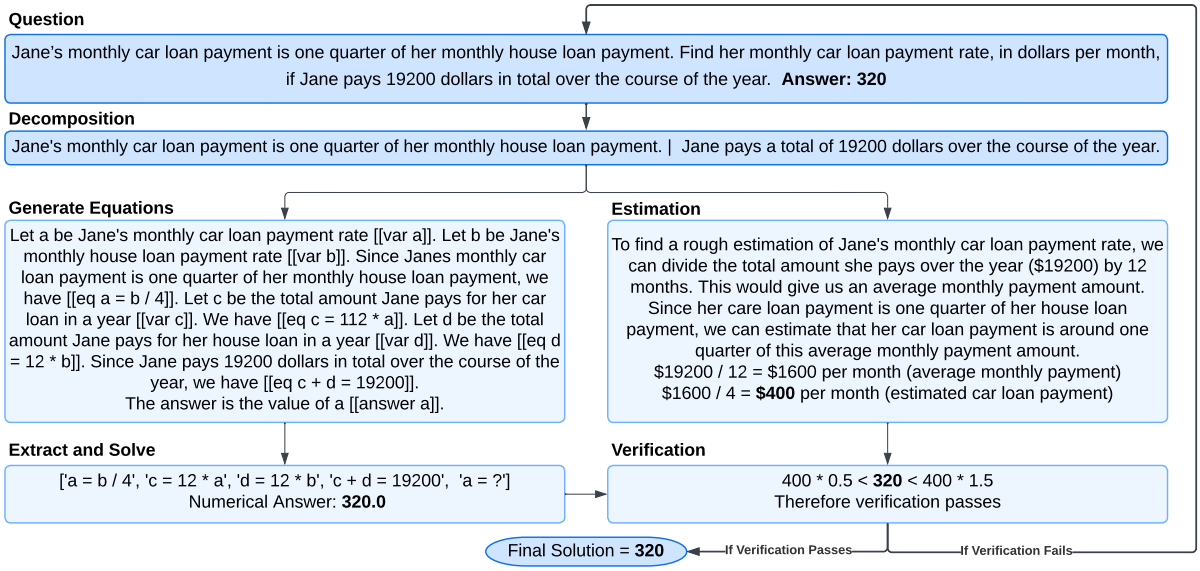


Figure 1: Simplified diagram example of the EVoSS method from the Algebra dataset

be counter-productive.

This paper proposes Equation Verification of Symbolic Solvers (EVoSS). Figure 1 illustrates EVoSS with an example from the Algebra dataset (He-Yueya et al., 2023). The method starts by instructing an LLM to decompose the problem statement and generate algebraic equations. Then, a symbolic solver (external to the LLM) is used to solve the problem precisely. However, any mistake in equations often leads to a significant error in the final crafting of the answer. To ameliorate, the proposed method uses a verification technique that asks the LLM to estimate the answer before committing to the generated answer as final. By comparing the rough estimate with the precise answer, the approach increases the chances of discovering mistakes during equation generation. The introduction of the answer-estimation step during the solving process is inspired by a common practice among teachers of elementary math, supported by publications from the National Council of Teachers of Mathematics, which recommend checking the obtained answer against an estimate when the work was finished (Mathematics, 1964 1969). The approach is as follows.

The first step involves decomposing the question into more straightforward statements and removing details that may hinder correct translation into equations. Once the equations have been generated, a symbolic solver attempts to solve them while fixing any errors.

Once the symbolic solver has computed a numerical value, the LLM is presented with the question again. However, the model is now asked to estimate the correct answer instead of solving it precisely. This estimate serves as a point of comparison with the numerical value derived earlier. If the estimation is aligned with the answer, the verification process confirms with high confidence that the correct answer has been found. If verification fails due to the estimate significantly diverging from the answer obtained by the symbolic solver, a rectification process finds an answer again. During this process, the LLM is presented with the original question and the estimation as a hint to guide the LLM to the correct answer.

This paper makes several key contributions. First, the paper introduces a novel MWP estimation verification scheme that checks answers from generated equations in conjunction with a symbolic solver to verify answers generated by MWPs. This method achieves new state-of-the-art results. Secondly, the paper creates a new trigonometry dataset that is utilized to stress-test the symbolic solver and estimation verification. Finally, this paper finds inadmissible errors in a commonly used dataset and fixes them so that future solvers will be tested fairly.

The rest of the paper starts by discussing related works on math word problem-solving. The next section presents the problem statement, following which the paper introduces the approach that uses answer estimation to verify MWP answers. Next, the paper elaborates upon the ex-

periments performed, the results obtained, and the new datasets that are published for use by future researchers. The final section concludes the paper.

2 Related Works

A number of recent methods use LLMs to solve MWPs. Here are a variety of state-of-the-art methods that use diverse methodologies.

2.1 CoT Prompting Methods

Chain-of-thought prompting methods ask LLMs to generate reasoning paths while solving MWPs to guide the process (Wei et al., 2022). The reasoning path helps the LLM solve the MWP because the LLM leverages the text that describes the reasoning path to get to the final answer (Tan, 2023). Two CoT prompting methods have been proposed recently: zero-shot prompting (Kojima et al., 2022) and few-shot prompting (Wei et al., 2022). Zero-shot CoT prompting instructs the LLM to find the answer without any examples. In contrast, few-shot prompting gives the LLM a few examples illustrating the instructions before asking the LLM to solve. Auto-CoT automatically generates reasoning chains by sampling questions to minimize manual effort and provide more diverse examples (Zhang et al., 2023). Self-consistency (SC) has improved CoT prompting by identifying and selecting the most frequently obtained answer across diverse reasoning paths (Wang et al., 2023b).

2.2 Progressive Rectification Prompting Methods

CoT and self-consistency tend to repeat the same mistakes. Progressive Rectification Prompting (PRP) aims to address these errors by verifying an answer through a process where one numerical value is removed from the MWP, and the large-language model is tasked with identifying the missing value (Wu et al., 2024a). If the LLM does not correctly find the missing value, it allows it to rectify its answer and gives it a hint to help guide it.

2.3 Decomposition Methods

A few methods decompose the question before finding the answer to assist the LLM. For example, the BRIDGE technique achieves high solution accuracy by using a 4-step process that proceeds as follows: erase, decompose, translate, and answer (Wang et al., 2024). Another method, Decomposed Prompting, solves complex tasks using a modular decomposition structure to optimize specific sub-tasks (Khot et al., 2023).

2.4 Methods Using External Solvers

Another common way to assist LLMs in solving MWPs is using an external solver to do the math. A common form of using an external solver is using Program-of-Thoughts prompting (PoT). PoT uses LLMs to generate program statements and then delegates the statements to a program interpreter (Chen et al., 2023). Another example, the Program-aided Language Model or PaL, generates programs as the intermediate reasoning steps and then uses a runtime environment such as a Python Interpreter to avoid mathematical errors by the LLM (Gao et al., 2023). Finally, another state-of-the-art technique, declarative prompting, uses an LLM to formalize word problem variables and equations. This process mitigates arithmetic errors using an external symbolic solver that solves the equations without error (He-Yueya et al., 2023). This method beats PaL by around 20 percent on their new dataset consisting of more complex word problems that we will also use to test our model, called Algebra.

3 Problem Statement

Despite many successes using various methodologies, there is still significant room for improving the effectiveness of LLMs in solving complex math word problems (MWPs). Current methods often suffer due to an increased chance of errors in problems requiring multiple equations and a lack of appropriate validation. In addition, current approaches to MWP solving have not ventured out of simple numerical and algebra problems. Our approach, Estimation Verification of Symbolic Solver Results (EVoSS), aims to develop a model that achieves higher accuracy in solving MWPs, particularly for MWPs that require multiple equations to solve or have large non-rounded numerical values by adding flexibility to the verification process.

4 Approach

This study introduces a novel estimation-based verification technique that validates the answer obtained by solving LLM-generated equations from the LLM-decomposed question using an external

symbolic solver. Specifically, this method takes a given question, decomposes it into different components that are easier to translate into equations, turns the decomposition into equations, finds an answer from the equations (or generates new equations and finds an answer if necessary), and finally finds an estimation from the LLM and compares that to the generated answer to verify and rectify if needed.

4.1 Pre-processing and Decomposition

In pre-processing and decomposition, the question is separated sentence-wise, and the last sentence, the asking part, is removed. Then, the statements and the asking part are presented to the LLM and asked to decompose into statements. A two-shot prompting approach that provides examples of wanted outputs increases accuracy and directs the LLM toward the desired outcome.

4.2 Equation Generation

The LLM is prompted to create equations from the decomposition in a specific format because the decomposition is more straightforward to turn into equations than the original question. We use a set of simple rules to guide the LLM while generating the equations. Some examples of the rules include: "Each equation only uses previously introduced variables" or "The last sentence gives the goal: which variable will contain the answer to the problem." Three-shot prompting, which gives three examples of the decomposition and generated equations, is used here to increase the accuracy of equation generation.

4.3 Solving Equations

The equations generated by the LLM are extracted and appended to an equation list. This list is then sent to the symbolic solver. The generated system of equations is commonly unsolvable due to reasoning issues such as being over-determined or having no unique solutions. To combat these errors, several checks are in place to identify any errors returned by the symbolic solver. If errors are found, the LLM is prompted to generate new equations with a helper statement reminding the LLM of the broken rule. For example, if the last sentence does not give the goal, we will provide the hint: "Make sure your response ends with 'The answer is the value of x [[answer x]].' Where x is the goal variable." If the LLM has been prompted too many times without generating valid equations, the equations are sent to the LLM to solve, and the process moves on. We set the value of the maximum attempts to generate equations to 3 for all tests. Once the symbolic solver has generated a numerical answer that is not an error, the answer is compared to the estimation to be verified.

Our approach specifically uses a symbolic solver to improve accuracy because it has been found that the most significant culprit in large language models getting incorrect answers is semantic misunderstandings of the question (Wang et al., 2023a). Having a symbolic solver solve the generated equations reduces those errors from being made. However, using such a solver increases vulnerability, as any error in equation generation will result in an incorrect answer. Thus, incorporating an additional step of answer verification using estimation improves the possibility of finding the correct result.

4.4 Estimation Verification

When generating equations, two outcomes are possible. Either the LLM is entirely accurate with no mistakes, resulting in the correct answer, or it introduces one or more errors in the equations, usually leading to significantly incorrect answers. For this reason, estimation verification works well.

Inspired by a publication from the National Council of Teachers of Mathematics (Mathematics, 1964 1969), during estimation verification, the LLM is given the question decomposition and prompted to find a rough estimate of the final answer. This approach reduces the likelihood of errors during complex problem-solving. Once the estimation is obtained, it is compared against the answer generated by the symbolic solver. The comparison involves checking whether the estimation is within alpha% of the generated answer. Testing has shown that alpha varies between different datasets ranging from 40 to 50. Table 3 displays the accuracy of various parameter values on the three datasets. If the estimation closely matches the generated answer, the verification passes, indicating confidence in the equations' accuracy. If the estimation is not close to the generated answer, the verification fails, and rectification ensues.

4.5 Rectification

When verification fails, the question will be prompted to the LLM another time, but this time, the estimation will be given as a hint to guide the

LLM. To gain more utility from the hint, the LLM will not be asked to generate equations but instead simply find an answer. Once the answer is generated, if it has been generated in previous steps, it declares that answer as the final solution. If the answer has not been generated, the process returns to the beginning and iterates through each step until it finds a value already generated. An answer will never be accepted if it has only been generated once. For example, when the first answer is generated, it will be appended to list A. Then, when the next answer is generated, it will be compared to A_0. If $A_0 = A_1$ then final answer $F = A_0$. If $A_0 \neq A_1$, then the the LLM will be sent the rectification prompt, and A_2 will be compared to A_0 and A_1. If $A_2 \neq A_0$ and $A_2 \neq A_1$ then the process will loop through the three steps, equation generation, estimation, and rectification prompting, until A_N equals a value from A_0 to $A_N - 1$.

5 Experiments

We test our method using the three most relevant datasets, plus two new datasets and compare results with the most significant recent methods while ensuring comparable implementations.

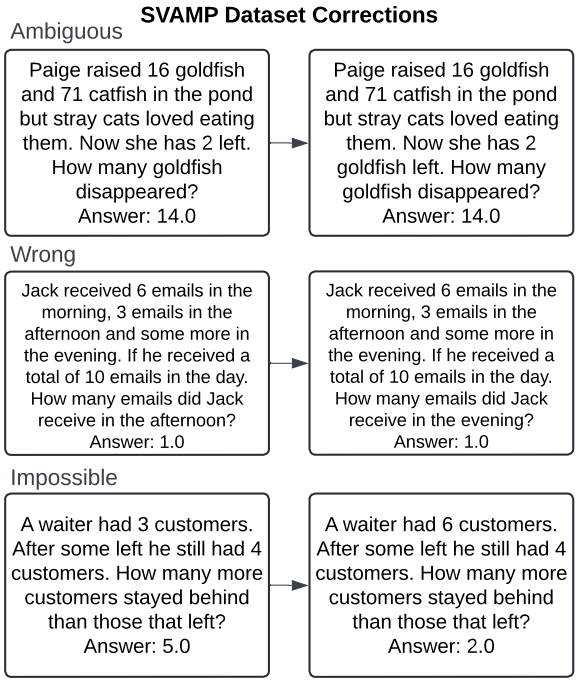


Figure 2: Three examples of corrections made to the SVAMP dataset

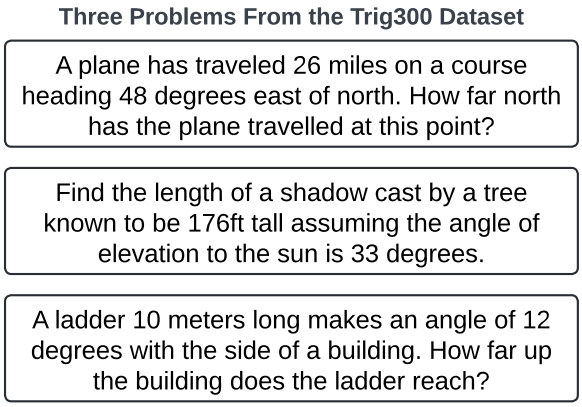


Figure 3: Three examples of questions in the new Trig300 dataset

5.1 Datasets

This work has used three different datasets for testing the proposed method: GSM8k (Cobbe et al., 2021), SVAMP (Patel et al., 2021), and Algebra (He-Yueya et al., 2023). GSM8k is a dataset of 1,319 grade school MWPs commonly used to test MWP solvers. The SVAMP dataset has 1000 math questions similar to the GSM8K dataset, which requires no more than 2 basic arithmetic operations to solve. Algebra is a more recent dataset consisting of 222 algebra-based questions from textbooks made for middle school students instead of elementary students. In addition to these datasets, while analyzing the SVAMP dataset, it was found that many questions were faulty due to the questions being simply wrong, ambiguous, or impossible to occur in real life, so we fixed these errors and ran additional testing on the modified dataset that we call SVAMPClean. This new SVAMPClean dataset is the same as before; however, 50 questions have been improved to make them correct, more solvable, or more specific. The difficulty level of the questions stayed the same as that of the original SVAMP dataset. Figure 2 shows an example of each type of error in the original dataset and how it was fixed.

As MWP solvers and LLMs get more advanced, more challenging datasets will be required to test the reasoning ability and growth of different models and methods, so we created a new dataset called Trig300. This dataset contains 300 trigonometry-based questions that require the solver to go beyond addition, subtraction, multiplication, and division to solve the problem correctly. All 300 questions require the use of one of the trigonometric operations. This dataset also

Method	GSM8K	SVAMP	Algebra	SVAMPClean	Average
EVoSS (Ours)	82.2	**89.4**	**92.8**	**90.5**	**88.73**
I3C	**84.3**	85.8	89.2	88.4	86.93
BRIDGE	77.2	82.3	82.0	–	80.50
PoT	76.3	88.2	64.0	–	76.17
Auto-CoT	78.8	80.9	53.6	–	71.10
PAL	79.5	77.8	–	–	78.65
Zero-Shot-CoT	78.6	82.7	82.8	84.3	82.10
PS	75.9	77.8	–	–	76.85
Manual-CoT	76.4	82.7	77.9	85	80.50
Direct	77.8	80.6	65.3	83.3	76.75

Table 1: Accuracy Comparison of ten methods on four datasets. The highest accuracy per dataset is bold, and the second highest is underlined. The average of the four datasets is also shown.

tests how estimation verification works on questions beyond addition, subtraction, multiplication, and division. To create this dataset, each question was hand-crafted to mimic various questions found in trigonometry textbooks and then manipulated to ask for different parts based on the information provided. This ensures the dataset tests the solver's diverse solving skills. For example, the question, "If a treadmill is 48 inches long and is set to an incline of 10 degrees. How much higher is the front of the treadmill than the back?" will be changed to also ask for the angle of incline of the treadmill given the length and height increase, and it will also ask for the length of the treadmill given the incline and height increase. All answers are rounded to the nearest thousandth if necessary. Examples of problems from the Trig300 dataset are shown in Figure 3.

5.2 Baselines

The results are compared with nine current state-of-the-art methods: Auto Chain of Thought (Zhang et al., 2023), BRIDGE (Wang et al., 2024), Direct (Kojima et al., 2022), Identify and Ignore Irrelevant Conditions (I3C) (Wu et al., 2024b), Manual Chain of Thought (CoT) (Wei et al., 2022), Plan-and-Solve (PS) (Wang et al., 2023a), Program-Aided Learning (PAL) (Gao et al., 2023), Program of Thought (PoT) (Chen et al., 2023), and Zero-Shot Chain of Thought (Kojima et al., 2022). We will also compare our estimation verification technique to self-consistency (SC) (Wang et al., 2023b).

5.3 Implementation

This method uses the gpt-3.5-turbo API as the backend large language model[1]. The gpt-3.5-turbo API is widely accessible and represents one of the most advanced large language models available. It has also been used to test many current state-of-the-art methods. We set the temperature to 0.4 and the max tokens to 256.

A few-shot reasoning approach is used for question decomposition and equation generation, which provides examples of questions and responses to help guide the LLM. Similar to the Declarative (He-Yueya et al., 2023) and BRIDGE (Wang et al., 2024) prompting methods, the equations generated are solved using SymPy (Meurer et al., 2017), a Python library for symbolic mathematics.

5.4 Metric

We evaluate our answers using the exact match metric. This means we count our answers as correct if the answer matches exactly. However, due to round-off errors and the Trig300 dataset answers being rounded to the nearest thousandth, we count our answers as correct if the generated answer is within 0.001 of the correct answer.

5.5 Results

Table 1 reports the accuracy of our method across the specified datasets. Our novel approach achieves state-of-the-art performance on the Algebra, SVAMP, and SVAMPClean datasets, achieving accuracies of 92.8, 89.4, and 90.5 percent, respectively. These results beat its nearest competitor by around 3 percent across all three datasets.

[1]The public API is available at https://openai.com/api/

Question:	A monthly pass for light rail is 1.2 times cheaper than a bundle with 20 day passes. Find the price of the monthly pass if the bundle costs $10 more than the monthly pass. Answer: 50
Direct	The answer is 30. **Incorrect**
I3C	Step 1: Let x be the price of the monthly pass. Step 2: Since the bundle with 20 day passes costs $10 more than the monthly pass, the price of the bundle is x + $10. Step 3: According to the given condition, the monthly pass is 1.2 times cheaper than the bundle with 20 day passes, so we can write the equation: x = 1.2(x + $10). Step 4: Now we can solve for x: x = 1.2x + $12, 0.2x = $12, x = $60. Step 5: Therefore, the price of the monthly pass is $60. **Incorrect**
Zero-shot CoT	1. Let the price of the monthly pass be M. 2. Since the bundle costs $10 more than the monthly pass, the price of the bundle is M + 10. 3. According to the statement, the monthly pass is 1.2 times cheaper than the bundle with 20 day passes. This means: M = 1.2(M + 10). 4. Solving the equation: M = 1.2M + 12, 0.2M = 12, M = 60. 5. Therefore, the price of the monthly pass is $60. **Incorrect**
Equation Generation (Ours)	Let a be the price of the monthly pass [[var a]]. Let b be the price of the bundle with 20 day passes [[var b]]. We have [[eq b = a + 10]]. Since the bundle costs $10 more than the monthly pass, we have [[eq b = 1.2 * a]]. The answer is the value of a [[answer a]]. **Correct**
Estimation (Ours)	To find a rough estimation, we can assume the monthly pass costs $x. According to the first statement, the bundle with 20 day passes costs 1.2 times more, so it would be 1.2x. And according to the second statement, the bundle costs $10 more than the monthly pass, so 1.2x = x + 10. Solving for x: 1.2x = x + 10, 0.2x = 10, x = 50. Therefore, the price of the monthly pass is roughly $50. **Correct**

Table 2: Case study of various answers generated by the listed methods. The question is underlined, and the responses generated by our method are in bold.

Method	GSM8K	SVAMP	Algebra
EVoSS	**82.2**	**89.4**	**92.8**
SS	69.9	80.8	79.7
EV	74.1	85.0	76.1

Table 3: Ablation study results comparing the use of the Symbolic Solver with Estimation Verification (EVoSS) to the use of the Symbolic Solver with decomposition (SS) and the use of estimation verification technique without the symbolic solver.

α	GSM8K	SVAMP	Algebra
40	78.0	85.3	**88.3**
50	**80.0**	**87.9**	87.8
60	79.0	83.9	87.3

Table 4: Accuracies when changing the percentage of how close the estimation needs to be away from the answer to pass verification. Temperature was set to 0.7 for all tests.

Our state-of-the-art results achieve an accuracy of 88.73 percent across the four datasets, beating the next-best average by almost two percent.

5.6 SVAMPClean Dataset

After the inadmissible errors were corrected in the SVAMP dataset, all methods tested received a higher accuracy with SVAMPClean than the original SVAMP dataset. The improvements from SVAMP to SVAMPClean are consistently between one and three percent, coherent with the five percent of questions being modified.

5.7 Ablation Studies

In addition to the presented results, an ablation study was performed on the datasets with and without estimation verification and a symbolic solver. These findings are summarized in Table 3. The results show improvements from eight to thirteen percent using estimation verification across all three datasets tested, demonstrating how our novel estimation verification can drastically improve equation generation solvers.

Testing was also done to see how results are affected when the accuracy needed to pass verification changes. Table 4 presents these results. The results show that the optimal α value while using the Algebra dataset was 40, while the optimal α value using the GSM8K and SVAMP datasets was 50.

In addition to the tests above, we have obtained the MWP solver's accuracies using different LLM temperatures. The results are shown in Table 6. This method is most accurate when the LLM temperature is set to 0.4 in all three datasets tested.

Finally, we compared our estimation verification technique to a current state-of-the-art technique, self-consistency (SC). Self-consistency is the act of repeatably finding the answer to the same question *M* times and selecting the final answer as the most frequent answer of all the generated answers (Wang et al., 2023b). We set *M* to 5 for our Self-Consistency of Symbolic Solver (SCoSS) experiments. Our results show using estimation to check answers generated has two to seven percent higher accuracy than using self-

Question:	Pacing off 150 ft. on flat ground from the base of a tree, a forester measures the angle of elevation to the top of the tree as 40○. What is the height of the tree? Answer: 125.865
Equation Generation	Let a be the distance from the base of the tree [[var a]]. We have [[eq a = 150]]. Let b be the height of the tree [[var b]]. Let c be the angle of elevation to the top of the tree in radians [[var c]]. We have [[eq c = 40 * pi / 180]]. Since the angle of elevation to the top of the tree forms a right triangle with the distance and the height, we have [[eq tan(c) = b / a]]. The answer is the value of b [[answer b]]. **Correct**
Estimation	To find a rough estimation of the height of the tree, we can use the trigonometric function tangent. Tangent(angle) = opposite/adjacent, Tangent(40 degrees) = height /150 ft, height = 150 ft * tangent(40 degrees), height ĩ150 ft * 0.84 ĩ25 ft. Therefore, the rough estimation of the height of the tree is approximately 125 feet. **Correct**

Table 5: Case study and example of a problem being solved using our methods from an example of the new Trig300 dataset.

Temperature	GSM8K	SVAMP	Algebra
0.7	80	87.9	88.3
0.4	**82.2**	**89.4**	**92.8**
0.1	80.8	89.2	88.7

Table 6: Accuracies when changing the temperature of the LLM

Method	GSM8K	SVAMP	Algebra
EVoSS	**82.2**	**89.4**	**92.8**
SCoSS	75.1	86.6	90.0

Table 7: Comparing our method of Estimation Verification to using the symbolic solver with self-consistency

consistency while using a symbolic solver. Estimation verification is superior to self-consistency due to the tendency of the LLM to get confused in the same ways when solving the question repeatedly. Using estimation to verify answers generated creates diverse solving paths and simplifies the problem, leading to more accurate results. The results are shown in Table 7.

5.8 Case Study

Table 2 presents a case study of various answers generated by a real example from the Algebra dataset. As questions become layered with more statements, standard solving techniques, such as the ones shown, cannot handle the multiple layers well and get the answer incorrect. However, solving the question by creating equations to be solved for a symbolic solver, or simplifying the question by estimating, both of which are used in our method, leads to a correct answer generated.

Method	Trig300
EVoSS	**65.5**
Zero Shot-CoT	10.7
Manual-CoT	10
Direct	4

Table 8: Accuracy percentage results of 4 different methods tested on the Trig300 dataset

5.9 Trig300 Dataset

We ran testing of our new Trig300 dataset on our method and three other baseline methods. Our method achieved 65.5 percent accuracy, compared to 4 - 11 percent accuracy on the other baseline methods. The obtained accuracy ranging between 4 and 66 percent across the four methods compared to the other datasets ranging from 53 to 93 percent, shows how this new dataset is significantly more challenging for LLMs, thus providing an improved way of testing the reasoning abilities of LLMs as they continue to get more advanced. In addition, our method beating the other baseline methods by 50 to 60 percent, shows how our method is superior to other baseline models, especially when solving more complex math problems. Additionally, this shows how estimation verification can still be successfully implemented on other problems beyond the four basic operators: addition, subtraction, multiplication, and division. Table 8 shows our results, and Table 5 shows an example of a question in the new dataset and a case study of the question being solved using equation generation and estimation.

6 Potential Risks and Downfalls

There are a couple of potential risks and downfalls associated with this study. The widespread use of LLMs has transformed the educational world, changing how students solve their assignments and search for information. While improving LLMs may help students access better-quality information, it will also enhance their abilities to not do their school work and instead rely on using LLMs to solve problems.

However, one potential downfall of this study mitigates that potential risk. As LLMs continue to evolve and improve, they may naturally become better at solving MWPs without the help of outside methods such as the proposed method. For example, GPT-4 solved MWPs from the GSM8K dataset with nearly 35 percent higher accuracy than GPT-3.5. (Achiam et al., 2023). It is important to note that as performance improves, datasets of more complex word problems from various domains will have to be developed to test the mathematical and reasoning abilities of the LLMs.

7 Conclusion

This study presents a novel estimation verification technique that verifies solutions obtained from an improved symbolic solver called EVoSS. This method leverages question decomposition to facilitate more accurate equation generation so that a symbolic solver can be used. The answer is then verified using an estimation verification technique that prompts the model to find an estimate and compares it to the obtained answer. The estimation verification technique improves results by eight to thirteen percent, helping this method solve math word problems with state-of-the-art accuracy. Additionally, this paper fixes a popular dataset, SVAMP, by removing inadmissible errors and ambiguities. We call the new dataset SVAMPClean. Finally, this paper creates a new, more complex trigonometry dataset to prepare for the advancement of various methods and models and to test if the estimation verification works beyond the simple MWPs based on the four operators: addition, subtraction, multiplication, and division.

8 Acknowledgements

All work herein reported is supported by the National Science Foundation under Grant No. 2349452. Any opinion, finding, or conclusion in this study is that of the authors and does not necessarily reflect the views of the National Science Foundation. We thank Ananya Kalita for her beneficial help with creating the presented Trig300 dataset.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Daniel G. Bobrow. 1964. Natural language input for a computer problem solving system. Technical report, Massachusetts Institute of Technology, USA.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR.

Kaden Griffith and Jugal Kalita. 2020. Solving arithmetic word problems using transformer and preprocessing of problem texts. In *Proceedings of the 17th International Conference on Natural Language Processing (ICON)*, pages 76–84.

Kaden Griffith and Jugal Kalita. 2021. Solving arithmetic word problems with transformers and preprocessing of problem text. *arXiv preprint arXiv:2106.00893*.

Joy He-Yueya, Gabriel Poesia, Rose E Wang, and Noah D Goodman. 2023. Solving math word problems by combining language models with symbolic solvers. *arXiv preprint arXiv:2304.09102*.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2023. Decomposed prompting: A modular approach for solving complex tasks. In *Proceedings of the International Conference on Learning Representations (ICLR)*. Camera-ready version.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in Neural Information Processing Systems*, 35:22199–22213.

National Council Of Teachers Of Mathematics. 1964-1969. Topics in mathematics, hints for problem solving, booklet 17.

Aaron Meurer, Christopher P Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K Moore, Sartaj Singh, et al. 2017. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094, Online, June. Association for Computational Linguistics.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H Chi, Nathanael Schärli, and Denny Zhou. 2023. Large language models can be easily distracted by irrelevant context. In *International Conference on Machine Learning*, pages 31210–31227. PMLR.

Juanhe TJ Tan. 2023. Causal abstraction for chain-of-thought reasoning in arithmetic word problems. In *Proceedings of the 6th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pages 155–168.

Yan Wang, Xiaojiang Liu, and Shuming Shi. 2017. Deep neural solver for math word problems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 845–854.

Lei Wang, Dongxiang Zhang, Jipeng Zhang, Xing Xu, Lianli Gao, Bing Tian Dai, and Heng Tao Shen. 2019. Template-based math word problem solvers with recursive neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7144–7151.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023a. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. *arXiv preprint arXiv:2305.04091*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023b. Self-consistency improves chain of thought reasoning in language models. In *Proceedings of the International Conference on Learning Representations (ICLR)*. Camera-ready version.

Dingzirui Wang, Longxu Dou, Wenbin Zhang, Junyu Zeng, and Wanxiang Che. 2024. Exploring equation as a better intermediate meaning representation for numerical reasoning of large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19116–19125.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837.

Zhenyu Wu, Meng Jiang, and Chao Shen. 2024a. Get an a in math: Progressive rectification prompting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19288–19296.

Zhenyu Wu, Chao Shen, and Meng Jiang. 2024b. Instructing large language models to identify and ignore irrelevant conditions. In *Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Zhipeng Xie and Shichao Sun. 2019. A goal-driven tree-structured neural model for math word problems. In *Ijcai*, pages 5299–5305.

Jipeng Zhang, Lei Wang, Roy Ka-Wei Lee, Yi Bin, Yan Wang, Jie Shao, and Ee-Peng Lim. 2020. Graph-to-tree learning for solving math word problems. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3928–3937.

Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. 2023. Automatic chain of thought prompting in large language models. In *The Eleventh International Conference on Learning Representations (ICLR 2023)*.

Author Index