Titre: Title:	Using Language Models for Knowledge Mining of Software Logs
Auteur: Author:	Vithor Gomes Ferreira Bertalan
Date:	2026
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Gomes Ferreira Bertalan, V. (2026). Using Language Models for Knowledge Mining of Software Logs [Ph.D. thesis, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/76206/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/76206/
Directeurs de recherche: Advisors:	Daniel Aloise
Programme: Program:	Génie informatique

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Using Language Models for Knowledge Mining of Software Logs

VITHOR GOMES FERREIRA BERTALAN
Département de génie informatique et génie logiciel

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Génie informatique

Avril 2026

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Cette thèse intitulée :

Using Language Models for Knowledge Mining of Software Logs

présentée par **Vithor Gomes Ferreira BERTALAN**
en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

Heng LI, président
Daniel ALOISE, membre et directeur de recherche
Amal ZOUAQ, membre
Eraldo FERNANDES, membre externe

DEDICATION

To Eliana,
for all the love and support,
and to Levi,
to you the future belongs.

ACKNOWLEDGEMENTS

My most sincere gratitude goes to my supervisor, Professor Daniel Aloise. I am truly grateful for the opportunity to work under your guidance. Thank you for your constant support, insightful advice, and for the knowledge shared throughout this journey. It has been a privilege to learn from you.

I want to express my deepest thanks to my family, especially my wife, Eliana, and my son, Levi. Your constant love and constant support were my greatest sources of motivation. I am eternally grateful for your patience and for being by my side every step of the way.

To my friends, colleagues and previous professors at Polytechnique Montréal, I extend my warmest thanks. The shared ideas and moments we spent together were fundamental to the completion of this work.

Finally, I would like to express my appreciation to the members of the thesis jury Prof. Heng Li, Prof. Amal Zouaq and Prof. Eraldo Fernandes for accepting the invitation to evaluate this work and for their valuable contributions to this research.

RÉSUMÉ

Dans le contexte des systèmes logiciels modernes, les journaux (logs) sont générés en quantités considérables et constituent une source essentielle d'information pour la surveillance, le débogage et la fiabilité des systèmes à grande échelle. Ces logs enregistrent des événements hétérogènes produits par de multiples composants interconnectés fonctionnant simultanément dans des conditions complexes et dynamiques. Bien qu'ils offrent des informations précieuses sur le comportement du système, leur volume important, leur nature semi-structurée et l'absence de sémantique explicite rendent leur analyse automatisée difficile, en particulier pour des tâches telles que la compréhension du système et l'analyse des causes profondes.

À mesure que les systèmes logiciels gagnent en taille et en complexité, l'inspection manuelle des logs devient de plus en plus impraticable. Cela a motivé le développement de techniques automatisées visant à structurer, résumer et analyser les logs afin d'en extraire des connaissances exploitables. Les avancées récentes en traitement automatique du langage naturel et en modèles de langage ont ouvert de nouvelles perspectives pour l'analyse des logs en permettant l'apprentissage de représentations contextuelles et sémantiques directement à partir des données brutes. Toutefois, l'exploitation efficace de ces modèles pour des tâches spécifiques aux logs demeure un défi en raison des particularités structurelles des logs et de la diversité des objectifs d'analyse.

Cette thèse étudie l'utilisation de techniques fondées sur les données et sur les modèles de langage pour l'analyse des logs logiciels. Les trois articles proposés constituent une investigation exploratoire de la manière dont les modèles de langage peuvent être appliqués aux différentes étapes de l'analyse des logs, en abordant des défis complémentaires liés au parsing des logs, à leur résumé par modélisation thématique et à l'inférence causale dans les logs.

La première contribution traite du problème du parsing des logs dans les systèmes logiciels à grande échelle. Nous proposons un modèle de parsing basé sur les transformeurs spécialisés qui extrait automatiquement des gabarits structurés à partir de messages de logs bruts en exploitant des représentations contextuelles apprises à partir de séquences de logs. Contrairement aux approches traditionnelles de parsing reposant sur des règles conçues manuellement ou des heuristiques fixes, la méthode proposée tire parti de la puissance expressive des architectures de type transformeur afin d'améliorer la robustesse face à la variabilité et au bruit présents dans les données de logs. Les résultats expérimentaux obtenus sur des ensembles de données réels montrent que l'approche proposée atteint des performances compétitives tout en offrant une plus grande flexibilité et une meilleure capacité de généralisation.

La deuxième contribution porte sur le résumé des logs par modélisation thématique. Nous proposons un cadre de résumé fondé sur le regroupement (clustering) qui intègre des représentations issues de grands modèles de langage avec des informations structurelles propres aux logs, notamment les variables extraites lors du parsing et la proximité positionnelle entre les lignes de logs. En combinant ces signaux hétérogènes dans une formulation unifiée de dissimilarité et en appliquant une modélisation thématique basée sur les transformeurs, la méthode proposée est capable d'identifier des thèmes cohérents et de sélectionner des lignes de logs représentatives résumant des fichiers volumineux et hétérogènes. Des évaluations expérimentales approfondies sur plusieurs jeux de données de référence montrent que l'approche proposée surpasse de manière systématique les techniques de résumé de logs à l'état de l'art.

La troisième contribution aborde le problème de l'inférence causale dans les logs logiciels. Nous proposons un nouveau cadre de découverte causale qui exploite des modèles de langage causaux pour estimer les probabilités conditionnelles entre événements de logs et intègre ces estimations dans un processus de découverte causale fondé sur des contraintes. En remplaçant les statistiques de fréquence brutes par des estimations probabilistes issues des modèles de langage et en utilisant l'information mutuelle conditionnelle pour les tests d'indépendance, l'approche proposée permet une inférence causale à la fois évolutive et précise dans des données de logs clairsemées et de grande dimension. Les résultats expérimentaux sur des logs industriels réels démontrent que la méthode proposée surpasse significativement les techniques classiques de découverte causale tant en termes de précision que de passage à l'échelle.

Dans son ensemble, cette thèse propose une investigation complète des méthodes fondées sur les modèles de langage pour l'analyse des logs logiciels. En abordant conjointement le parsing, le résumé et l'inférence causale, les travaux proposés montrent comment les modèles de langage peuvent progressivement enrichir le niveau de structuration, d'abstraction et de raisonnement appliqué aux logs. Les résultats contribuent à faire progresser l'état de l'art en analyse informée des logs et fournissent des bases pratiques pour améliorer la surveillance, le débogage et le diagnostic dans les systèmes logiciels complexes.

ABSTRACT

In the context of modern software systems, logs are generated in vast quantities and provide critical input for monitoring, debugging, and reliability purposes in large-scale systems. These logs record heterogeneous events produced by multiple interconnected components operating concurrently under complex and dynamic conditions. Although logs provide valuable insights into system behavior, their large volume, semi-structured nature, and lack of explicit semantics make automated analysis challenging, particularly for tasks such as system understanding and root cause analysis.

As software systems continue to grow in scale and complexity, manual log inspection becomes increasingly impractical. This has motivated the development of automated techniques for structuring, summarizing, and analyzing logs in order to extract actionable knowledge. Recent advances in natural language processing and language models have opened new opportunities for log analysis by enabling the learning of contextual and semantic representations directly from raw log data. However, leveraging these models effectively for log-specific tasks remains a challenge due to the structural peculiarities of logs and the diversity of analysis objectives.

This thesis investigates the use of data-driven and language-model-based techniques for the analysis of software logs. The three proposed articles form an exploratory investigation of how language models can be applied to different stages of log analysis, addressing complementary challenges related to log parsing, log summarization through topic modeling, and causal inference in logs.

The first contribution addresses the problem of log parsing in large-scale software systems. We propose a specialized transformer-based log parsing model that automatically extracts structured templates from raw log messages by leveraging contextual representations learned from log sequences. Unlike traditional parsing approaches that rely on handcrafted rules or fixed heuristics, the proposed method exploits the expressive power of transformer architectures to improve robustness to variability and noise in log data. Experimental results on real-world log datasets demonstrate that the proposed approach achieves competitive performance while offering greater flexibility and generalization capability.

The second contribution focuses on log summarization through topic modeling. We propose a clustering-based summarization framework that integrates large language model representations with log-specific structural information, including parsed variables and positional proximity between log lines. By combining these heterogeneous signals into a unified dissim-

ilarity formulation and applying transformer-based topic modeling, the proposed method is able to identify coherent topics and select representative log lines that summarize large and heterogeneous log files. Extensive experimental evaluations on multiple benchmark datasets show that the proposed approach consistently outperforms state of the art log summarization techniques.

The third contribution addresses the problem of causal inference in software logs. We propose a novel causal discovery framework that leverages causal language models to estimate conditional probabilities among log events and integrates these estimates into a constraint-based causal discovery process. By replacing raw frequency statistics with probability estimates derived from language models and by using conditional mutual information for independence testing, the proposed approach enables scalable and accurate causal inference in sparse and high-dimensional log data. Experimental results on real-world industrial logs demonstrate that the proposed method significantly outperforms classical causal discovery techniques in terms of both accuracy and scalability.

Collectively, this thesis provides a comprehensive investigation of language-model-based methods for software log analysis. By jointly addressing log parsing, log summarization, and causal inference, the proposed works demonstrate how language models can progressively enhance the level of structure, abstraction, and reasoning applied to logs. The results contribute to advancing the state of the art in informed log analysis and provide practical foundations for improved system monitoring, debugging, and diagnosis in complex software systems.

TABLE OF CONTENTS

DEDICATION iii

ACKNOWLEDGEMENTS iv

RÉSUMÉ v

ABSTRACT vii

LIST OF TABLES xiii

LIST OF FIGURES xiv

LIST OF SYMBOLS AND ACRONYMS xv

CHAPTER 1 INTRODUCTION 1
1.1 Thesis Motivation 1
1.1.1 Log Parsing 2
1.1.2 Log Topic Modeling 3
1.1.3 Log Causality 4
1.2 Research Objetives 6
1.3 Thesis Outline 8
1.4 Publications 8

CHAPTER 2 LITERATURE REVIEW 10
2.1 Creating and Using Logs 10
2.2 Log Parsing 12
2.2.1 Log Parsing Techniques 15
2.2.2 Current Trends in Log Parsing 17
2.3 Log Mining 18
2.3.1 Log Classification 20
2.4 Topic Modeling 22
2.4.1 BerTopic 23
2.5 Attention and Transformer Networks 25
2.5.1 Attention and Self-Attention 25
2.5.2 Transformers 27
2.6 Large Language Models 32
2.7 Causal Structure Learning 34

2.7.1 Causal Structures . . . 36
2.7.2 Large Language Models for Causality . . . 38
2.8 Detection of Causality in Log Files . . . 39
2.8.1 Methods for Tracking Causality . . . 40
2.8.2 Causality Identification with Machine Learning . . . 41
2.8.3 Large Language Models for Log Causality Detection . . . 42

CHAPTER 3 OVERVIEW . . . 45
3.1 Gaps in the Literature . . . 45
3.2 Steps of the Research . . . 46

CHAPTER 4 ARTICLE 1: USING TRANSFORMER MODELS AND TEXTUAL ANALYSIS FOR LOG PARSING . . . 47
4.1 Introduction . . . 48
4.2 Background and Motivation . . . 51
4.2.1 Limitations of Tradicional Log Parsing . . . 51
4.2.2 Transformer Models . . . 52
4.2.3 Practical Applications of Transformer-Based Log Parsing . . . 53
4.3 Methodology . . . 54
4.3.1 Data collection . . . 54
4.3.2 Application of regular expressions . . . 55
4.3.3 Data transformation . . . 56
4.3.4 Text processing . . . 56
4.3.5 Clustering data . . . 57
4.3.6 Creation of frequency dictionaries . . . 59
4.3.7 Verification of English vocabulary . . . 60
4.4 Empirical Evaluation and Results . . . 61
4.4.1 Evaluation using LogHub metrics . . . 61
4.4.2 Flaws in LogHub ground truths parsing . . . 64
4.4.3 Flaws in LogHub ground truths grouping . . . 65
4.5 Practical Application - Lessons Learned . . . 65
4.6 Related Work . . . 67
4.6.1 Log parsing . . . 67
4.6.2 Transformers for log analysis . . . 68
4.7 Conclusion . . . 68

CHAPTER 5 ARTICLE 2: CLUSTERING TEXTUAL FEATURES FOR LOG SUMMARIZATION IN LARGE SOFTWARE SYSTEMS 69
5.1 Introduction 69
5.2 Related Work 73
5.3 Methodology 76
5.3.1 Collecting Data 76
5.3.2 Computing dissimilarity between log lines 76
5.3.3 Clustering log lines 81
5.3.4 Bertopic modeling from clustered log lines 82
5.3.5 Choosing the log summaries 83
5.4 Empirical Evaluation 84
5.4.1 Summarization Metrics - Rouge Score 84
5.4.2 Evaluating the unified matrix M for log line clustering 86
5.4.3 Evaluating the Log Summarizations Against the State-of-the-Art . . 91
5.5 Practical Application 93
5.6 Conclusion 94

CHAPTER 6 ARTICLE 3: EXTRACTING CAUSAL RELATIONS FROM LOG SEQUENCES USING CAUSAL LANGUAGE MODELS 98
6.1 Introduction 98
6.2 Bibliographical Review 101
6.3 Methodology 103
6.3.1 Representation of Log Sequences 106
6.3.2 Conditional Probability Distribution (CPD) Extraction 106
6.3.3 Conditional Mutual Information for Independence Testing 108
6.4 Experimental Evaluation 109
6.4.1 Datasets 109
6.4.2 Baselines 109
6.4.3 Implementation 110
6.5 Results 111
6.5.1 PC results 112
6.5.2 Granger results 113
6.5.3 Our method's results - LLM-based CMI 113
6.6 Threats to Validity and Limitations 114
6.6.1 Construct validity 114
6.6.2 Internal validity 114

6.6.3 External validity 114
6.7 Industry Application 115
6.8 Conclusion 116

CHAPTER 7 CONCLUSION 118
7.1 Summary of Works 118
7.1.1 Article 1 118
7.1.2 Article 2 119
7.1.3 Article 3 119
7.2 Limitations 120
7.3 Future Research 121
7.4 Final Remarks 122

REFERENCES 123

LIST OF TABLES

Table 4.1	Classification of logs into log templates	49
Table 4.2	Examples of log styles from LogHub	53
Table 4.3	Frequency dictionary for the last cluster of Figure 4.3	59
Table 4.4	Accuracies of the methods using LogHub's framework. Top values in each column in bold.	62
Table 4.5	F1 measures of the methods using LogHub's framework. Top values in each column in bold.	62
Table 4.6	Precisions of the methods using LogHub's framework. Top values in each column in bold.	63
Table 4.7	Recalls of the methods using LogHub's framework. Top values in each column in bold.	63
Table 4.8	Examples of ground truth lines from LogHub	65
Table 4.9	Examples of ground truth lines from LogHub that could be joined	66
Table 5.1	Example of log lines	77
Table 5.2	Example of parsed lines	79
Table 5.3	Example of boolean matrix F for the parsed variable tokens	79
Table 5.4	Example of a closeness matrix	80
Table 5.5	Example of topic generated by BERTopic from the `BGL` file.	83
Table 5.6	Example of summary references and hypothesis	85
Table 5.7	Example of summary metrics using the texts from Table 5.6	85
Table 5.8	Example of ground truth summaries	87
Table 5.9	Comparison of lexicon of log datasets	87
Table 5.10	Best combinations of α, β, and γ for the unified matrix M in each tested dataset	89
Table 5.11	F1 ROUGE scores of our method against baselines A and B	90
Table 5.12	Execution time for the algorithms	91
Table 5.13	F1 Scores for the summarization methods	92
Table 6.1	Example of causal links between logs	99
Table 6.2	Comparison of required independence tests and the number of edges	112
Table 6.3	Results obtained by each approach	112

LIST OF FIGURES

Figure 1.1	Illustration of the log parsing process.	3
Figure 1.2	Illustration of the log topic modeling process.	4
Figure 1.3	Example of causal relationships between log lines. Blue arrows represent inferred causal dependencies.	5
Figure 2.1	Parsing of a raw log to a structured log	13
Figure 2.2	Log text preprocessing techniques	14
Figure 2.3	BerTopic workflow	23
Figure 2.4	Simplified Model of an Encoder-Decoder Architecture	28
Figure 2.5	Architecture of a transformer.	30
Figure 2.6	Example of the Attention Mechanism in Transformers	32
Figure 2.7	Causal structure resulting from the set of equations 2.8, 2.9, and 2.10.	36
Figure 4.1	Methodology of the parsing process.	55
Figure 4.2	Example of sentence processed with our custom tokenizer	57
Figure 4.3	Example of clustering raw log lines.	58
Figure 4.4	Raw HDFS dataset log data.	58
Figure 4.5	Clustered HDFS dataset log data using HDBSCAN.	59
Figure 5.1	Sequential workflow of the proposed log summarization method.	77
Figure 6.1	Comparison of workflows: PC and LLM-augmented PC	104
Figure 6.2	Example of a $\mathcal{P}_{x \to y}$ calculation	107

LIST OF SYMBOLS AND ACRONYMS

BERT	Bidirectional Encoder Representations from Transformers
CI/CD	Continuous Integration/Continuous Deployment
CLM	Causal Language Model
DAG	Directed Acyclic Graph
DevOps	Development and Operations
GPU	Graphical Processing Unit
LLM	Large Language Model
NLP	Natural Language Processing
TPU	Tensor Processing Unit

CHAPTER 1 INTRODUCTION

1.1 Thesis Motivation

Software systems have become increasingly complex, distributed, and highly automated, leading to a significant increase in the volume of data collected during their execution. Within this body of data, software log lines have played a crucial role for monitoring and controlling software systems, troubleshooting errors, and measuring system performance, becoming one of the main sources of information for understanding system reliability in modern computing systems.

Despite their importance, software log lines have become increasingly difficult to analyze. Indeed, modern software systems are able to generate millions of log lines per minute, making it difficult to identify those log lines that are relevant for analysis and understanding their relationships. Capturing those connections between log lines has become one of the fundamental requirements for effectively troubleshooting software systems. As an example, one error message can easily trigger other log lines across different components and systems, making it difficult to establish the root cause of an error or problem.

In the course of analyzing software logs, it is necessary to perform a profound research of the domain and the software under consideration. This is due to the fact that software logs are not generated independently but are closely related to the software's architecture, deployment, and the environment in which the software is executed. The stability of a continuous integration/continuous deployment (CI/CD) pipeline needs to be analyzed with a high degree of precision, as the frequent release and configuration changes can significantly influence the software logs.

Furthermore, it is mandatory to provide a precise investigation of the periodicity and goal of the software logs, along with the software events that the logs are expected to capture. For instance, when dealing with causal reasoning, it is not adequate to rely solely on the time stamps, as the software can use multithreading and concurrent execution, resulting in interleaved software logs that do not correspond to the linear execution sequence. However, in the current industrial landscape of computer systems, the analysis of logs is still a manual process, taking considerable time by analysts and engineers.

As more time is spent analyzing logs, less time is spent on activities that actually create business value, such as feature development or optimization [1]. Moreover, the ever-increasing volume of logs in a running server environment makes such problems even more complex

[2]. Programmers also spend a considerable amount of time in bug remediation, a process that could be much shorter if log entries that indicate root causes were detected earlier in the debugging process [3]. Hence, the inefficient analysis of logs has a direct impact on productivity, reliability, cost, and potential impact.

Motivated by these challenges, this work aims to investigate software log data by developing log analysis methods based on language models. In this endeavor, we explore three main areas of log analysis: *log parsing*, *log topic modeling*, and *log causality*. In the following three sections, we examine each of these areas in greater detail.

1.1.1 Log Parsing

Log parsing is the transformation of raw, textual log messages into a more structured and normalized format. In practice, the goal of log parsing is to identify the invariant portion of a log message, which is also called the *log template*, and separate it from the variable parts of the log messages, which may include identifiers, parameters, timestamps, or any other variable information.

By abstracting the variable information from the underlying structure of the log messages, log parsing significantly reduces the dimensionality of the logs and allows messages describing the same underlying system behavior to be mapped to the same underlying structure. We can see an example of log parsing in Figure 1.1. In this figure, a collection of raw log messages is processed to extract invariant textual structure and abstract variable fields into a structured log template.

The need for log parsing is primarily driven by the intrinsic characteristics of software logs. Unlike tabular data or numerical metrics, software logs are primarily generated for human interpretation and follow non-structured and system-dependent formats. As a result, software logs have high syntactic variation, fragmented vocabulary, and varying formats that may change across software versions, deployments, or even system components. In the absence of a proper log parsing technique, the subsequent steps of log analysis are often overwhelmed by noise, making it difficult to aggregate logs, identify patterns, or compare system executions.

In our work, we adopt language models to improve existing log parsing methods, offering a new approach to extracting meaningful information while maintaining performance and efficiency.

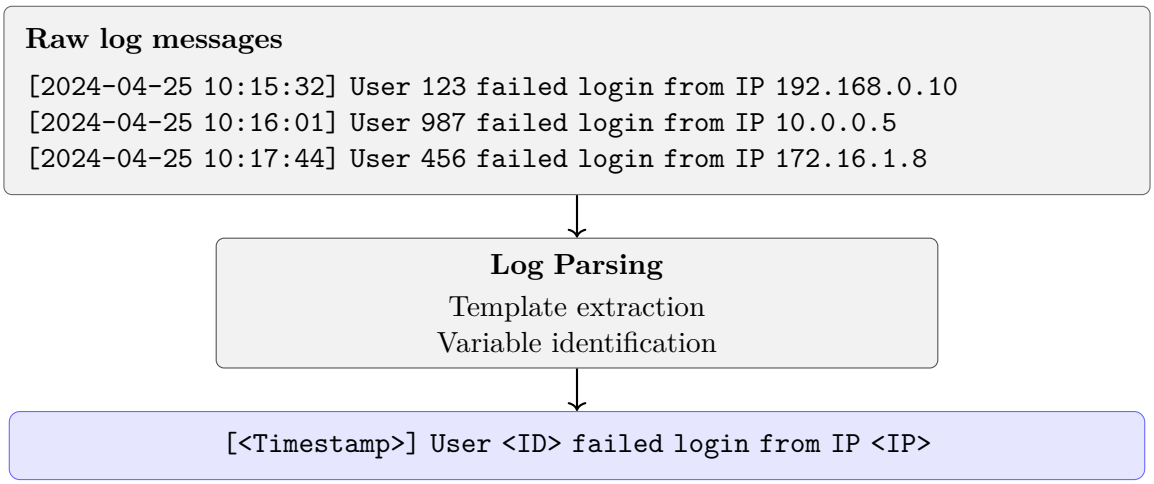

Figure 1.1 Illustration of the log parsing process.

1.1.2 Log Topic Modeling

The purpose of topic modeling is to unveil the underlying semantic structures present in large volumes of text. For the domain of software logs, the goal is to identify groups of log messages that have semantic relatedness and represent underlying system behaviors, operational concerns, or failure modes. Rather than analyzing individual log messages, topic modeling groups log messages into meaningful topics, thereby creating a higher level abstraction over the low level log information. This abstraction is useful in understanding complex system activity by unveiling underlying system behaviors embedded in large volumes of log information.

Applying topic modeling on software logs is subject to a range of unique challenges. First and foremost, unlike traditional text documents containing natural language, log messages are short, semi-structured, and contain a plethora of technical words, identifiers, and domain-specific terminology. Consequently, effective topic modeling on log information requires representations that are sensitive to contextual and semantic similarity beyond the traditional paradigms of natural languages. An illustration of the log topic modeling process is shown in Figure 1.2. In this example, parsed log messages are embedded and clustered into semantically coherent topics representing underlying system behaviors.

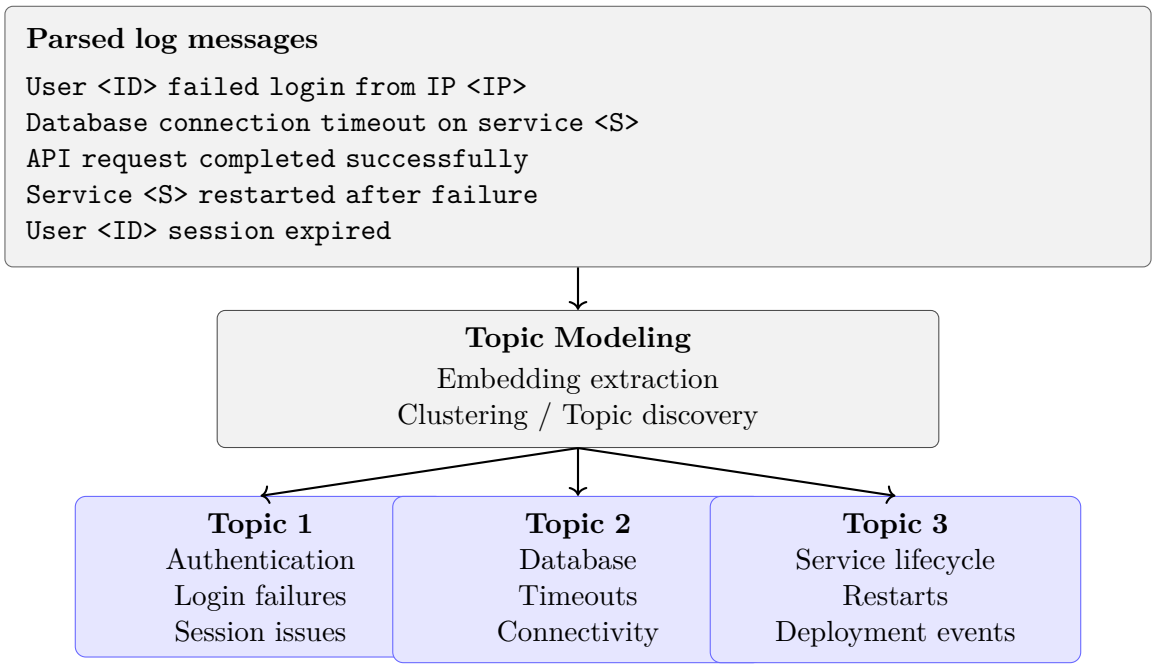

Figure 1.2 Illustration of the log topic modeling process.

By using the strength of contextual embeddings from transformer-based language models, log messages can be embedded in a continuous semantic space in which log messages with similar operational behaviors are proximally located. Topic modeling can be applied on the log information using clustering and embedding-based methods that leverage the strength of contextual embeddings. In the context of log information analysis, topic modeling is not only a tool for summarization but also an intermediate layer that facilitates the analysis of log information.

In this research, we adopt language models to develop a novel approach to log topic modeling, ensuring that the identified topics remain coherent with the non-natural language characteristics of logs, while maintaining performance and interpretability.

1.1.3 Log Causality

Understanding causal dependencies between log lines has become essential for effective and efficient system troubleshooting and diagnosis. Being able to capture connections between log lines in real time may work as a very effective asset for bug prevention and system monitoring, since tracing the root cause of issues can help mitigating similar problems that may arise in the future.

The field of causal discovery has continued to evolve, making it a growing field of research. Causal discovery has been thoroughly investigated in the literature. Traditional methods, such as the PC algorithm [4] and Granger causality [5], are commonly used in causal discovery from observational data. The application of these methods in software logs, however, is restricted. Concurrently, recent breakthroughs in large language models (LLMs) have provided a new research direction in analyzing complex, unstructured, and semi-structured data. Despite the proven effectiveness of LLMs in natural language processing, there has been limited research in applying LLMs in causal discovery in software logs.

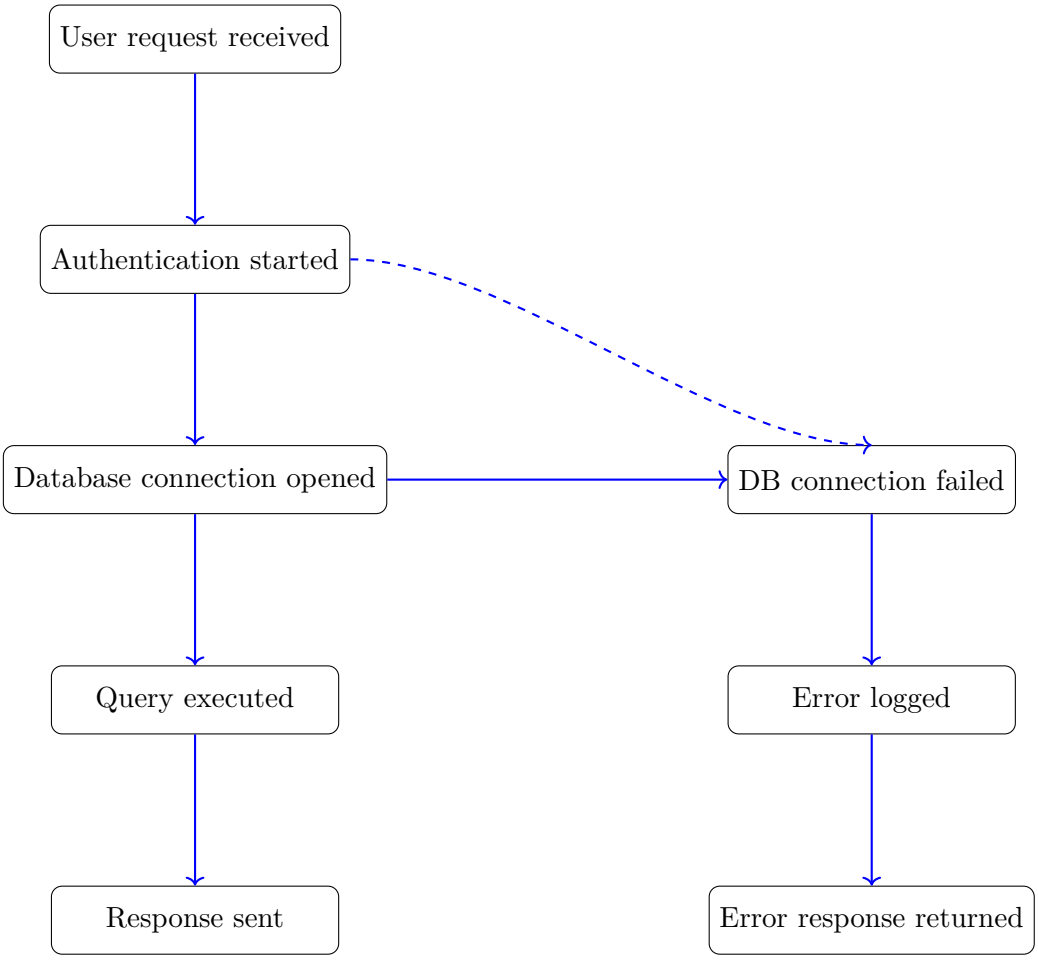

Figure 1.3 Example of causal relationships between log lines. Blue arrows represent inferred causal dependencies.

Figure 1.3 shows an example of a simplified view of the causal relationship between log entries in a software system. The nodes represent the log events, and the blue arrows represent the causal relationships between them. The main workflow (workflow in the left column) represents the successful execution flow, starting from the receipt of the user request and

ending with the sending of the response. In the right column we have the failure case, ending with the sending of the response with the error code. This tracing represents a directed acyclic graph (DAG), where different paths represent different causal sequences in the system execution.

Existing causal models have a number of limitations when they are applied to log data. Firstly, as with the other fields of this research, software logs do not have any natural language properties such as grammatical structures like subject-verb-object structures or any form of narrative. Instead, they have a strict but highly variable form depending on the software that produced them.

Secondly, understanding causal relationships between different lines in a log can also depend on the background of a person interpretation. For instance, a causal relationship identified by a business analyst might not mean the same to a software engineer. Therefore, in order to incorporate human expertise in the process of causal discovery, it is very important the introduction of human-in-the-loop approaches [6].

In this research, we adopt language models to develop a novel method for identifying causal relationships in software log files, providing a new approach to extracting meaningful information while maintaining performance and efficiency.

1.2 Research Objetives

The main goal of this thesis is to explore the possibility of utilizing contemporary language models for the effective knowledge mining in log lines. In order to do so, our research focuses on the adoption of large language models to perform an effective analysis of software logs and for supporting more advanced reasoning processes that go beyond traditional natural language methods. In particular, the doctoral work presented in this thesis seeks to investigate the possibility of progressively enabling more advanced levels of understanding over software logs, including structural normalization, semantic abstraction, and causal reasoning.

The research work carried out in this thesis is divided into a cohesive set of three individual studies, each of which addresses a critical challenge of software log analysis. The collective body of work carried out in this thesis is a unified research work that seeks to bridge the gap between raw, highly heterogeneous software logs and more advanced levels of understanding that may potentially support debugging, root cause analysis, and system understanding within complex software systems.

1. **Language-model-based log parsing for robust log abstraction.** The first contribution of this thesis seeks to address a critical challenge in software log analysis,

namely, log parsing. In particular, the process of log parsing seeks to transform raw, unstructured software logs into more structured representations, referred to as software log templates, that abstract variable parameters while preserving semantic meanings. However, traditional approaches to software log parsing are typically based on manually crafted rules, clustering, and fixed regular expressions, which are generally unable to generalize across a wide range of software systems and logging practices. In particular, such challenges become more apparent within industrial settings, where software logs are typically very heterogeneous, dynamic, and often deviate from traditional, anticipated syntax.

In order to deal with these problems, the first part of this thesis introduces a new log parsing methodology based on language models and shows that it is possible to learn the implicit log format using the implicit knowledge of the log messages encoded by the language models. This methodology has the merit of being more robust to format changes and of not relying on domain knowledge. This contribution is of crucial importance to the thesis because a good and coherent log parsing methodology is a prerequisite to the following steps of the work.

2. **Semantic topic modeling of software logs using language models.** The second contribution of the thesis extends the capabilities of the language models dealing with log parsing and tries to validate the hypothesis that it is also possible to use these models to deal with the topic modeling of log messages. Indeed, finding the log topics using the traditional topic modeling techniques is difficult because of the textual characteristics of the log messages. As mentioned before, these messages are often sparse and non-natural.

   The second part of the thesis explores the application of language models to extract and model latent topics from sequences of log messages, thus allowing the grouping of log lines according to underlying system behaviors and operational concerns. By using contextual embeddings learned from the log data, the proposed method is able to capture semantic similarities that are not immediately apparent on the lexical level. In addition, the proposed contribution expands the applicability of language model-based log analysis beyond the realm of structural normalization, thus showing the effectiveness of these models in the extraction of meaningful semantic patterns from log data. In the context of this thesis, the proposed contribution is seen as an intermediate step between low-level representation learning and causal reasoning, thus showing the effectiveness of language models in the organization of log data into meaningful semantic structures that aid the process of analysis.

3. **Causal discovery in software logs using language-model-driven probability estimation.** The third part of the thesis revolves around the problem of causal discovery in software logs, a problem that is considered to be vital in the context of understanding the failure propagation, behavior, and root cause analysis in software logs. Causal discovery, in general, can be achieved using traditional methods such as the PC algorithm [4], which is based on the concept of constraint-based causal discovery, and Granger causality [5], which is based on time series analysis. These traditional methods, however, are not suitable for the specific properties that are associated with software logs, such as the sequential, sparse, high-dimensional, and context-specific tokens.

   To address the problem, the third part of the thesis presents an approach that uses a causal language model to estimate the probability between the software logs, reducing the computational complexity associated with traditional methods while improving the accuracy in the context of causal discovery.

1.3 Thesis Outline

The remaining of this document is organized as follows. Chapter 2 presents the literature review of the main topics used in this research, including log parsing, log mining, tokenization methods, machine learning, transformers, large language models, and causal structure learning. Chapter 3 outlines the overview of the research. Chapter 4 describes our approach that uses Transformers and Large Language Models to parse software logs. Chapter 5 introduces our approach to Topic Modeling using Large Language Models. Chapter 6 investigates our approach to causal modeling with log lines, and proposes a new method to identify causal links. Finally, Chapter 7 summarizes the thesis contributions, limitations, and presents future research possibilities.

1.4 Publications

The chapters 4, 5, and 6 of this thesis include, respectively, the following published articles:

- 1. V. Bertalan and D. Aloise, **"Using Transformer Models and Textual Analysis for Log Parsing"** 2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE), Florence, Italy, 2023, pp. 367-378, doi: 10.1109/ISSRE59848.2023.00037.

- 2. V. Bertalan and D. Aloise, **"Clustering Textual Features for Log Summarization in Large Software Systems"** Submitted to Software: Practice and Experience.
- 3. V. Bertalan, F. F. Daneshgar and D. Aloise, **"Extracting Causal Relations from Log Sequences Using Causal Language Models"** 2026 33rd IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), Limassol, Cyprus, 2026. Accepted.

CHAPTER 2 LITERATURE REVIEW

In this chapter, a literature review of concepts and techniques related to the research is presented. This review aims to establish the theoretical and methodological foundations necessary to support the proposed research objectives. In Section 2.1, we discuss why logs exist and why they are collected. In Section 2.2, we discuss log parsing and the most cited techniques of log parsing. In Section 2.3, we describe what can be done with the result of log parsing, with the goal of extracting knowledge.

Building upon these foundations, the subsequent sections broaden the discussion toward more advanced analytical and learning techniques. In Section 2.4, we discuss the concept of Topic Modeling and BERTopic, a topic modeling approach. In Section 2.5, we introduce the concept of Attention and Transformer Networks. In Section 2.6, we present Large Language Models (LLM) and their main concepts.

Finally, in Section 2.7, we describe Causal Structure Learning (CSL) and its mathematical foundations. In Section 2.8, we discuss why CSL is important to our project and the current CSL techniques presented in the literature.

2.1 Creating and Using Logs

Due to the high complexity of modern computer systems, the ability to quickly and effectively solve unexpected problems has become a fundamental requirement. In this context, system logs constitute an essential data source used to record information about operating systems and services, including detailed runtime behavior. Therefore, online log analysis and automatic detection of abnormal events are valuable solutions for software development and system maintenance, as they enable the tracking of issues related to software deployment and system execution.

Log analysis technologies are widely adopted in the management of computer system services. According to [7], it is common practice to record detailed runtime information in logs, allowing developers and support engineers to understand system behavior and identify potential issues. Logs are also commonly used to track system behavior [8]. They represent interactions among data, files, services, and applications, and are typically used by developers, DevOps teams, and AI-based methods to understand system behavior in order to identify, localize, and resolve problems.

Despite the significant value contained in logs, analyzing them effectively remains a major

challenge. Modern software systems generate massive volumes of logs, often amounting to gigabytes of data per hour in commercial cloud applications [7].

Moreover, log messages are inherently unstructured, as developers favor flexibility when recording system events. This lack of standardization further complicates automated log analysis. As noted by [7], although logs are indispensable in software development and maintenance, there is no standardized format, and log contents may vary significantly across systems.

In addition to operational monitoring, logs also play a critical role in system security. Online computer systems are exposed to various malicious threats [9], and the ability to analyze logs in near real time is a fundamental step toward system protection. System logs capture detailed computational events and are therefore essential for modern log administration.

Error detection in log files is also a critical task in modern software development and DevOps workflows. According to [1], developers may deploy packages 7 to 10 times per day, and a substantial portion of these deployments result in build or compilation errors. At Google, for example, 37.4% of C++ builds and 29.7% of Java builds result in compilation errors. These findings suggest that the prevalence of build errors in small and medium-sized enterprises may be even higher, given that many quality assurance practices employed by large companies are not always feasible in smaller organizations [10].

Therefore, technological frameworks capable of tracking errors through logs, and preventing their recurrence, represent valuable assets for modern organizations. In [1], we can read that some Google build logs were reported to require more than 12 hours to resolve. Moreover, software developers allocate between 13% and 25% of their working time solely to reading compilation errors, excluding the time required to correct them [11]. These findings highlight the substantial productivity costs associated with manual log inspection.

From this perspective, automation emerges as a natural response to the increasing scale and complexity of log data. Activities such as software engineering, security infrastructure monitoring, and server management, where log collection plays a central role, can benefit significantly from automated log analysis.

Meanwhile, the introduction of machine learning outside academia has transformed the way professionals perform routine activities. In [12], the authors indicate that machine learning algorithms are sufficiently adaptable and flexible to be applied in various domains, including cybersecurity, smart cities, healthcare, e-commerce, and agriculture.

As machine learning algorithms can achieve high accuracy in diverse tasks, they have been increasingly adopted to automate repetitive activities, particularly in scenarios where data

volumes exceed human analytical capacity. Advances in computational resources, such as Graphical Processing Units (GPUs) and Tensor Processing Units (TPUs), further accelerate this trend. As a result, it is foreseeable that many routine activities will be increasingly supported or replaced by machine learning methods, allowing human operators to focus on tasks that provide higher productivity and strategic value.

After considering these points, an alternative approach is to adopt machine learning techniques to automate log analysis, that is, transforming raw log files into structured and interpretable representations for software developers and DevOps professionals by the use of machine learning.

Research in this field has intensified in recent years, with growing efforts to expand the range of feasible analytical tasks. Two of the most prominent research directions are Log Parsing [8, 13, 14, 15, 16], discussed in Section 2.2, and Log Mining [17, 18, 19, 20, 21], discussed in Section 2.3.

2.2 Log Parsing

As mentioned in Chapter 1, log entries are text produced by events (e.g., `print()` , `logger .log()`) embedded in the source code of software systems. Before mining and analyzing log files, it is usually necessary to structure them due to their unstructured nature. However, log records are generally written as non-natural language text by software developers, which makes automatic processing difficult.

Log parsing consists of assigning templates to incoming log messages so that they can be transformed into structured representations. According to [7], log parsing aims to convert each raw log message into a specific event template (e.g., *"Received block <*> of size <*> from /<*>"*) associated with key parameters. This process generally requires identifying all log statements and generating appropriate expressions to distinguish constant components from variable ones. Usually, regular expressions are often employed as an initial solution. However, they require significant manual effort and domain knowledge.

We illustrate this transformation in Figure 2.1. In the first block, a code snippet shows the logging instructions used to generate log messages. The second block presents an example of a raw log message produced by this code. The third block shows the structured log obtained after parsing, where each component of the message is assigned a specific type.

Typically, log lines are composed of constant string templates and variable values. The string templates correspond to logging instructions (e.g., `print()`, `log.info()`), which record specific system events. After parsing, each log message is represented as a template along with

a list of parameters, which can be leveraged in multiple downstream tasks, such as anomaly detection, system monitoring, and fault diagnosis.

Many data mining models used for log analysis require structured inputs, such as event sequences or matrices. However, raw log messages are typically unstructured. Consequently, due to this unstructured nature, log parsing is commonly the first step in automated log analysis pipelines, enabling the transformation of free-text messages into structured data [7]. This structured representation enables systematic analysis and comparison of log events across large datasets.

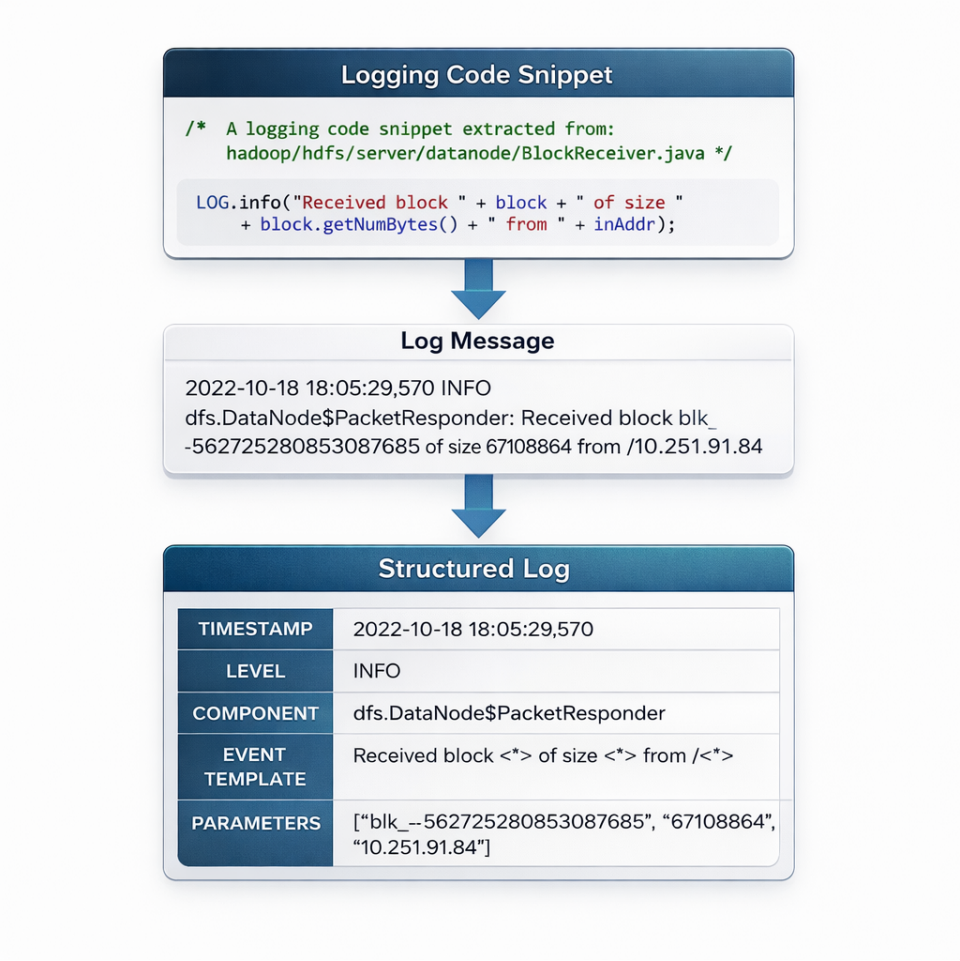

Figure 2.1 Parsing of a raw log to a structured log

This task is nontrivial, as log formats can vary significantly across systems. According to [7], logs may differ in timestamp formats, verbosity levels (e.g., `ERROR`, `INFO`, `DEBUG`), and component structures, and there is no universal standard for log generation.

To address this challenge, Natural Language Processing (NLP) techniques are frequently ap-

plied during the preprocessing stage. As illustrated in Figure 2.2, two common preprocessing techniques are tokenization and type detection. Tokenization divides text into sequences of tokens, such as words, symbols, or punctuation marks [22]. Type detection identifies predefined entities, such as dates, times, IP addresses, and numerical values, and replaces their concrete values with abstract type labels [23]. This abstraction reduces vocabulary size and improves template generalization.

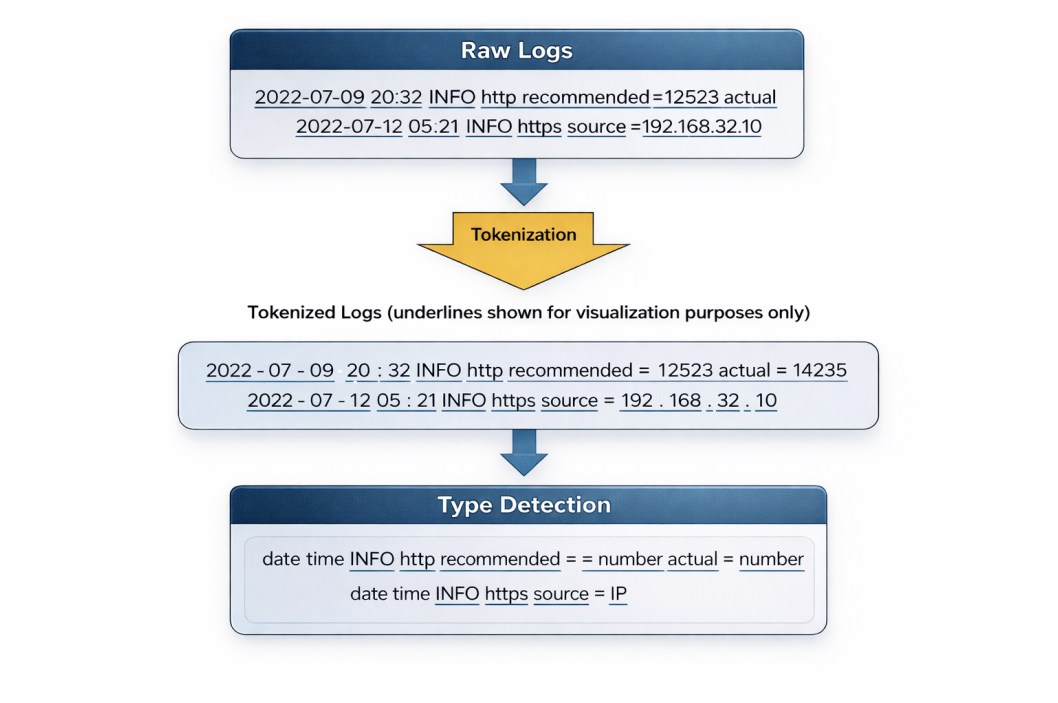

Figure 2.2 Log text preprocessing techniques

However, in large-scale systems composed of heterogeneous software and hardware components, maintaining preprocessing pipelines becomes increasingly complex. Frequent software updates often require continuous adjustments to tokenization rules and type detection heuristics, making the process error-prone and difficult to scale.

Most existing log parsing methods are either tailored to specific systems or perform well only when log datasets exhibit limited variability in their templates. As a result, these methods often lack the generalization capability required to handle diverse log writing styles across multiple systems.

Furthermore, operators of large infrastructures face additional overhead when managing multiple parsing approaches for different components. Given the massive volume of logs generated by complex systems, human monitoring of individual events becomes impractical. This limitation often leads to reliance on coarse-grained severity labels such as *information*, *warning*,

and *error* [16]. Such labels provide only limited contextual information and depend heavily on prior assumptions made during system design.

Analyzing the performance of existing log parsing techniques across diverse systems reveals inconsistencies in reliability and accuracy [24]. As a result, domain-specific knowledge is often required to select or adapt parsing methods for a given application. This dependency highlights the need for more robust and generalizable parsing approaches.

2.2.1 Log Parsing Techniques

Based on the challenges discussed above, log parsing methods should ideally achieve high accuracy across logs from heterogeneous systems, ranging from single applications to large-scale cloud infrastructures, while requiring minimal human intervention. Such methods would offer flexibility for developers and expressiveness for users, enabling more effective system monitoring and diagnosis.

Several data-driven log parsing approaches have been proposed based on semi-supervised or unsupervised learning techniques. Following the taxonomy proposed by [24], these approaches can be categorized into *longest common subsequence* methods, *structured parse trees* methods, *directed acyclic graph* methods (which can be grouped as Online Streaming Methods), *clone detection* methods, and *evolutionary search* methods (grouped as Offline Search Methods).

Online Streaming (Graph Search Methods)

In these methods, the log event detection is based on high-level structures to capture data at execution time. Common architectures that capture data at runtime are Longest Common Subsequences, Structured Parse Trees, and Directed Acyclic Graphs (DAGs).

- **Longest Common Subsequences**: In this approach, the algorithm compares the longest common subsequence between two sets of strings. A longest common subsequence (LCS) γ is called a common subsequence of α and β if it is a subsequence of each. The algorithm most cited that uses this approach is Spell [14]. One of the main advantages of the method, according to the authors, is that it is possible to still derive a particular message template even with very few instances of log entries produced by its log statement.

  Spell's behavior works like the following: when a new log entry arrives, it is first parsed into a token sequence, using a predefined set of delimiters. After that, the algorithm compares the token sequence with the longest common subsequence from all previous

objects in the a data map, to see if the sequence is similar to one of the existing subsequences, or if the algorithm needs to create a new object and insert it into the map.

- **Structured Parse Trees**: In this approach, developed by [25], the algorithms use a data structure, called parse trees, to map of variable-length fields to fixed-length fields in a lossless source code, associating a string of binary source symbols with each of a set of K code symbols. The resulting parse tree is a complete binary tree with K leaf nodes and $K - 1$ internal nodes that include the root at the top of the tree.

  The algorithm most cited that uses this method is SHISO, developed by [26]. In their work, the authors use a structured tree using the nodes generated from log messages, creating a method for mining and refining the log format continuously in real-time. Their method works by structuring a tree of log messages based on log format similarity and then archiving continuous clustering log messages and identifying a log format without previous knowledge after the input.

- **Directed Acyclic Graphs**: A directed acyclic graph (DAG) is a data structure composed of variables (nodes) and arrows between nodes (directed edges) such that it is not possible to start at any node, follow the directed edges in the arrowhead direction, and end up back at the same node [27]. Furthermore, according to [27], a causal DAG is one in which the arrows can be interpreted as causal relationships and in which all common causes of any pair of variables in the graph are also included in the graph. Some techniques, such as the one presented by [28], use graph search for evolving structures, in order to capture system information at runtime.

  The algorithm most cited that uses DAGs is Drain, by [29] . The Drain algorithm abstracts log messages into event types using a fixed depth parse tree to guide the log event analysis process, which avoids constructing a profound and unbalanced tree and encodes specially designed parsing rules in the parse tree nodes. Although the basic algorithm used parse trees, the updated authors are working to use DAGs in a posterior work [30], obtaining better results.

Offline Search

When a runtime operation is not necessary, several methods use batch processing, therefore needing all the log data to be available during discovery in order to develop a knowledge base [24]. Also according to [24], offline search requires batch processing (collecting logs for

a certain time) before abstracting log messages into event types, therefore characterizing an offline discovery phase.

- **Clone Detection**: Clone detection refers to the capture of duplicated log records, merging similar ones. The algorithm most cited for this purpose is AEL, developed by [31]. In this approach, the authors use a log abstraction technique that recognizes the internal structure of each log line. Using the recovered structure, log lines can be easily summarized and categorized to help to comprehend the complex behavior of large software applications. The method first uses heuristics to recognize dynamic tokens in log lines. Then, it tokenizes log lines into different groups according to the number of words and estimated parameters in each log line. Afterwards, the method compares the log records and then reexamines all execution events to identify which ones are to be merged.

- **Evolutionary Search**: Evolutionary algorithms are those for which a single fitness function is optimized, usually adopted to search a specific space of possible feature subsets. An evolutionary search approach destined to log parsing is Molfi, developed by [32]. The authors use a customized version of NSGA-II, an evolutionary search algorithm, to work with log lines. First, the authors use regular expressions to identify trivial variable parts within the log messages based on domain knowledge. Then, the algorithm encodes the lines by using pre-defined templates, each corresponding to a group of pre-processed log messages having the same length and sharing all fixed tokens. Subsequently, the method uses the Initial Population algorithm to generate the chromosomes. It applies the standard steps of evolutionary search (crossovers and mutations), so as to return a subset of Pareto optimal solutions.

2.2.2 Current Trends in Log Parsing

The current landscape of log parsing research emphasizes a shift toward semantic-aware models that eliminate the need for manual rule definition and hyper-parameter tuning. Systems like LogParser-LLM [33] integrate the expansive knowledge of large language models to blend semantic insights with statistical nuances, facilitating rapid adaptability through online parsing. Similarly, LibreLog [34] leverages open-source models like Llama3-8B to enhance privacy while utilizing a fixed-depth grouping tree and similarity-scoring-based retrieval to distinguish between static templates and dynamic variables. To address the high computational costs of these models, frameworks such as LILAC [35] and LibreLog incorporate adaptive parsing caches or template memory to store processed results, significantly reducing the number

of required LLM invocations. Furthermore, recent exploratory studies like LLMParser [36] demonstrate that smaller, fine-tuned models such as Flan-T5-base can achieve accuracy levels comparable to much larger architectures with significantly shorter inference times.

In the domain of log-based anomaly detection, recent methodologies prioritize self-supervised learning and domain generalization to handle the variability of modern system logs. LogFiT [37], for instance, utilizes a BERT-based language model fine-tuned on normal data through masked sentence prediction, allowing it to identify deviations without requiring labeled anomaly data. Other frameworks like LogLLM [38] streamline this process by replacing traditional template extraction with regular expression preprocessing, using a projector to align semantic vectors between BERT and Llama for cohesive sequence classification. To improve performance across heterogeneous environments, LogFormer [39] introduces a pre-training and adapter-based tuning pipeline that establishes shared semantic knowledge across multiple domains.

Research has also increasingly focused on the interpretability and security-specific applications of log analysis, ensuring that automated predictions are actionable for human experts. LogPrompt [40] addresses the limited transparency of traditional methods by employing advanced prompting strategies that generate readable rationales for predictions, a feature highly rated by experienced practitioners for its usefulness in online scenarios.

In security contexts, the LLM4Sec [41] pipeline demonstrates that fine-tuned sequence classification models like DistilRoBERTa can achieve near-perfect F1-scores across diverse web and system log sources. While modern neural approaches dominate semantic interpretation, traditional machine learning models like Random Forests remain effective for real-time ransomware detection by focusing on specific behavioral features such as process creation and file access patterns. Collectively, these works underscore the necessity of moving toward large-scale, representative benchmarks to accurately evaluate the granular performance of these tools in production environments.

The evolution of these tools highlights a clear trend: the integration of LLMs is no longer just about accuracy, but about making log analysis more efficient, private, and understandable for the engineers who rely on them daily.

2.3 Log Mining

In essence, log mining consists of applying text mining techniques in order to extract information from log files. Many traditional data mining techniques have been adapted to automate log analysis, insofar as the interaction between log mining and log parsing has been only

partially examined [16]. Since the logs are automated messages, raw messages often display repetitive patterns that can be effectively extracted to capture their patterns.

Before machine learning, the usual log mining approach chosen has often been based on domain rules and aimed at distinguishing between constant parts and variable parts to categorize log files. However, log files are composed of sentences or partial sentences that are composed of connected tokens and, therefore, can also have their semantic aspects taken into account. Therefore, machine learning can be used as an effective tool in aiding in this process.

In many cases, a strategy is to explore static analysis techniques to extract event templates directly from the source code. Although it is a viable approach in some cases, the source code is not always accessible in practice, according to [7], and tracking all changes is difficult, as cited by [28].

Based on [23], an automated log analyzer is a tool that parses and mines logs in a automated way. The authors describe that an effective analyzer must have one component to identify patterns from log messages and another component to match this pattern with the flow of log messages to identify events and anomalies. In [23], the author argues that such log-message analyzers must have the following desirable properties:

- **No-supervision**: Pattern recognizers must work from scratch without prior knowledge or human supervision. Pattern recognition should not need administrator input for new log message formats;
- **Heterogeneity**:There may be log messages generated by different applications and systems. Each system can generate log messages in several formats. Automated recognizers must find all formats of log messages, regardless of their origin;
- **Efficiency**: Computer systems produce millions of log messages per day. Log processing must be performed so efficiently that the process rate is always faster than the log generation rate;
- **Scalability**: Pattern recognizers must be able to process large batches of log messages to maintain the current set of patterns without causing input/output and memory bottlenecks.

As highlighted by [42], both user navigational behavior and system monitoring events are essential for ensuring system sustainability. Meanwhile, the emergence of new technologies

has created significant opportunities in domains such as distance learning, cooperative work, and information dissemination within large, heterogeneous information systems.

All these technologies inherently generate substantial volumes of log data. For instance, in distance learning environments, software systems track students's behavioral patterns on e-learning platforms, monitoring which functionalities are used more frequently than others [43]. In this context, log mining becomes a valuable analytical approach, enabling stakeholders to extract actionable insights that support student success and improve system effectiveness.

2.3.1 Log Classification

According to [44], log classification is the task of categorizing them into groups of relevant content, given a set of documents. Three of the most popular techniques are *frequent itemset mining*, in which we extract the more frequent relations; *natural language processing* techniques, in which we employ NLP methods to classify the log lines; and *clustering*, in which we use grouping algorithms to separate the log lines based on their characteristics.

- **Frequent Itemset Mining**: In the words of [45], frequent itemset mining is the task of extracting any existing frequent itemset (having an occurrence frequency no less than some threshold) in a dataset. A frequent itemset is, by definition, the number of data records, or percentage of them, that include the subset of items above a predefined threshold.

  Two techniques that use frequent itemset mining for log records are LogHound [46] and Iplom [47]. LogHound is a generalization of the Apriori algorithm for log files, using specific optimization techniques to improve the performance of the algorithm. The authors used a summary vector to reduce the memory cost of mining frequent items, a memory-based cache to load the most commonly used transaction data, and a separate pass over the data set to detect correlations between frequent items.

  On the other hand, Iplom does not use Apriori as a starting point. Instead, it works by iteratively partitioning a set of log messages used as training exemplars. At the partitioning process, the resultant partitions come closer to containing only log messages, which are produced by the same line format. At the end of the partitioning process, the algorithm attempts to discover the line formats that produced the lines in each partition.

- **Natural Language Processing Techniques**: Standard Natural Language Processing techniques are also used to analyze the linguistic structure of log files and try to

obtain knowledge from their characteristics. The two examples mentioned by [24] is the use of Conditional Random Fields by [48] and the use of Neural Language Models by [49].

The work of [48] uses Conditional Random Fields, a supervised learning technique for segmenting or labeling sequential word data, by calculating the conditional probability of each log message with feature functions, the probability of positional relations of words, and their features defined with positional relations of words and labels to be used in making feature functions. The model first requires feature templates and training data. Then, it runs over the train data to label words of system logs to prepare a new train data set with labels referring to fixed or variable tokens. Afterwards, the algorithm classifies every line of target logs, returning the log templates.

In contrast, the Neural Language Model adopted by [49] uses a character-based language model to determine variables and fixed tokens (which the authors called mutable and nonmutable characters) of a log message. Mutable characters are replaced with a wildcard, and non-mutable characters are kept the same. Then, the model attempts to predict what type each character is, namely 1 if it belongs to a mutable part and 0 otherwise, using a neural network architecture composed of an embedding layer, a bidirectional long-short-term memory (bi-LSTM) layer, a dropout layer, a feedforward layer, and softmax layer.

- **Clustering**: Clustering is one of the most popular non-supervised learning techniques. The process is defined by [50] as grouping data instances into subsets in such a manner that similar instances are grouped together, while different instances belong to different groups, by using a pre-defined metric to determine the similarity or dissimilarity between any pair of objects.

  There are several techniques that use clustering methods to gather information from log files. Examples are SLCT [51], that used a density-based clustering algorithm to cluster data by processing log datasets at the word level and clustering it clusters log data with common frequent words; LFA, by [52], a modified version of SLCT in which the clustering algorithm finds clustering within log lines, instead of the whole log file; POP, developed by [53], that use heuristic rules and clustering methods, outputting templates with Longest Common Subsequences, by using parallel computers; LogSig, developed by [54], that use a message signature based algorithm to generate system events from textual log messages, and search the most representative message signatures to categorize log messages into a set of event types; LKE, developed by [55], that uses a heuristic-based approach to group log messages from the same statements and considers

their similar tokens as event types; HLAer, developed by [56], that use log abstraction based on a hierarchical clustering approach and pattern recognition using the Smith-Waterman algorithm; and LogMine, developed by [23], that uses map-reduce techniques to deal with heterogeneous log messages generated in a wide variety of systems.

2.4 Topic Modeling

Topic modeling is a text mining method that can be applied to automatically discover the topics being discussed in a *corpus* of documents. A topic is defined by a list of words that have strong relationships with each other and that also share similar concepts and semantics.

Topic modeling can be applied to improve the understanding and summarization of large document collections, to find new topics in a given area of research or over a certain period of time, and to facilitate other applications such as personalization of content and trending.

Topic modeling was originally created and remains applicable to the fields of information retrieval and text classification. Its use has become prominent in the field of the digital humanities and social sciences through the framework of distant reading, where it has become increasingly popular to apply it to research questions about the distribution of content. Distant reading refers to a type of analysis that differs from close reading, which involves the careful and detailed analysis of a text, and that relies on the application of text mining techniques to find the surface features of a text [57].

In [58], we can see that a topic model is a mathematical construct that takes a set of documents as input and produces a set of topics that accurately describe the contents of the input documents. The fundamental objective of topic modeling is to determine the primary subjects of a given set of documents, which allows the condensation of a document set containing thousands of documents into a concise summary that encapsulates the dominant subjects.

Furthermore, according to [58], the author describes that the precursors of topic models were developed in 1990, termed Latent Semantic Analysis (LSA). LSA is a technique that applies Singular Value Decomposition (SVD) of the term-document matrix to reduce dimensionality.

A further improvement of the technique, proposed by [59], is termed Probabilistic Latent Semantic Indexing (pLSI), where the fundamental idea is to define topics based on the joint probability of words.

The term “Topic Modeling” was coined by [60] based on the introduction of Latent Dirichlet Allocation (LDA). Latent Dirichlet Allocation (LDA), according to the authors, is a probabilistic topic model that represents each document as a mixture of topics and each topic as a distribution over words. It assumes a generative process in which, for each word in

a document, a topic is sampled from a document-specific Dirichlet distribution, and then a word is sampled from the corresponding topic-specific Dirichlet distribution. Model inference estimates the hidden topic distributions that maximize the likelihood of the observed corpus. However, various variations of the technique were proposed to address different problems, including different sources of data, modeling temporal variations of topics, faster computation, and scalability to large-scale databases.

2.4.1 BerTopic

According to [61], BERTopic is a framework that incorporates algorithms for the automatic search of dense topics within a document collection, based on the assumption that semantically similar documents form topics.

The methodology proposed by [61], illustrated in Figure 4.4, follows a flow similar to the pipeline presented in Section 3.2. It is composed of five steps performed sequentially, which are detailed below:

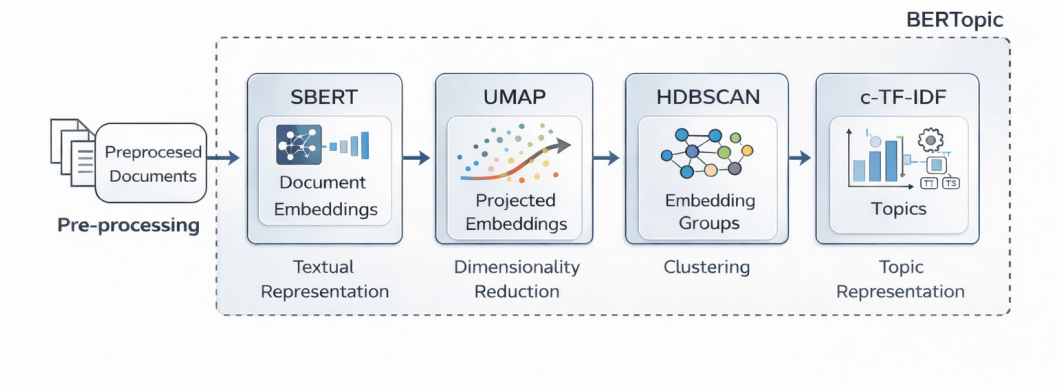

Figure 2.3 BerTopic workflow

- **Preprocessing**: Preprocessing plays a fundamental role in NLP tasks, as it generates a more structured representation of the text and reduces the final set of words (i.e., the vocabulary) that will be used as input for learning models. In fact, applying a specific preprocessing step for unsupervised learning analysis directly impacts the precision of the final models, reducing noise and improving the quality of the identified topics.
- **Textual Representation**: Mapping sentences or documents into numerical vector spaces is an efficient method for generating richer text representations in NLP tasks. These representations preserve semantic, syntactic, and even contextual information within the documents, leading to better performance in learning models that rely on

the vector representation of text as input. BERTopic was proposed to be used in conjunction with Sentence-BERT (SBERT) models [62], which are an adaptation of the BERT model for sentence embeddings. SBERT models use an additional layer to aggregate word representations into a single representation. One of SBERT's main strengths is its scalability for processing large volumes of documents, in addition to its adaptability, as it leverages knowledge previously acquired by BERT models.

- **Dimensionality Reduction**: The representations generated by SBERT models have high dimensionality, which hinders the semantic clustering of documents due to the so-called *curse of dimensionality*. To handle this limitation, BERTopic adds a dimensionality reduction step that applies the Uniform Manifold Approximation and Projection for Dimension Reduction (UMAP) algorithm [63] to reduce the size of the representations used in the subsequent clustering stage. UMAP stands out for preserving more local and global characteristics of high-dimensional data when projected into lower dimensions. Using it allows for a more condensed representation of information using fewer dimensions, reducing noise caused by highly correlated variables.

- **Clustering**: For the semantic clustering of the documents's projected vectors, BERTopic employs the HDBSCAN algorithm [64]. HDBSCAN is known for its superior performance compared to other density-based clustering algorithms in terms of both precision and efficiency. HDBSCAN can handle clusters of different shapes and sizes, making it ideal for identifying topics within a document set. The method can identify instances that do not belong to any group (i.e., outliers), reducing noise in the final topic representation. As a result, the detected topics are more coherent and representative of the underlying data structure.

- **Topic Representation**: As the final method for topic representation, BERTopic uses its own technique called *c-TF-IDF*. In this method, each group identified in the clustering stage is treated as a single corpus to which *TF-IDF*is applied. This technique works similarly to the original *TF-IDF*, which compares the importance of words by analyzing the entire corpus. However, *c-TF-IDF* utilizes the clusters generated in the clustering stage by applying *TF-IDF* to each of them. This process ranks words according to their importance for each group generated during the clustering process and extracts the main topics for each group. The calculation is performed using the formula:

$$c - \text{TF-IDF} = \frac{t_i}{w_i} \cdot \log \frac{m}{\sum_j^n t_j} \tag{2.1}$$

Where the frequency of each word t is extracted for class i and divided by the total

number of words in class i. This step can be seen as a form of regularization for frequent words within the class. Then, this value is multiplied by the logarithm of the total number of documents (m) divided by the total frequency of word t across all n classes. Thus, an importance value is obtained for each word within a cluster, which will be used to create the topics.

This procedure allows for the calculation of the most important terms in each cluster, then generating the topic distribution. The technique is based on the assumption that by extracting the most relevant words from each cluster, one obtains representative descriptions of the topics.

2.5 Attention and Transformer Networks

2.5.1 Attention and Self-Attention

Attention mechanisms, according to [65], are a family of computational methods that are inspired by the human concept of attention to provide algorithms with the ability to learn to focus their resources and assign different degrees of importance to different aspects of its inputs, which usually leads to an improvement in the results obtained in several applications.

Furthermore, neural attention is a correlation detection mechanism. In the context of natural language processing applications, this mechanism allows the detection of correlations between text components, such as words, in order to map the surface structure of the language and its context and ordering relationships.

Attention is capable of receiving a complete text in the form of a sequence of vector representations $V = v_0, ..., v_n : V \in \mathbb{R}^{n \cdot d}$, one for each word that makes up the text. In this way, the algorithm generates a vector representation c for the entire text, of the same dimension d as the vectors v_i, which captures the entire context of the full text, using a large amount of representations c to train a neural network to perform label classification.

A trivial way to generate the c representation of the full text would be to add the v_i representations of the words that compose it, but there is a specific class of words that can carry more importance than others when it comes to establishing the classification of labels: the words to which a human being would pay more attention during their reading, thus having a higher coefficient of attention.

These words also tend to be closer morphologically, syntactically, and semantically to each other than to common words in a given language. If we know that this property is well represented in the space of the input vectors v_i that we receive, the algorithm can generate a

representation vector u, called the pattern vector, that captures the common pattern of these words with the highest attention coefficient. The mathematical description of this procedure is shown in Equations 2.2, 2.3, and 2.4.

$$e_i = \varphi(u, v_i) \tag{2.2}$$

$$\alpha(i) = Softmax(e_i) = \frac{e_i}{\sum_i e_i} \tag{2.3}$$

$$c = \sum_i \alpha_i v_i \tag{2.4}$$

First, the algorithm calculates a similarity coefficient e_i between each representation vector v_i of the input sequence and the pattern vector u, using a similarity function $\varphi : \mathbb{R}^d \times \mathbb{R}^d \longrightarrow \mathbb{R}$. Then normalize the similarity coefficients e_i, using the Softmax normalization function and thus generating the attention coefficients α_i. The algorithm then generates the vector c, making a sum of the input vectors v_i, balanced by the attention coefficients α_i, so that the vector c is nothing more than a combined measure of linear correlation between the standard vector and the representation vectors.

Therefore, the full text representation will be much more influenced by representations of words that have a high correlation with the u pattern vector, that is, words that are or resemble the most important words.

A generalization of this attention mechanism is the increase in the dimensionality of the vector α, replacing it with an attention matrix, in the two-dimensional case, or by an attention tensor, in the cases of dimension greater than or equal to 3. This increase in dimensionality is achieved by projecting the v_i representations into different vector spaces. With this multitude of projections of the same representation, it is possible for the engine to focus its attention on multiple aspects of the relationship between each vector v_i and the pattern u.

In the case of natural language processing, as mentioned before, the attention mechanism with multiple pattern vectors is used to allow the algorithm to pay attention to the different words of the input sentence, when generating each of the words of the output sentence. However, it is possible to use the same mechanism to make an algorithm pay attention to different parts of the same sentence when processing it for some purpose. This variant of this attention mechanism is usually called *self-attention*, or internal attention, because in this case the u_j pattern vectors represent other parts of the same sentence that we are processing. In this context, the representations generated are still a measure of correlation, but between

the words that make up the sentence itself.

The main example of the use of self-attention, both in general and with regard to this work, is the generation of so-called dense representations, or embeddings. In this case, an embedding generation algorithm receives the representations v_i, corresponding to embedding prototypes, generated by a more rudimentary algorithm, one for each word that composes the input sentence. At each iteration j of the algorithm, all embedding prototypes are then combined through a sum, balanced by the attention coefficients α_{ji} between the embedding prototype u_j of the iteration and all other embedding prototypes v_i of the sentence, including itself.

The context vector c_j, generated by this combination, is used as the new embedding of the j-th word of the sentence and contains, not only information about that word, but also combined information of all other words of the sentence, weighted some degree of similarity between them. Used in this way, the attention mechanism allows for the encapsulation of relationships of context internal to the sentence itself when generating the dense representations that will be used to process it. There is no limit to the distance between such relationships and this even allows the representation of long-range context relationships, which was a significant challenge for previous linguistic processing algorithms.

An alternative type of attention is memory-based attention, cited by [30]. In a simplified way, this method tries to address the problem of finding, for a query q, a value v, searching among the keys K of the values, the one that is most appropriate for the question q. In the non-deterministic version of the memory-based search, the answer must not be a single v value, but a combination of v values that gives greater weight to those that are most appropriate for answering q.

A possible way to represent memory-based attention is to address the memory by calculating the similarity between the question q and each key k_i, then these similarities are normalized and an answer c is generated by summing all the response values v_i, each weighted by the normalized similarity coefficient between its key k_i and the question q. The set of equations is identical to the one shown above, except that similarity, in the case of memory-based attention, is calculated with respect to a set of values k_i different from the set v_i that is combined to generate the answer c.

2.5.2 Transformers

The Transformer [65] architecture, developed by Google Brain researchers and introduced in 2017, is an encoder-decoder architecture based on attention mechanisms. First, we must

understand the operation of an *encoder* − *decoder*. The encoder's function is to process the input data and collect the pertinent information, generating a sequence of continuous representations that will be fed to the decoder, which will do the opposite. In this way, the decoder will incorporate the information generated in the coding layers, generating an output sequence.

A limitation of a simple encoder-decoder, illustrated in Figure 2.4, is the fact that, for long data sequences, when compressing an input to a single vector of representations at the output of the encoder, the information referring to the beginning of the sequence tends to disappear and the context of the encoded data is biased towards the end of the sequence generated by the encoder. One of the solutions to this problem is the use of attention mechanisms.

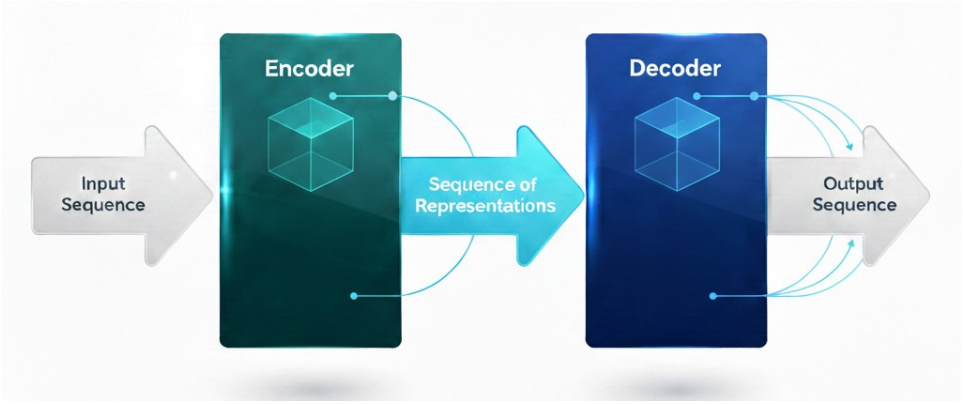

Figure 2.4 Simplified Model of an Encoder-Decoder Architecture

As mentioned in Section 2.5, attention is a technique that aims to reproduce human cognitive attention. Thus, the objective of attention in the context of machine learning is to make the model focus on the most interesting region of a given input for each step and, consequently, to spend a greater computational cost on this part of the data. The Transformer architecture employs three-step attention mechanisms in three different ways. The first is the encoder-decoder attention layers. Here, the vectors k and v are taken from the output of the encoder, while q originates from the last layer of the decoder. In this way, each position in the decoder is passed to all positions in the input sequence.

The second way is the mechanism of self-attention employed in the encoders. In this type of layer, in order that each encoder position can access the positions of the previous layer, the vectors q, k and v are obtained from the output of the previous layer. The third and final way is equivalent to the previous one. Only this time, the mechanism attention span is applied to the decoder. Additionally, the mask explained above is used here, which prevents the flow of information in the opposite direction.

One of the Transformer's differentials lies in the way the input and output vectors are provided to the encoder and decoder, respectively. First, this architecture uses learned embeddings to convert input and output elements into d_{model} dimension vectors.

Given the fact that this architecture is used for applications in sequence data and does not contain convolutions or recurrence, the Transformer makes use of the so-called *positional encoding*, on the basis of the encoder and decoder. This encoding is responsible for introducing the position information of each element in the sequence. This applies both to the input and to the output. It is worth noting that, like embeddings, the positional encoding has dimension d_{model}, so both can be added. In the work that elicited this architecture [65], the positional coding is given by Equations 2.5 and 2.6.

$$PE(pos, 2i) = sin(\frac{pos}{10000^{2i/d_{model}}}) \tag{2.5}$$

$$PE(pos, 2i+1) = cos(\frac{pos}{10000^{2i/d_{model}}}) \tag{2.6}$$

Here, *pos* is the position and i is the dimension. The choice of this function lies in the hypothesis that once given a value k, the position PE_{pos+k} can be represented as a linear function of PE_{pos}. The transformer architecture, which contains all the elements mentioned above, is presented in Figure 2.5.

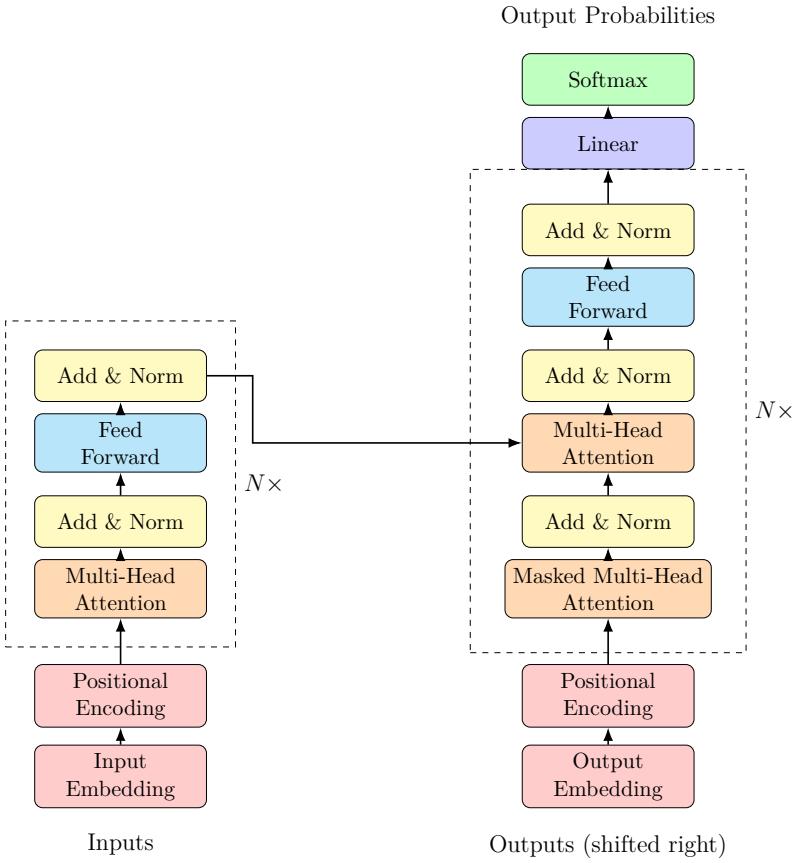

Figure 2.5 Architecture of a transformer.

Originally designed to handle sequential data, the Transformer has been widely used in natural language processing tasks. One of its objectives is to solve several problems associated with recurrent neural networks (RNNs), such as the lack of parallelization, since in RNNs data must be processed in order. Taking the classic example of translating a sentence, RNNs take sequential steps during encoding and decoding; that is, the processing of the current word depends on the previous one. Specifically, in RNNs, a sequence of hidden states h_t is generated as a function of the previous hidden state h_{t-1} and the input at position t. In Transformer networks, based exclusively on attention mechanisms, training time is drastically reduced thanks to the computational parallelization made possible by the use of Multi-Head Attention.

It is also important to note two details: first, in the case of the Transformer, the attention function encapsulates normalization, weighting the value vectors by attention coefficients, and applying a similarity function, which here corresponds to the Softmax argument just before the output probabilities. Second, it is important to note that here k, q, and v encapsulate the

entire sentence they represent and not just one of its artifacts, and therefore these structures manifest themselves in the Transformer context as matrices and not vectors.

Finally, the algorithm uses structures called *masks*, which are a complementary structure added to the Decoder's attention functions that do not allow it to take into account, at any moment, the representations of words after that moment. They serve to equate the training process with the execution process, in which the encodings of the latter words are not known precisely because they will be generated by the Decoder itself in the future; at the same time, the mask allows training to be done much faster, without having to resort to each prediction generated previously, as in the case of RNNs. It is precisely this property that generates such a huge time gain on the part of the Transformer in relation to other state-of-the-art algorithms. In practice, the mask consists of replacing the scores corresponding to illegal connections by the value $-\infty$ in the Decoder instances, to bring their contributions to zero in Softmax.

The conversion of word sentences, both the source sentence and the target sentence, to their vector representations in embeddings is performed by segmenting the sentences into subwords, using algorithms such as Byte Pair Encoding (BPE) or Wordpiece. Depending on the application, the function is applied using an embeddings generation function defined as a function of a matrix B, shared with the final linear transformation of the decoding component. Furthermore, the Transformer adds, to each generated vector representation, a position embedding, that is, a vector that characterizes the position of that word in the sentence. In this manner, the algorithm can differentiate the same word when it appears in different positions, which recurring models would naturally do, but that is not, a priori, within reach of the Transformer architecture. As mentioned, the position embeddings used are as defined in Equations 2.5 and 2.6.

In the light of these details, it is possible to see that what Transformer does is find new representations for the sentences, by iteratively adding, through the leak connections, their previous representations with measures of autocorrelation of these same representations; the algorithm learns, through its multi-attention functions, which parts of these measures should be intensified and which should be attenuated, thus generating representations, for each word of the sentence, that encapsulate its context relations with the other words of the same sentence. verdict.

The result of this process is illustrated in Figure 2.6, extracted from [66]. In this image, it is possible to see a color map representing the attention score, that is, the measure of autocorrelation, given by the transformer to each word of a sentence to measure its relationship with the word *it*; It is possible to notice that the values assigned to *The* and *animal*, which

represent the referent of the word *it*, are prominent, indicating that this relationship was learned by the algorithm through its autocorrelation representation.

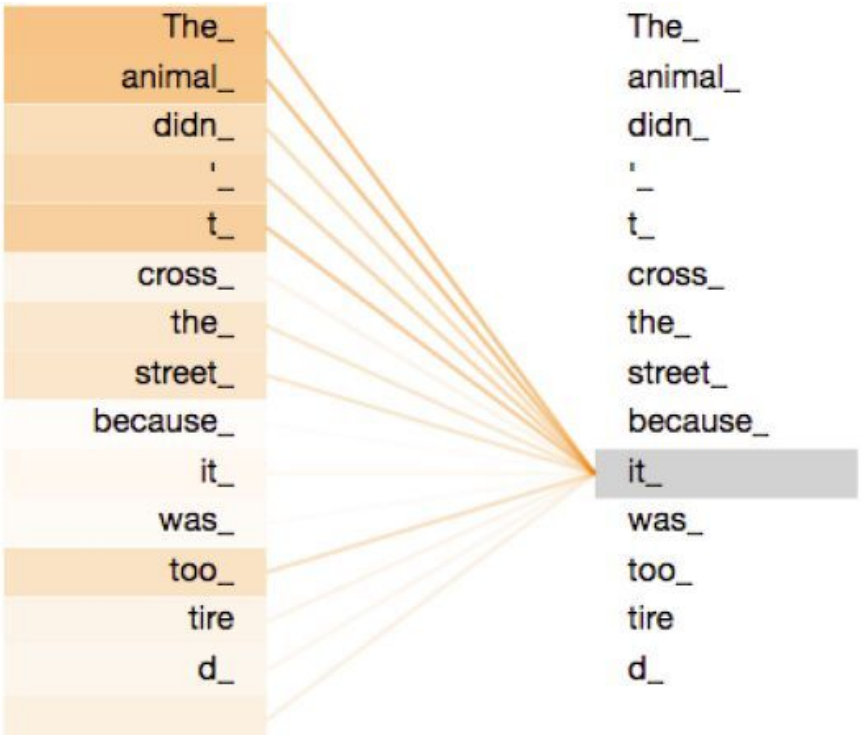

Figure 2.6 Example of the Attention Mechanism in Transformers

Although it is possible to verify the popularity of the transformer model, there are considerable disadvantages that need to be evaluated. As cited by [67], Transformers model training can take large computational and financial resources through the use of GPU and TPU parks, effectively limiting the use of the model to only the largest companies in the technology market, and moving researchers away from model training for specific datasets.

2.6 Large Language Models

One of the most emerging themes in artificial intelligence in recent years is language modeling, specifically generative models. Language models have received significant attention in the literature and, according to [68], can be grouped into four main stages of evolution: Statistical Language Models (SLM), Neural Language Models (NLM), Pre-trained Language Models (PLM), and Large Language Models (LLM).

Statistical language models (SLM) emerged in the 1990s using statistical learning methods. In these models, Markov chains are used to predict the next word given a recent context [69].

Generally, they are used with fixed-size context windows of size n, and thus became known as n-gram language models [70]. However, due to the simplicity of this type of modeling, such models were not capable of handling deep grammatical structures and performed poorly in grouping words from a vocabulary to form complete sentences [71].

The second milestone regarding language models was the emergence of Neural Language Models (NLM). These models sought to calculate the probability of word sequences using neural networks, such as Recurrent Neural Networks (RNN). Important works contributed to this field, such as the study by [72], which introduced the idea of distributed word representations and created a context-based word prediction function. Subsequently, [73] presented *word2vec*, a shallow neural network that also followed the concept of distributed word representation but featured improvements in training efficiency, scalability, and representation quality by using two primary new architectures: Continuous Bag of Words (CBOW) and Skip-gram [73].

During the second half of the 2010s, studies emerged proposing the concept of Pre-trained Language Models (PLM). One of the pioneering models was ELMo, developed by [74], which introduced the idea of capturing contextual word representations through bidirectional neural networks instead of using fixed word representation learning. Shortly after, the Bidirectional Encoder Representations from Transformers (BERT) model appeared, based on the highly parallelizable Transformer architecture and utilizing bidirectional language models with specially designed pre-training tasks on large-scale unlabeled corpora. These pre-trained contextual word representations are highly effective as general-purpose linguistic features, significantly raising the performance of NLP tasks. This study inspired a vast amount of subsequent work, establishing the paradigms of pre-training and fine-tuning [75]. This study was a precursor to the emergence of LLMs, inspiring the first Generative Pre-trained Transformer (GPT) models, such as GPT-2 [76].

In the early 2020s, researchers observed that expanding both the model size and the input data size in PLMs frequently resulted in improvements in the model's ability to perform downstream tasks [77]. These tasks are those to which the pre-trained language model is applied after its general learning phase. Several studies explored increasing the parameters in PLMs to billions, exemplified by GPT-3, which has 175 billion parameters, and PaLM, with its 540 billion parameters, compared to BERT's 330 million parameters [78].

In the work of [79] describe emergent abilities as "abilities that are not present in small models but emerge in large-scale models," which is one of the most notable characteristics distinguishing LLMs from smaller language models. Among the main characteristics noted in emergent abilities is the significant increase in model performance across a range of tasks

requiring context knowledge and reasoning.

Language models do not operate directly on characters or words, but rather on sequences of tokens. For this reason, LLMs typically use subword tokenization schemes, where words are broken down into smaller units called subtokens. In this approach, the text is segmented into a finite set of subwords that can be combined to form both common words and rare terms. Algorithms such as Byte Pair Encoding (BPE) and SentencePiece are popular examples of subword tokenization. In practice, this means that a single word can be decomposed into one or more tokens, and less common terms (e.g., Latin expressions) can still be represented by combinations of subwords present in the vocabulary.

In general, a language model seeks to assign a probability to sequences of tokens $(t_1, t_2, \ldots, t_T)$. A classic way to formulate this problem is to decompose the joint probability as a product of conditional probabilities:

$$P_\theta(t_1, t_2, \ldots, t_T) = \prod_{i=1}^{T} P_\theta(t_i \mid t_1, \ldots, t_{i-1}), \tag{2.7}$$

where $P_\theta(t_i \mid t_1, \ldots, t_{i-1})$ represents the probability of the next token given the previous history, parameterized by a model with parameters θ. Models trained with this objective are called autoregressive and learn to predict the next token in a sequence.

Another important type of objective is masked language modeling, popularized by models such as BERT. In this case, some tokens in the sequence are replaced by a special symbol ([MASK]), and the model is trained to predict the original tokens based on the remaining context. Instead of always predicting the next token, the model learns to reconstruct missing parts of the sentence by exploring information from both sides of the masked token.

It is worth mentioning that two of the most cited transformer models are BERT, created by [75] and already cited in this document, and GPT, created by [76]. As for the tokenization method, BERT uses WordPiece, and GPT uses BPE.

2.7 Causal Structure Learning

Machine learning has obtained good results in both supervised and nonsupervised approaches, whether in labeling results or grouping different values by their intrinsic characteristics. However, machine learning is not as effective when it comes to identifying causality between data, finding whether data A might have caused the existence of data B [80].

Moreover, even though log anomaly detection and log parsing are indeed very useful tech-

niques, they are not dedicated to preventing log errors from happening. They only start working when log files are already generated and when errors have already appeared. A useful strategy to succeed in predicting future log errors is to identify which of the current errors - errors that developers have already seen - are more prone to creating numerous others, and which of the errors have a lower impact on the overall logs.

Naturally, this process would necessarily require a causality detection in the log files to track which errors have a higher impact potential. In this way, DevOps platforms could clearly signal the impact potential of a build log, for example, by lowering the time taken developers to read error logs [11].

For that reason, researchers started a field of research called Causal Structure Learning (CSL). According to [81], causality inference is basically identifying the effect of an experimental intervention given observational data or a combination of observational and interventional data. Therefore, the main goal of the area is to track the consequence of a single piece of data, whether it is introduced in an experimental dataset or when the dataset is already complete and ready for observation.

Another possible strategy is root cause detection. After investigating all errors, a causality detection could point to the original error that caused subsequent ones. In that way, developers would be able to solve problems faster, since quick fixing of the root cause is expected to prevent the system from crashing from errors that would appear later.

Causality, as stated in [80], can be classified into *causal inference*, where the method tries to estimate the effects of a certain action or decision, and *causal discovery*, where the method examines whether there is a set of causal relationships between the data presented. Both approaches are being studied with the adoption of machine learning, opening new research opportunities in many fields of expertise.

As argued by [82], traditional statistical models are based on covariation, not causation. Therefore, the need to identify the clues that lead agents to perceive causal relationships in the data is still a pressing issue. According to the authors, the first variable that agents tend to use to track the causality between the data is the temporal precedence. Although temporal precedence can be an effective way to identify causality in a dataset, the use of temporal data alone is not sufficient to avoid spurious associations caused by unknown factors.

However, causality detection is not a trivial task. According to [80], traditional machine learning is based on the concept of *association* instead of *causality*. Consequently, machine learning shows high accuracy with labeling new items (in which the algorithms are trained with previous and similar items, with supervised learning), grouping data (in which the al-

gorithms try to minimize intracluster distances between items, with nonsupervised learning), or maximizing the cumulative reward of agents (in which the algorithms balance exploration and exploitation to increase the actual gains).

2.7.1 Causal Structures

In order to understand the question mathematically, we can draw a small data set of 3 pieces of data using equations 2.8, 2.9 and 2.10. In this equation system, according to [81], we see a model of structural equations on the three variables X, Y, and Z, and the corresponding independent noise variables u_x , u_y and u_z.

$$X \leftarrow f_1(u_x) \tag{2.8}$$

$$Y \leftarrow f_2(X, u_y) \tag{2.9}$$

$$Z \leftarrow f_3(X, Y, u_z) \tag{2.10}$$

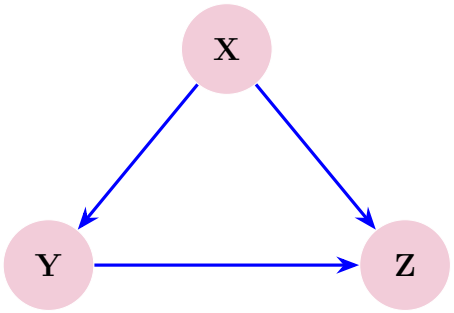

Figure 2.7 Causal structure resulting from the set of equations 2.8, 2.9, and 2.10.

The set of equations will result in a DAG, shown in Figure 2.7. We call two nodes adjacent if there is an edge between them. According to [83], the assumption of acyclicity is natural when considering endogenous data that are defined at certain times, since the intuitive notion of causality dictates that a cause precedes any of its effects. For example, in our picture, Y and Z are adjacent. We also call parents of a node α all nodes that have arrows that point to α. In the opposite direction, we call children of a node α all nodes that are pointed by arrows departing from α. In our scenario, X is the father of the child node Y. Specifically, we call ancestors of a node α all those nodes that have a direct path that begins in α. On

the contrary, descendants are the nodes that are the end of a direct path. In our picture, X is an ancestor of the descendant node Z.

Causal inference is usually called structural due to the fact that each generating equation is presumed to be invariant to possible changes in other equations [84]. This presumption is tantamount to the model since the substitution of the equation of a single node does not compromise the entire model because of their independence. In this way, we can summarize the fundamental aspects of a causal model in the following points, still according to [84]:

- **Causal sufficiency**: According to this principle, we presume that there are no hidden or latent variables, thus avoiding any other equation outside of the system.
- **Causal faithfulness**: According to this principle, only the variables that are d-separated in a DAG will be independent. So, all other variables will depend on this. Two variables are d-separated by Z if they are d-separated by Z along all pathways.

  According to [85] a path is d-separated by variable set Z just in case:

  - The path contains a triple ißmßj or i¸mßj such that m is in Z
  - The path contains a collider ißm¸j such that m is not in Z and no descendant of m is in Z. As mentioned above, Y is a descendant of X if there is a path with unidirectional arrows from X to Y.
- **Acyclicity:** According to this principle, the system requires that the asymmetric part P of the relation (e.g., the subrelation of strict preference) contain no cycles. That is, for no sequence of alternatives $x_1, x_2, \ldots, x_n$ is it true that $x_1 P x_2, x_2 P x_3, \ldots, x_{n-1} P x_n$, and $x_n P x_1$.
- **Gaussianity of the noise distribution:** According to this principle, the system has to consider both Gaussian distributions and t- distributions with various degrees of freedom.
- **One or several experimental settings:** According to this principle, the system needs to have both homogeneous data, where all experiments are from the same experimental setting, and heterogeneous data, where the observations come from different experimental settings.
- **Linearity:** According to this principle, the assumptions of linearity, normality, and error independence are sufficient to calculate all counterfactual queries in the model.

2.7.2 Large Language Models for Causality

In recent years, there has been an increase in the level of scholarly interest in the exploration of the relationship between LLMs and causality, particularly owing to their growing adoption in crucial domains such as healthcare, finance, law, and scientific exploration. Although LLMs have been known to exhibit impressive performance on a wide range of natural language processing tasks, their potential to engage in genuine reasoning about causality is still an area of ongoing scholarly debate.

Early studies have suggested that LLMs have the potential to frequently arrive at correct answers to posed causal questions when the relevant information is already available to them. In this context, [86] have suggested that LLMs have the potential to function as powerful aggregators of existing knowledge of causality. In other words, the authors have suggested that LLMs have the potential to combine information acquired during pre-training to arrive at responses to posed causal questions. Nevertheless, the authors have also suggested that such potential is restricted to recalling or recombining existing information about causality.

The above-mentioned limitations of LLMs have led to the emergence of a series of studies that have attempted to critically examine whether LLMs genuinely understand causality or simply engage in the reproduction of correlations between linguistic information. In this context, the concept of "causal parrots" was proposed by [87], which argued that LLMs simply lack the potential to engage in genuine reasoning about causality as per the framework of Structural Causal Models. Instead, the authors have suggested that LLMs have the potential to exploit correlations between causal information and their textual descriptions. The authors have also suggested that empirical evidence supports the poor potential of LLMs to engage in reasoning about causality.

This assertion has also been backed by other empirical studies. For example, [88, 88] have shown that LLMs' causal prediction capabilities are heavily dependent upon the quality and quantity of causal relations available during their pre-training phase. For example, they have shown that if causal relations are less frequent or more noisy and context-dependent, LLMs' accuracy and confidence are greatly impacted.

Several benchmark-oriented research studies have also been proposed for testing causal reasoning capabilities for LLMs. For example, datasets such as CLadder [89], CausalBench [90], and CausalProbe [91] have proposed novel tasks for testing causal reasoning capabilities for LLMs. These datasets have reported that although LLMs have shown promising causal reasoning capabilities for some surface-level causal queries, their accuracy and confidence are greatly impacted for novel scenarios and deeper causal abstractions. These findings also

support our hypothesis that LLMs are currently operating at a shallow causal reasoning level.

Despite these limitations, some research has also shown promising ways for using causal reasoning with LLMs rather than solely depending upon LLMs for causal predictions. For example, [92] has shown that LLMs can effectively assist in generating causal arguments and causal graphs based upon textual descriptions and outperform traditional causal models for some causal tasks based upon textual descriptions. However, they have also warned that LLMs' outputs can have unpredictable failure modes and should be used as decision-support tools rather than standalone causal models.

Hybrid models that combine LLMs with formal causal models have therefore attracted increased interest. In [93] and [94], the authors propose models where LLMs are used to provide prior knowledge or candidate causal relations, which are then refined by statistical causal discovery techniques. The proposed models have shown enhanced robustness and accuracy compared to solely data-driven approaches, especially in domains where expert knowledge is limited or costly. Agent-based models such as LLM4Causal [95] and Causal Agent [96] also provide LLMs with causal instruments, memory, and reasoning capabilities, thereby enhancing their interaction with structured data and formal causal models.

From a broader perspective, new surveys have synthesized existing literature on causality and LLMs. In [97] and Ma [98] provide a comprehensive overview of the potential benefits of causal inference and LLMs, which can complement each other in various aspects, such as reasoning capacity, explainability, fairness, robustness, and multimodality. The surveys emphasize that although LLMs alone cannot provide causal inference, their integration with causal models offers promise in creating more interpretable and trustworthy AI models.

All these studies point to a common conclusion: LLMs, as they exist currently, do not constitute causal models. The apparent causal capabilities of LLMs primarily rely on correlations learned during pre-training. However, LLMs can act as valuable additional tools when used in conjunction with formal causal models or statistical causal discovery techniques. This conclusion serves as a call to action in exploring new models that combine the power of LLMs with formal causal models, a call that directly aligns with the objectives of this work.

2.8 Detection of Causality in Log Files

Recent research highlights the possibilities of causal discovery by simply using observational data, such as the work of [99].

The main challenge of causality detection is to develop a good heuristic that could work with high precision on a specific data set. Since there is no predefined way to detect causality of

data sets in general, a good approach is to study the structure of the domain being analyzed and develop heuristics tailored to those characteristics.

Although our model can be trained in a very large dataset containing thousands of known events, some external factors could occur and directly impact and increase the appearance of log errors. For example, an interruption of a DevOps software package could trigger many log errors that had not been predicted in our machine learning models.

2.8.1 Methods for Tracking Causality

There are five traditional methods for causality identification: constraints-based methods, score-based methods, hybrid methods, methods based on models of structural equations with additional restrictions, and methods that exploit invariance properties [84]. The description of each of the five methods is shown below:

- **Constraint-based methods**: In constraint-based structure learning, the algorithm assumes acyclicity, causal faithfulness, and causal sufficiency. Then, it conducts numerous conditional independence tests to learn about the structure of the underlying DAG. First, it determines the skeleton of the graph, it determines the v-structures, and then it determines further edge orientations.

  The most cited example of a constraint-based algorithm for causal structure discovery is the Peter-Clark algorithm (PC), by [4]. This algorithm starts by working with a complete undirected graph. Then, it tests each edge to see if there is any conditioning set S, so that X is conditional independent of Y given S. If it is found, the edge between X and Y is removed.

- **Score-based methods**: Assuming acyclicity, causal faithfulness, and causal sufficiency, score-based algorithms use the fact that each DAG can be scored in relation to the data, typically using a penalized likelihood score.

  An example of a score-based algorithm for causal structure discovery is the Greedy Equivalence Search (GES) algorithm, which works by scoring the causal structure using a score-equivalent and decomposable score, such as the regularized log-likelihood [100]. According to [81], a score is called equivalent if it assigns the same value to all DAGs within the same Markov equivalence class and is decomposable if it can be computed as a sum of terms (typically one term for each node) depending only on the local features of the causal structure.

- **Hybrid-based methods**: In this approach, the algorithms combine concepts from both constraints-based and score-based alternatives.

  A case of a hybrid base algorithm is the Max-Min Hill Climbing (MMHC), by [101]. The algorithm builds on the Sparse Candidate algorithm, working by discovering Bayesian networks that are structurally closer to the data-generating network and then performs a Bayesian scoring greedy hill-climbing search to approach and orient the edges, outputting a DAG.

- **Structural equation models with additional restrictions**: According to [102], structural equation models are a multivariate technique to test and evaluate multivariate causal relationships, combining two statistical methods: confirmatory factor analysis, intended to learn latent traits; and path analysis, which is assumed to find the causal relationship between variables by creating a path diagram.

  An approach that uses structural equation models with an additional restriction algorithm for the discovery of causal structures is Lingam [103]. The algorithm works by applying an ICA decomposition, obtaining the correct correspondence between rows and columns of the ICA decomposition, normalizing the rows of the permuted matrix, and then removing the main diagonal and flipping the sign of the remaining coefficients. Finally, causal ordering can be found by permuting both rows and columns (using the same permutation) of the matrix B (containing the estimated connection strengths) to produce a strictly lower triangular matrix.

- **Exploiting invariance properties**: In this class of algorithms, the methods make use of a nonindependent and nonidentically distributed structure in the data and unknown shift interventions on variables.

  An alternative to an algorithm that takes advantage of the properties of invariance is backShift, by [104]. This algorithm tries to learn linear causal cyclic models in the presence of latent variables, using second moments of the data, and performs simple joint matrix diagonalization, applied to differences between covariance matrices.

2.8.2 Causality Identification with Machine Learning

Different approaches have been tried to combine machine learning techniques with CSL. Since the main data structure used to represent causality is a DAG, many methods use graph algorithms to track causality links and to form a new graph from scattered data. This is the case, for example, of [105], which uses Generative Adversarial Networks (GAN), a neural network structure proposed by [106], to learn causality in observational data. The

authors propose the utilization of GANs to find the Nash equilibrium property of networks and propose a novel scoring function to exploit the interactions between different samples.

Another approach is proposed by [107]. In this method, the authors used convolutional neural networks (CNN) to track causality in DAGs. In the first step, the authors use the mechanism of generation of structural causal models (SCM) and propose an integrated neural network model that combines a fully connected layer (FC) and a one-dimensional convolutional layer (1D-Conv). The algorithm then uses the eigenvalues of the weighted adjacency matrix instead of the Hadamard product of the adjacency matrix to represent the acyclicity of the graph.

The work of [108] is also important in the field and has served as a reference point for further methods. In this algorithm, the authors use a deep generative model that generalizes a linear structural equation model, apply a variant of the structural constraint to learn the DAG, using variational autoencoders. Another paper, by [109], also uses autoencoders to generalize gradient-based methods to a graph, allowing for mapping of nonlinear structural equation models.

Also, with the intention of forming a DAG from scattered data, we have the work of [110], beginning with finding an initial cyclic solution to the optimization problem and then employing a Hodge decomposition of graphs and learning an acyclic graph by projecting the cyclic graph to the gradient of a potential function. In another recent work, [111], the authors propose a new algorithm, M2LC, which uses multi-label feature selection as a local CSL problem.

2.8.3 Large Language Models for Log Causality Detection

The increasing sophistication and size of modern software systems have increased the need for efficient log analysis techniques to facilitate debugging, monitoring, and fault identification. The execution logs generated by large-scale and safety-critical systems are semi-structured, voluminous, and difficult to comprehend. Therefore, manual log analysis becomes unfeasible. Recently, advances in Large Language Models (LLMs) have attracted increased interest in their potential to aid log analysis tasks by tapping into their natural language understanding and reasoning abilities.

Initial studies into the suitability of LLMs for log-related tasks have demonstrated both promising aspects and notable limitations. The first systematic evaluation of the suitability of the ChatGPT model for log-related tasks is reported in [112]. According to their findings, although LLMs can accomplish basic interpretation and summarization of log data, their reliability is inconsistent, robustness to heterogeneous log formats is lacking, and their ability

to handle large log volumes remains limited. These observations suggest that general-purpose LLMs are not sufficient to support log-related tasks in their current form.

Several studies have also explored the application of LLMs for root cause analysis (RCA), which is a critical yet labor-intensive process in software testing and system operations. For instance, in safety-critical domains such as railway systems, [113] present a benchmarking study on the application of several state-of-the-art LLMs to real-world execution logs generated through digital twin-based testing environments. Although the performance of some models indicates promising reasoning capabilities, overall accuracy remains constrained, with performance varying significantly across models, highlighting the difficulty of applying LLMs to complex and domain-specific log data. Similar conclusions are drawn in [114], where the authors introduce the OpenRCA benchmark for evaluating LLMs on root cause identification using heterogeneous telemetry data. Their results indicate that even state-of-the-art models are unable to reliably perform root cause identification, with success rates remaining low in realistic scenarios.

Other research has investigated the use of LLMs for additional core log analysis tasks, such as log parsing. For example, [115] introduce CausalLog, a framework that integrates causal intervention principles with LLM-based log parsing to address biases associated with experiential shortcut behaviors learned during pre-training. By leveraging counterfactual rewriting and n-gram adjustment scores, the proposed approach demonstrates improvements in both log grouping and template extraction accuracy, highlighting the potential contribution of causal reasoning to LLM-based log analysis.

Further studies have examined the deployment of LLMs in industrial settings. [116] present a case study on the application of LLMs for IT software support, in which large volumes of log data are analyzed for automated issue diagnosis and summarization. The results indicate substantial benefits in terms of productivity gains and cost savings. However, it should be noted that these improvements are primarily achieved through system design choices and inference optimizations rather than through the intrinsic reasoning capabilities of LLMs.

While most LLM-based log analysis methodologies focus on pattern recognition or textual interpretation, a parallel body of work argues for the integration of causal inference into system diagnosis workflows. In particular, [6] propose a human-in-the-loop framework that applies causal inference over logs for diagnosing large production systems. Their results show that causal modeling can significantly reduce diagnostic effort while providing higher-quality explanations, emphasizing the need to move beyond purely correlation-based approaches and incorporate causal reasoning into log analysis methodologies.

Overall, the literature suggests that while LLMs are valuable tools for interpreting and sum-

marizing log data, their ability to perform complex log analysis tasks remains limited. Performance volatility, sensitivity to context, scalability challenges, and the lack of principled causal reasoning consistently emerge as key limitations. Recent advances that combine LLMs with causal inference or structured representations represent promising directions for mitigating these issues. These findings motivate continued research on hybrid approaches that combine the flexibility of LLMs with the rigor of causal reasoning for log analysis.

CHAPTER 3 OVERVIEW

In this chapter, we discuss the main gaps identified in the literature that motivated our research and summarize how these gaps were addressed through the academic publications that compose this thesis.

3.1 Gaps in the Literature

Software log analysis has been widely studied in the context of monitoring, anomaly detection, and root cause analysis. However, several limitations remain when dealing with logs from modern large-scale systems. As an example, traditional log parsing approaches typically rely on handcrafted rules, regular expressions, or clustering-based heuristics. While effective in controlled settings, these approaches often struggle to generalize across heterogeneous log formats and evolving software systems who might change its log vocabulary. The literature lacks robust learning-based methods capable of adapting to unseen log structures without heavy manual tuning.

Second, topic modeling techniques applied to logs often inherit assumptions from natural language processing that do not hold for software logs. Logs are semi-structured, sparse, and contain domain-specific tokens with fragmented vocabularies. Classical topic modeling methods, such as Latent Dirichlet Allocation (LDA), rely heavily on word co-occurrence statistics and may fail to capture meaningful semantic relationships in log data. In logs, having a high frequency does not mean that a token is more important, since it may only represent a frequent alert. There remains a gap in adopting contextual language representations to extract coherent semantic themes that reflect underlying system behaviors.

Finally, causal discovery in software logs remains an open and challenging problem. Existing constraint-based methods, such as the PC algorithm, and time-series-based approaches, such as Granger causality, were originally designed for tabular or dense numerical data. These methods do not naturally accommodate the sequential, sparse, and high-dimensional nature of log sequences. Moreover, most existing approaches rely on frequency-based statistics or local windows, without leveraging the full contextual information embedded in log sequences. To the best of our knowledge, limited work has explored the integration of modern language models into the causal discovery process for software logs.

3.2 Steps of the Research

The research presented in this thesis is structured into three complementary studies, each addressing a distinct but connected challenge in software log analysis.

In Chapter 4, we introduce a language-model-based approach for log parsing. The proposed method uses contextual language representations to automatically extract structured log templates from raw log messages. Unlike rule-based and clustering-driven parsers, our approach learns implicit structural patterns directly from log data, enabling improved robustness to format changes and unseen patterns. Experimental evaluations on benchmark log datasets demonstrate that the proposed method achieves competitive or superior performance compared to the state of the art log parsing techniques, while maintaining strong generalization capabilities.

In Chapter 5, we extend the use of language models to the problem of topic modeling and log summarization. The proposed framework combines transformer-based embeddings with clustering techniques to identify coherent semantic topics within large collections of parsed log messages. By integrating contextual representations with log-specific structural information, the method captures latent system behaviors that are not apparent through lexical similarity alone. Extensive experiments on multiple datasets show that the approach produces more coherent and interpretable topics compared to traditional topic modeling methods.

In Chapter 6, we address the problem of causal discovery in software logs using a language-model-driven probability estimation framework. The proposed method adopts causal language models to estimate conditional probabilities between log events and integrates these estimates into a constraint-based causal discovery procedure using conditional mutual information. By replacing raw frequency statistics with model-derived probability distributions, the approach improves robustness in sparse and high-dimensional log data. Experimental results in real-world industrial logs demonstrate significant improvements in both scalability and accuracy compared to traditional causal discovery methods.

CHAPTER 4 ARTICLE 1: USING TRANSFORMER MODELS AND TEXTUAL ANALYSIS FOR LOG PARSING

Authors

Vithor Bertalan and Daniel Aloise. Published at the 34th IEEE International Symposium on Software Reliability Engineering (ISSRE 2023). Submitted on 2023-07-29. [1]

Contributions

Vithor Bertalan: Development and implementation of the proposed model, textual analysis, experiments, validation, log data analysis, literature review and writing.

Daniel Aloise: Development of the proposed model, writing, results analysis and supervision.

Abstract

Log parsing has become an essential tool for extracting valuable information from a vast volume of log lines. It involves identifying standard patterns and extracting templates from these lines, enabling researchers and companies to employ advanced mining techniques like log deduplication and log anomaly detection. However, existing log parsing approaches have limitations. They often operate on small batches of log text and lack consideration for the entire context, necessitating prior knowledge of the log dataset. Furthermore, there is a scarcity of practical experience reports on the utilization of these log parsing approaches in the literature. In our paper, we address these challenges by proposing a novel log parsing approach that combines Transformers with a customized textual analysis. This textual analysis balances the density clustering of similar log lines, the calculation of word frequencies inside each cluster and the presence of the words inside an English vocabulary to parse new log lines. Our method outperforms existing parsing methods in terms of accuracy and operates as an unsupervised learning approach, eliminating the need for prior knowledge. Additionally, we present a proof-of-concept application of our parsing method in an industrial setting, showcasing its practical implementation.

[1] Available at [117]

4.1 Introduction

Log files are an essential component of modern software engineering. Their importance has grown significantly in recent years, mainly because they contain valuable information about the behavior of software applications. Log files are generated by the application as it runs, and they record various events and actions performed by the system, such as user actions, errors, warnings, and system events [118]. Thus, they constitute a valuable source of knowledge for software engineers, providing insights into the performance, behavior, and issues of an application. The analysis of log files helps engineers understand how the application is operating, identify potential issues or bugs, and troubleshoot errors [119].

A practical application of log files is monitoring system performance over time. Software engineers can use log files to track the usage of system resources, such as memory or CPU usage, and identify trends or patterns that may be impacting system performance. Additionally, log files are crucial for auditing and compliance purposes. They provide a record of system activities that can be used to ensure adherence to regulations and internal policies [119]. However, modern software applications generate a tremendous amount of data, including logs that can number in the millions or even billions of entries [32]. Without an effective tool to help their work, engineers would have to sift through all of these entries manually, which is impractical and time-consuming.

Log parsing is the process of searching, filtering, and analyzing log data, enabling software engineers to quickly identify patterns and trends [120] in log files generated by software applications. Through log parsing, we are able to classify different log lines into basic log templates, so as to organize and reduce the amount of data to be observed. The process of log template composition involves capturing the tokens that exhibit repetition and remain constant across lines. Conversely, tokens that appear in repeated lines but display variability are categorized as variables.

In Table 4.1, an example of log file parsing is presented. By analyzing the log files, we can observe that multiple lines begin with the tokens "*visible*" and "*is*", while the subsequent token varies from line to line. As a result, the tokens "*visible*" and "*is*" are recognized as static, while the following token is considered a variable and replaced by a wildcard symbol (in this case $< * >$). By employing this approach, we can consolidate distinct lines into a unified template, as exemplified by the first and second lines in Table 4.1, effectively grouping together similar lines that exhibit minor variations in tokens.

The parsing process enables us to consolidate multiple lines into a single log template, resulting in a significant reduction in computational complexity compared to analyzing each

line individually. This consolidation provides a streamlined representation of the log data, which is particularly beneficial for subsequent activities such as log anomaly detection or log deduplication. By focusing on the parsed lines rather than the raw log data, algorithms can operate more efficiently and effectively, as they can directly leverage the structured information contained within the log templates.

Table 4.1 Classification of logs into log templates

Original Log Line	**Parsed Log Line Template**
visible is system.charge.show	visible is <*>
visible is system.exit	visible is <*>
visible is system.call.count gt 0	visible is <*> gt <*>

Moreover, log parsing also allows engineers to focus their attention on specific areas of interest within the log data. For example, engineers can use filtering tools to isolate errors, warnings, or other specific types of log entries [121]. They can also search for specific keywords or phrases within the log data, such as the name of a particular feature or component [2]. In addition, log parsing enables engineers to monitor system performance in real-time. They can set up alerts and notifications based on specific criteria, such as the occurrence of a critical error or a sudden spike in system resource usage. This proactive monitoring approach can help prevent issues before they become serious problems.

In recent times, researchers have been exploring the potential for enhancing log parsing techniques using machine learning [120, 122]. Traditional log parsing methods rely on predefined rules or heuristics to identify patterns or anomalies in log data. However, these methods may face limitations in effectively handling complex or dynamic log data. They may struggle to adapt to new types of log data or changes in the system [121]. In contrast, machine learning offers a solution by enabling the automatic learning of patterns and trends in log data, without the reliance on predefined rules or heuristics [123].

Amidst the plethora of machine learning models, Transformers are taking the lead in performance gains. Transformer models, which are a type of neural network architecture, can also be used to improve log parsing by enabling more advanced analysis and processing of log data. Transformer models have been shown to be particularly effective at processing sequential data, such as text, making them well-suited for analyzing log data [67]. One of the key benefits of transformer models is that they can learn to represent text data in a more meaningful and structured way than traditional methods, since they usually process the whole batch of the text at once, instead of separating the context into smaller subsets [67]. They can learn to encode the relationships between different log entries, which can be

important for identifying patterns and trends in the data [124].

In order to improve their performance, Transformers can also be combined with traditional textual analysis techniques to improve log parsing, such as the work of [125], that combined Transformers with subword tokenization. Traditional textual analysis techniques, such as the analysis of word frequencies, can be used to identify specific patterns in the log data, while Transformers can be used to learn more complex patterns and relationships between log entries.

To help in these endeavors, in this paper, we propose a new log parsing architecture that combines the data processing capabilities of Transformers with a novel word analysis approach, integrating the analysis of word frequencies with their presence in the English vocabulary. Our primary objective is to develop a framework that delivers high performance across various datasets while maintaining a controlled level of complexity. By doing so, we aim to ensure that professionals without prior machine learning experience can easily utilize our framework without hassle. Thus, our framework dismisses a supervised learning step as well as the need for data pre-processing, making it flexible enough to deal with different datasets without a specific data tailoring and previous domain knowledge.

To evaluate the effectiveness of our framework, we compare our results to other log parsing methods available in the LogHub datasets [118], a group of log datasets from different system sources. We also discuss some of the problems found in LogHub, which could hinder the accuracy of methods using their standards for performance evaluation. Furthermore, to demonstrate the practical applicability of our framework, we have conducted tests in collaboration with our industry partner. This collaboration allowed us to validate the effectiveness of our log parsing approach using real-world data, and to obtain valuable feedback from software engineers.

The contributions of our work can be summarized as follows:

- Our log parsing approach adopts a dataset-agnostic methodology, eliminating the need for prior knowledge or familiarity with specific datasets. Thus, our approach provides a versatile solution, enabling companies to monitor and parse multiple log files seamlessly, without requiring intricate tailoring or customization for each dataset.
- Our method provides the flexibility to incorporate regular expressions, allowing users to customize the parsing process for enhanced effectiveness.
- Our parsing process is designed to be accessible and user-friendly, requiring minimal expertise in machine learning. Users are only tasked with fine-tuning a parsing thresh-

old, experimenting with different values to determine the optimal configuration that aligns with their specific requirements.

The rest of the paper is organized as follows. Section II provides background for our research. Section III illustrates the methodology of our framework. Section IV presents the theoretical results, comparing the accuracy of our method with other log parsing approaches. Section V reports about the use of our log parsing method by our industry partner, as well as the lessons learned from this experiment. Section VI writes about recent researches related to our work. Finally, Section VII draws the conclusion and indicates some directions for future work.

4.2 Background and Motivation

Given the overwhelming volume of information generated by software and server monitoring systems in today's context, it is crucial to possess a tool that can effectively distinguish between log lines that are significant and warrant further analysis from those that can be safely disregarded. Log parsing plays a pivotal role as the initial step in this process as it paves the way for various log mining tools, which are instrumental in extracting valuable information. Notable examples of such tools include log extraction [126], log deduplication [127], and log anomaly detection [128].

4.2.1 Limitations of Tradicional Log Parsing

According to [129], several log parsers exhibit characteristics that may limit their practical applicability. A common approach involves employing pre-processing techniques to capture the textual structure of log lines, typically through the use of regular expressions (e.g. [31, 29, 130]). Thus, methods that use pre-processing are able to produce results with higher accuracy levels than methods that consider the text without any modifications. However, the usage of pre-processing makes the method highly specific to datasets, losing its adaptability to other log structures.

A second approach that log parsers generally adopt is the processing of the textual information in batches, representing small samples of the overall text. This strategy is used by log parsers such as Loghound [47], Molfi [32], and SLCT [51]. Partitioning the text into smaller samples is a beneficial approach to reduce the computational complexity of the algorithm. By processing smaller portions of data, the overall computational load is decreased. However, it is important to note that in the context of textual analysis, the ability to comprehend the

intrinsic relationships between tokens can be highly valuable. This enables capturing the complete meaning conveyed by the log lines.

4.2.2 Transformer Models

A Transformer model is a neural network that learns context and thus meaning by monitoring relationships in sequential data like the words in a sentence. Transformer models apply an evolving set of mathematical techniques, calling *attention* or *self-attention*, to detect the subtle ways in which even distant data elements in a series influence and depend on each other [65]. Using Transformers, we can process texts integrally, being able to learn from the relationships and positions between tokens across all text content.

Due to the attention mechanism, unlike traditional neural networks, Transformers do not require sequential data to be processed in order. Instead, the attention operation identifies the context for any position in the input sequence [65]. For example, if the input data is a natural language sentence, Transformer does not need to process the beginning before the end. Rather, it identifies the context that gives meaning to a word in the sentence. Due to this feature, Transformer allows much more parallelization than traditional neural networks, thus reducing training times.

Transformers can offer several advantages to log parsing, namely:

- Improved Accuracy: Transformers can learn to identify and trace complex patterns and intrinsic relationships within log data, due to the utilization of attention mechanisms [131]. This can lead to more accurate parsing of log data and faster identification of issues to be corrected in real time. Transformers are able to process the whole text is an input, thus being able to detect the connections between the tokens throughout the entirety of the content.

- Automated Processing: Transformers can be used to automate the log parsing process [132], reducing the need for manual analysis and increasing the speed at which log data can be processed.

- Scalability: Transformers are highly scalable and can process large volumes of log data quickly and efficiently [133], making them well-suited for processing the massive amounts of log data generated by modern applications.

- Flexibility: Transformers can be trained on a wide range of log data types and formats [134], making them highly flexible and adaptable to different types of log data. This

is very important to our scope, since we do not have a single and unified format of log data for each system available.

- Continuous Improvement: Transformers can be continuously trained on new log data, allowing them to improve their accuracy over time as they learn from new data [135].

Finally, the primary objective of our work is to develop a dataset-agnostic log parsing method that offers flexibility in handling log lines under different formats. Transformers enable software developers and system administrators to perform log parsing without prior knowledge of the specific log content.

4.2.3 Practical Applications of Transformer-Based Log Parsing

In Table 4.2, we present examples of log lines extracted from the Android, HDFS, and Linux datasets, respectively. Each dataset exhibits distinct log line styles. For instance, the Android log line includes a numerically written date, a timestamp with milliseconds, and numerical values with minimal punctuation. In contrast, the HDFS log line features a different datestamp and additional information. Finally, the Linux log line consists of an alphabetically written date accompanied by various punctuation marks, such as parentheses, brackets, and semicolons.

To maintain the dataset-agnostic feature of our method, we have deliberately chosen not to pre-process the data. Consequently, our log parser needs to exhibit flexibility in handling different log line formats. In this regard, Transformers serve as a suitable model due to their ability to effectively process diverse textual inputs.

Table 4.2 Examples of log styles from LogHub

Log Dataset	Example of log line
Android	03-17 16:13:38.859 2227 2227 D TextView: visible is system.time.showampm
HDFS	081109 204525 512 INFO dfs.DataNode$PacketResponder: PacketResponder 2 for block blk_572492839287299681 terminating
Linux	Jun 14 15:16:02 combo sshd(pam_unix)[19937]: check pass; user unknown

Using Transformers and our own word analysis method, that combines the analysis of word frequencies and their presence in the English vocabulary, our log parser is designed to help

our industry partner, a multinational company with several different DevOps workflows that sustain software development and systems maintenance.

Given the fundamental role of a robust log parsing method in various activities like log anomaly detection and log deduplication, we are eager to collaborate with our industry partner in the development of an advanced log parsing approach that yields favorable outcomes in their day-to-day operations. Moreover, our log parsing method can serve as a valuable tool for identifying performance issues and optimizing system efficiency to enhance the overall user experience. This aspect holds particular significance for our partner, as their operations heavily rely on real-time system stability.

4.3 Methodology

In this section, we outline the workflow of our log parsing approach. To the best of our knowledge, no prior research has explored the application of Transformers combined with NLP techniques in the field of log parsing. Our novel approach combines the power of Transformers alongside NLP techniques, including word frequency analysis and the utilization of an extensive English vocabulary. Additionally, we incorporate state-of-the-art tokenization techniques to enhance the effectiveness of our log parsing methodology. By combining these innovative elements, we aim to push the boundaries of log parsing and introduce new insights in this domain. We show the steps of our methodology in Figure 4.1.

4.3.1 Data collection

The initial step of our method involves capturing the raw log lines, preserving their original form without any specific preprocessing. This approach ensures that we retain the integrity of the logs during the subsequent analysis stages. In order to assess the accuracy of our method, we used LogHub's datasets [118]. These datasets encompass raw log lines collected from various sources, including Hadoop distributed file system logs, Hadoop map reduce job logs, Spark job logs and ZooKeeper service logs. The datasets account to a total of 440 million log messages that amounts to 77 GB in size. The authors of LogHub's model conducted extensive testing of different parsing models to compare their respective accuracies. In our evaluation, we performed comparisons to the following log parsing methods: AEL [31], Drain [29], IPLoM [47], Lenma [136], LKE [130], LogMine [137], MoLFI [32], SHISO [138] and Spell [139]. To evaluate the classification performance of each parsing method, the LogHub authors manually labeled and created ground-truth data for each of the log sources, such as Hadoop and Spark, by sampling 2,000 log lines from each source. The accuracy of a parsing

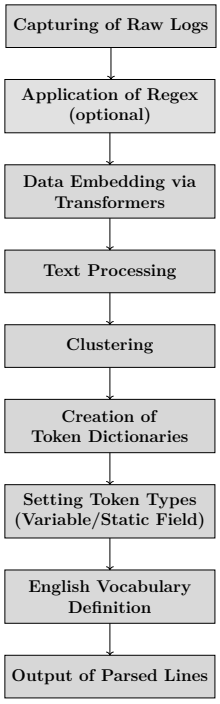

Figure 4.1 Methodology of the parsing process.

method is then determined by calculating the ratio of correctly predicted template lines to the total number of lines. For instance, if a method accurately predicts the templates of 1,500 lines, its accuracy is calculated as 0.75 ($\frac{1500}{2000}$).

To gain a clearer understanding of LogHub's accuracy assessment, let us revisit Table 4.1. The initial log lines are presented in the first column, while the parsing methods transform them into the corresponding output. A successful prediction is achieved when the results align with those in the second column. To calculate the accuracy, the total number of successful predictions is divided by 2,000, which serves as the denominator in the accuracy calculation.

4.3.2 Application of regular expressions

Following the raw log data capturing process, we have included an optional step to incorporate regular expressions. In our tests, we deliberately refrained from employing any regular expressions to maintain the versatility of our model in handling diverse log data. Nevertheless, as part of our practical implementation plan for our industry partner, we have chosen to retain this step to assess its significance within a real-life context. By doing so, we aimed to comprehensively evaluate the potential impact and effectiveness of regular expressions in our log parsing methodology.

4.3.3 Data transformation

Subsequently, we employed a Transformer architecture to convert all textual log lines into dense vectors, commonly known as embeddings. The selection of the Transformer architecture stems from its remarkable capability to capture the semantic meaning of text, preserving the contextual nuances and interrelations between words, as discussed in Section 5.1. We used the RoBERTa model [140] using 12 hidden layers. The hidden size dimension is set to 768, and it employs 12 self-attention heads. The model's vocabulary size is determined by the text length of each dataset. To ensure this process, we utilized the SentenceTransformers Python framework developed by [62]. We experimented with various hyperparameters inside a grid search, including learning rates and batch sizes, and observed the resulting clusters for several datasets to establish a standardized approach. Subsequently, we applied these values across all datasets to avoid over-specialization for each individual dataset.

It is worthy to emphasize that no preprocessing (such as tokenization) is conducted at this stage, aiming to present the raw text in its entirety to the Transformers, thus ensuring a fair representation within the machine learning pipeline. This approach aligns with the principle of avoiding any bias or unfairness during the data preprocessing phase, as pointed out by [141].

4.3.4 Text processing

After the data transformation, our method continues by tokenizing the log dataset. We experimented with various tokenization techniques and ultimately chose a customized tokenizer that separates tokens based on blank spaces and specific punctuation marks, such as commas, semicolons, and parentheses. We observed that certain publicly available tokenizers were inadvertently splitting crucial punctuation marks for our log parser, such as periods, resulting in inconsistencies like the separation of IP addresses. Therefore, we opted for a tokenizer that ensures the integrity of such punctuation marks. Since we wanted to keep the dataset agnosticism of our method, the same customized tokenizer is used to every dataset processed, without any specific modifications.

To standardize the text, all tokens were lowercased before the calculation of the frequency dictionaries, since logs usually have many tokens with some characters or whole words in upper case. A pratical example of our tokenizer is shown in Figure 4.3.

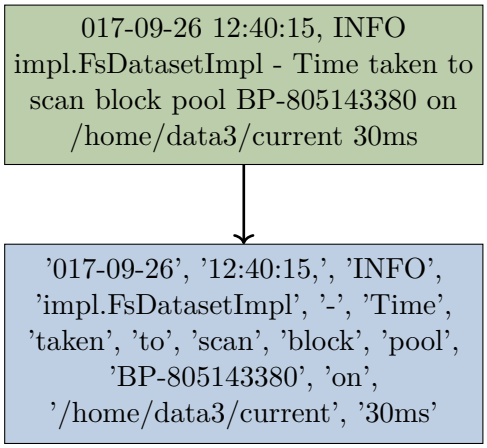

Figure 4.2 Example of sentence processed with our custom tokenizer

4.3.5 Clustering data

Once the log lines have been transformed in numerical vectors by our Transformer model and processed, our next step consists in clustering such vectors to identify meaningful patterns. The rationale behind this approach is illustrated in Figure 4.3. In this particular example, we have a set of 16 raw log lines that are grouped into 5 distinct clusters. This reduction step allows to scale log analysis that would otherwise be extremely hard to accomplish given the size of industrial log files.

For each dataset, whether sourced from LogHub or generated by our industry partner, we have employed the same clustering method. Currently, our clustering algorithm of choice is HDBSCAN (*Hierarchical Density-Based Spatial Clustering of Applications with Noise*), originally proposed by [64]. HDBSCAN is a nonparametric clustering technique that does not rely on parametric distribution assumptions. It incorporates a crucial hyperparameter, denoted as *minpts*, which specifies the minimum number of points within a given ϵ radius for a point to be considered a core point. Once the clusters are formed, HDBSCAN proceeds to prune out clusters that contain fewer points than the specified *minpts* value. The Euclidean distance measure serves as the default distance metric, and we have set a minimum requirement of 10 log entries per cluster to ensure sufficient representation.

By adopting HDBSCAN, we can effectively identify clusters from the log data, facilitating subsequent analysis and gaining insights into distinct log patterns. An example of clustering using HDBSCAN is shown in Figures 4.4 and 4.5. Figure 4.4 shows the raw log data of the LogHub HDFS dataset, numerically embedded by the Transformer, and plotted in 2D using T-SNE [142]. Figure 4.5 shows the clustering obtained by HDBSCAN over this dataset.

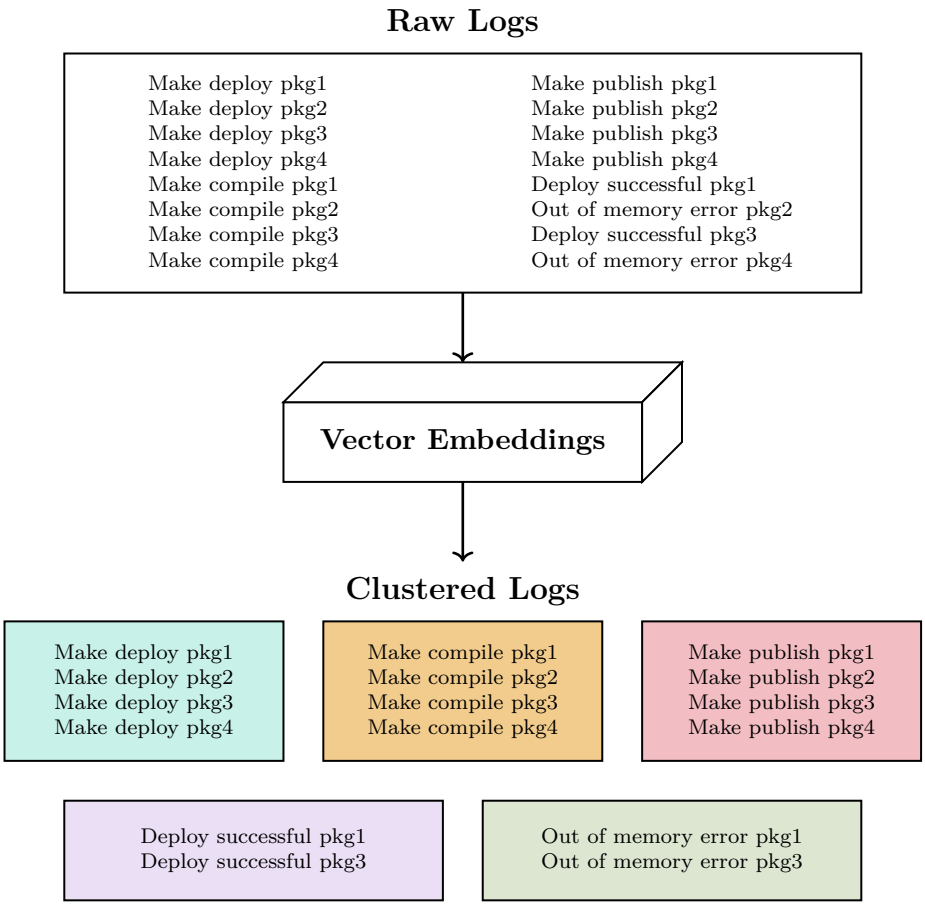

Figure 4.3 Example of clustering raw log lines.

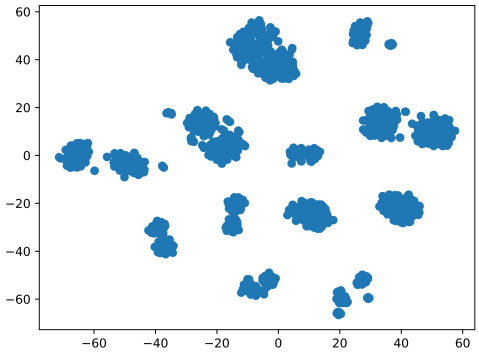

Figure 4.4 Raw HDFS dataset log data.

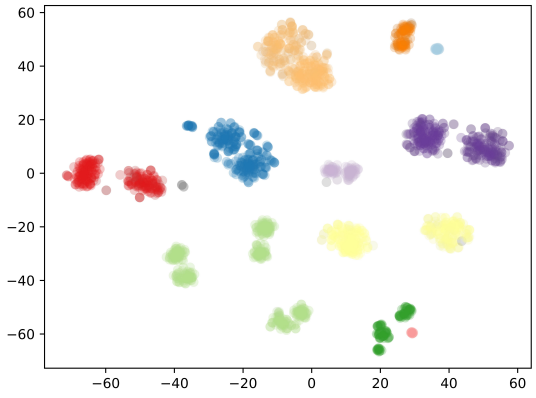

Figure 4.5 Clustered HDFS dataset log data using HDBSCAN.

4.3.6 Creation of frequency dictionaries

After completing the clustering and text processing steps, our focus shifted towards differentiating static fields from variables in the tokenized logs. To accomplish this, we constructed a frequency dictionary for the tokens found in each identified cluster. By analyzing the relative frequencies of tokens within each cluster, we were able to devise an initial rule for segmenting static elements that remain constant and dynamic variables that vary across log entries.

The relative frequence rf_i of each token i in cluster C is computed as

$$rf_i = \frac{f_i}{max_{i \in C} f_i}, \tag{4.1}$$

where f_i refers to the frequency of a token i in cluster C divided by the maximum frequency among all tokens i in that cluster.

To illustrate our method, let us consider the final cluster in Figure 4.3 for which we obtain the frequency dictionary shown in Table 4.3.

Table 4.3 Frequency dictionary for the last cluster of Figure 4.3

Token	Cluster	Frequency	Relative Frequency
Out	2	2	1
Of	2	2	1
Memory	2	2	1
Error	2	2	1
Pkg2	2	1	0.5
Pkg4	2	1	0.5

Having generated the dictionary encompassing all clusters and their associated tokens, we use a *parsing threshold* β to effectively distinguish static fields from variables by analysing their relative frequencies. In our example, any β ranging from 0.5 to 1 ($0.5 < \beta \leq 1$) would classify tokens such as *error*, *null*, *pointer*, and *exception* as static fields, while tokens like $pkg2$ and $pkg3$ would be identified as variables. By assigning labels to each token, our method gains the ability to parse and interpret individual tokens based on their respective labels.

It is worthy to note that our parsing threshold mechanism ensures flexibility in accommodating the requirements of software developers and DevOps engineers, without necessitating in-depth expertise in machine learning. By employing this mechanism, users are empowered to adjust the threshold value according to their specific needs during the execution of the method. In real-world scenarios, log parsing lacks a standardized ground truth and heavily relies on the desired outcomes defined by the user.

Even within LogHub, the ground truth is established based on the manual understanding and separation of static fields from variables by [118], a process that inherently entails subjectivity. By adopting our parsing threshold approach, users gain the ability to test different scores in order to select log templates that align with their specific demands, tailoring the parsing algorithm to their unique requirements, objectives and preferences.

Our method initiates by optionally applying regular expressions to the input data. Next, it proceeds to train our Transformer models and generate the corresponding embedding files, if applicable. The algorithm then loops back to the raw log data and performs tokenization, converting all tokens to lowercase. The embeddings are subsequently clustered using HDBSCAN.

From there, the algorithm iterates through each cluster, analyzing the tokens within, to determine the total frequencies and relative frequencies of each token. For each token, if its relative frequency falls below our parsing threshold, it is classified as a variable. If the relative frequency surpasses the threshold but the token is not present in the English language, it is also labeled as a variable. Otherwise, it is categorized as a static field.

4.3.7 Verification of English vocabulary

In addition to the parsing threshold, we aimed to augment our algorithm with a complementary approach that analyzes the English vocabulary. With that purpose, we built a comprehensive dataset comprising words used in the English language to determine whether a given token is issued from the English language or not. We have curated a dataset containing 631,940 tokens obtained from Aspell, the spell checker utilized by GNU/Linux. In

our method, following the transformation of the frequency-based dictionary, we additionally verify whether a given word belongs to our English dictionary.

If the word does not match any entry in our English dictionary, it is then categorized as a variable, being replaced by a standard wildcard $< * >$. Therefore, a field is categorized as static only if it stands above our parsing threshold, and if it does belong to the English language. If those two conditions are not met, the token becomes a variable. This step allows us to differentiate between words that align with standard English language usage and those that deviate from it, facilitating the identification of variables within the log data.

It is important to mention that a similar technique has been employed in prior research studies, such as that conducted by [139], albeit with a dataset comprising 588,054 tokens. Additionally, [143] also employed a comparable method, utilizing the Python NLTK framework to ascertain whether a token corresponds to an English word. This linguistic analysis is particularly beneficial in capturing the semantic meaning of variables within the log data, enhancing the overall accuracy and interpretability of the parsing process. By incorporating these language-based techniques with our prior frequence-based dictionary, we can effectively enhance the capabilities of our log parsing algorithm and achieve a more comprehensive understanding of the log entries.

4.4 Empirical Evaluation and Results

In this section, we commence by presenting the results obtained using LogHub's parsing accuracy metric. Subsequently, we delve into a comprehensive analysis of the metric's validity by examining certain aspects of the ground truth lines that are deemed correct. This critical examination allows us to address potential limitations and considerations related to the accuracy evaluation process. By scrutinizing the ground truth lines, we aim to gain deeper insights into the reliability and robustness of the parsing accuracy metric employed in LogHub. This analysis helps shed light on any potential discrepancies or areas for improvement in the evaluation methodology.

4.4.1 Evaluation using LogHub metrics

After finishing our method, we compared our results with the other log parsing techniques available in LogHub. LogHub compares results by evaluating precision, recall, f-1 score and their custom *parsing accuracy* metric. As mentioned in Section 4.3, this parsing accuracy is calculated by dividing the total number of correctly predicted template lines, according to a pre-defined ground truth file, by the total number of lines of the ground truth files, which

is 2,000 lines for each dataset. The accuracy results can be seen in Table 4.4, F1 measures can be seen in Table 4.5, precisions can be seen in Table 4.6, and the recalls can be seen in Table 4.7.

Table 4.4 Accuracies of the methods using LogHub's framework. Top values in each column in bold.

	HDFS	Hadoop	Spark	Zookeeper	BGL	HPC	Thunderbird	Windows	Android	OpenStack
AEL	0,997	0,869	0,905	0,921	0,957	**0,903**	0,941	0,689	0,681	0,757
Drain	0,900	0,604	0,918	0,962	0,472	0,410	0,828	0,708	0,159	0,121
IPLoM	**1,000**	0,954	0,920	0,961	0,939	0,829	0,663	0,563	0,711	0,330
Lenma	0,997	0,885	0,883	0,840	0,689	0,829	**0,943**	0,565	0,879	0,742
LKE	**1,000**	0,769	0,633	0,437	0,645	0,574	0,812	**0,989**	0,910	**0,787**
LogMine	0,850	0,869	0,575	0,687	0,724	0,784	0,918	0,992	0,504	0,743
MoLFI	0,997	0,886	0,418	0,839	0,957	0,810	0,655	0,711	0,701	0,213
SHISO	0,997	0,867	0,906	0,660	0,711	0,324	0,576	0,700	0,585	0,721
Spell	**1,000**	0,777	0,905	0,963	0,786	0,654	0,843	0,988	**0,918**	0,764
Our Method	0,997	**0,983**	**0,996**	**0,990**	**0,974**	0,755	0,934	0,693	0,450	0,491

Table 4.5 F1 measures of the methods using LogHub's framework. Top values in each column in bold.

	HDFS	Hadoop	Spark	Zookeeper	BGL	HPC	Thunderbird	Windows	Android	OpenStack
AEL	**1.000**	0.996	0.991	0.995	**1.000**	**0.992**	0.999	0.999	0.940	0.986
Drain	0.990	0.883	0.992	0.999	0.310	0.711	0.994	0.900	0.699	0.684
IPLoM	**1.000**	0.996	0.992	0.999	0.999	0.978	0.999	0.995	0.949	0.909
Lenma	**1.000**	0.997	0.989	0.998	0.939	0.981	**1.000**	0.995	**0.992**	**0.994**
LKE	**1.000**	0.867	0.226	0.994	0.943	0.177	0.982	**1.000**	**0.992**	0.936
LogMine	0.999	0.996	0.841	0.985	0.970	0.976	0.999	**1.000**	0.893	0.994
MoLFI	**1.000**	0.891	0.498	0.953	**1.000**	0.982	0.999	0.912	0.977	0.727
SHISO	**1.000**	0.998	0.992	0.993	0.994	0.541	0.911	0.913	0.844	**0.994**
Spell	**1.000**	0.920	0.991	**1.000**	0.957	0.986	0.994	**1.000**	**0.992**	**0.994**
Our Method	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	0.991	**1.000**	0.999	0.727	0.916

Our method demonstrates a high accuracy across a diverse range of datasets, namely Hadoop, Spark, Zookeeper, BGL, and HDFS. Notably, it achieves the highest overall accuracy among these datasets. It is important to highlight that we observed distinct behaviors in relation to the parsing threshold for each dataset present in LogHub. Each dataset exhibited a unique pattern that influenced the optimal performance of our parsing approach. For instance, the HDFS dataset showcased improved results when the parsing threshold was lowered, suggesting that a more relaxed threshold was conducive to better accuracy. Conversely, datasets like Hadoop and Spark demonstrated enhanced outcomes when the parsing threshold was increased, indicating that a more stringent threshold yielded superior results.

An identical pattern is replicated across the other measures. Generally, whenever our method achieves the highest attainable value in parsing accuracy, the same trend is also evident in F1, precision, and recall. Furthermore, in the case of the Thunderbird dataset, although

Table 4.6 Precisions of the methods using LogHub's framework. Top values in each column in bold.

	HDFS	Hadoop	Spark	Zookeeper	BGL	HPC	Thunderbird	Windows	Android	OpenStack
AEL	**1.000**	0.997	0.984	0.999	0.999	0.985	**1.000**	0.999	0.889	0.973
Drain	**1.000**	0.983	0.984	0.999	0.997	0.867	0.989	**1.000**	0.537	0.773
IPLoM	**1.000**	0.993	0.984	0.999	0.998	0.958	**1.000**	0.989	0.907	0.833
Lenma	**1.000**	**1.000**	0.984	**1.000**	**1.000**	0.984	**1.000**	0.989	0.992	0.993
LKE	**1.000**	0.766	0.128	0.999	**1.000**	0.097	0.965	**1.000**	**1.000**	0.880
LogMine	0.998	0.997	0.726	**1.000**	0.942	**1.000**	0.999	**1.000**	0.813	**1.000**
MoLFI	**1.000**	**1.000**	**1.000**	**1.000**	0.999	0.968	**1.000**	0.998	0.969	0.823
SHISO	**1.000**	0.997	0.985	0.996	0.996	0.372	0.968	**1.000**	0.743	0.991
Spell	**1.000**	**1.000**	0.984	0.999	0.999	0.977	0.990	**1.000**	0.999	**1.000**
Our Method	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	0.982	**1.000**	0.999	0.572	0.847

Table 4.7 Recalls of the methods using LogHub's framework. Top values in each column in bold.

	HDFS	Hadoop	Spark	Zookeeper	BGL	HPC	Thunderbird	Windows	Android	OpenStack
AEL	**1.000**	0.994	0.998	0.991	**1.000**	**1.000**	0.998	**1.000**	0.998	**1.000**
Drain	0.980	0.801	**1.000**	**1.000**	0.184	0.603	0.999	0.819	0.998	0.614
IPLoM	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	0.998	**1.000**	0.995	**1.000**
Lenma	**1.000**	0.993	0.994	0.997	0.886	0.977	0.999	1.000	0.992	0.996
LKE	**1.000**	**1.000**	**1.000**	0.990	0.892	**1.000**	**1.000**	**1.000**	0.984	0.999
LogMine	**1.000**	0.994	**1.000**	0.971	1.000	0.953	0.999	**1.000**	0.991	0.988
MoLFI	**1.000**	0.804	0.332	0.911	**1.000**	0.996	0.998	0.840	0.985	0.651
SHISO	**1.000**	0.999	0.998	0.991	0.993	0.997	0.860	0.840	0.975	0.996
Spell	**1.000**	0.852	0.998	**1.000**	0.918	0.996	0.999	**1.000**	0.985	0.988
Our Method	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**0.999**	0.996

our method doesn't exhibit the highest parsing accuracy, it does showcase the highest values in terms of F1, precision, and recall. Notably, in the recall table, our method garnered the highest values in 9 out of the 10 datasets analyzed. It's also important to highlight that certain datasets, such as OpenStack, yielded lower results for all methods, suggesting that this dataset may contain sentences more intricate to parse than the others.

Moreover, in some of the datasets, such as Android, OpenStack and Windows, our results were not the highest among the methods. That is because our method is based on the variability of tokens within the same sentences. For example, if we have a sentence *giving up after 4 tries* and another sentence *giving up after 2 tries*, we have four similar tokens and one token that varies between the sentences, thus being classified as a variable. In the datasets mentioned above, we observe many variables that are classified as such by LogHub, even though they do not appear differently in distinct sentences. This poses a challenge to the accuracy of our method. We also cannot underestimate the importance of the flaws in the LogHub ground truth, which are pointed out in Section 4.4.2 of our paper.

This variability underscores the importance of dataset-specific tuning to achieve optimal

parsing performance, enabling users to adapt the parsing threshold to the characteristics of each individual dataset. By accommodating such variations, our method can effectively address the diverse requirements and intricacies present in log datasets, ensuring accurate and reliable parsing outcomes.

In the analysis of our methodology, we posit that a favorable performance with a lower parsing threshold indicates a greater variation in the tokens representing variables. This suggests that the variables exhibit significant changes and variations across different log lines. Conversely, a satisfactory performance with a higher parsing threshold suggests a dataset where variable tokens occur more frequently and exhibit repetitive patterns across log lines. While prior knowledge of the log lines is not a prerequisite, conducting a textual analysis can prove beneficial in determining the optimal parsing threshold for each new dataset. By examining the characteristics of the log lines and identifying the underlying patterns, one can discern the appropriate parsing threshold that strikes a balance between capturing the diverse variations of variables and identifying recurrent patterns within the dataset.

4.4.2 Flaws in LogHub ground truths parsing

During our utilization of LogHub's parsing accuracy metrics, we encountered certain concerns regarding the validity of LogHub's ground truths. Notably, these concerns have been previously highlighted by other studies, including the works of [144] and [145]. These works shed light on potential limitations and challenges associated with the establishment of accurate ground truths in log parsing tasks. This awareness prompts us to adopt a critical approach and exercise caution when interpreting the parsing accuracy metrics provided by LogHub, encouraging further investigation and refinement of the evaluation process to enhance the overall reliability of the results. We can see some points to be discussed in Table 4.8.

In the analysis of the Spark dataset's first line, we observe a condensation of information after the Spark call ($spark : //$) into a single wildcard token ($< * >$). This condensation raises concerns about potential content loss, as it combines the possible process name, $CoarseGrainedScheduler$, with the IP and port accessed by that process, 10.10.34.11 : 48069. In our perspective, a more suitable parsing approach would involve separating these two pieces of information to retain their individual significance.

Moving on to the second line, from the OpenStack dataset, we note the omission of the allocated size metric (MB and GB), which could serve as important indicators for subsequent analysis. Lastly, the third line from the BGL dataset fails to parse the action name, $treeaddr$, and merges $0xffffffff$ and $E19$ into a single token, despite the presence of a blank space between them. These instances of inadequate parsing serve as a poignant reminder of the

Table 4.8 Examples of ground truth lines from LogHub

Original Log Line	Ground Truth Log Line
Connecting to driver: spark://CoarseGrainedScheduler @10.10.34.11:48069	Connecting to driver: spark://< ∗ >
[instance: bf8c824d-f099-4433-a41e-e3da7578262e] Attempting claim: memory 2048 MB, disk 20 GB, vcpus 1 CPU E1	[instance: < ∗ >] Attempting claim: memory < ∗ > MB disk < ∗ > GB, vcpus < ∗ > CPU
ciod: cpu 0 at treeaddr 438 sent unrecognized message 0xffffffff E19	ciod: cpu < ∗ > at treeaddr < ∗ > sent unrecognized message < ∗ >

importance of not solely relying on LogHub accuracy metrics as the sole factor of analysis. While these metrics provide valuable insights, they should be complemented by additional evaluation methods to ensure a comprehensive understanding of the parsing performance.

4.4.3 Flaws in LogHub ground truths grouping

Furthermore, we have identified certain ground truths that, in our assessment, could be combined into a single template. Several instances of such possibilities are illustrated in Table 4.9.

Upon examining the first two rows, it becomes evident that they could be conveniently grouped into a single template if the boolean values, such as "true" and "false," were also included in the parsing process. Similarly, in the case of the HDL dataset, an efficient tokenization approach would result in the consolidation of the two lines, as the sole distinction between them lies in the presence of a hyphen. Lastly, the last word of the three last sentences, if parsed, could potentially enable the merging of those sentences into a unified template.

4.5 Practical Application - Lessons Learned

Following the development of our log parser, we proceeded with its practical implementation in collaboration with our industry partner. Given the presence of numerous parallel computer systems within our partner's infrastructure, each generating substantial volumes of log files, the availability of a log parser capable of efficiently classifying the vast number of log lines proves to be an invaluable asset.

Table 4.9 Examples of ground truth lines from LogHub that could be joined

Dataset	Ground Truth Log Lines
Android	"animateCollapsePanels:flags=< * > force=false, delayed=false, mExpandedVisible=false"
Android	"animateCollapsePanels:flags=< * > force=false, delayed=false, mExpandedVisible=true"
HDL	not responding
HDL	not-responding
Thunderbird	registered new driver hub
Thunderbird	registered new driver usbfs
Thunderbird	registered new driver usbhid

We initiated our evaluation process by collecting small datasets from our industry partner, allowing us to assess the performance of our log parser on these specific files. Encouraged by the positive feedback received, we proceeded to expand our evaluation to larger datasets. It is crucial to emphasize that log parsing lacks a definitive ground truth, as the interpretation and parsing of log lines often rely on subjective analysis. Consequently, different parsing outcomes may be utilized in various ways, depending on specific requirements and preferences.

We leveraged the ongoing feedback from our partner's analysts to identify potential errors and make necessary adaptations to our dataset. One significant finding was that the parser's performance could be enhanced by incorporating regular expressions into the tokenization process. As we had anticipated the inclusion of an optional phase for regular expressions in our parser, this feature proved instrumental in improving parsing results. This approach allowed us to maintain dataset agnosticism while benefiting from the valuable domain knowledge of key users.

However, in our pursuit of achieving satisfactory results for our partner, we encountered the need to experiment with various parsing thresholds. During our initial analyses, our partner expressed their perception that our method was overly aggressive, converting more tokens into variables than necessary. As mentioned earlier, a lower parsing threshold corresponds to a higher degree of token variability within the dataset. On the other hand, a higher threshold may be suitable for datasets characterized by frequent token repetition. Nevertheless, in real-life scenarios, log lines often exhibit diverse patterns and syntaxes, making a lower threshold a more appropriate choice. The flexibility to adapt the parsing threshold allows us to strike a balance that aligns with the specific requirements and characteristics of each dataset.

During our practical tests, we made an important observation regarding the behavior of

our parser. We noticed that variables occurring in lines that are not highly frequent and belong to larger clusters are not accurately parsed. If these lines were treated as separate clusters, our method would likely be able to parse the variables correctly. However, when these lines are incorporated into larger clusters, the cumulative frequency of their appearances may fall below the parsing threshold, resulting in all tokens within those lines being parsed as variables. To address this challenge, we have considered potential alternatives. One approach involves exploring different clustering methods, as the current HDBSCAN algorithm automatically determines the number of clusters based on the data. Another option is to employ techniques that divide the overall dataset into smaller subsamples before clustering. This would create distinct clusters for each subsample, potentially improving the accuracy of the log parsing.

Furthermore, our partner expressed the need for parsing numerical characters, even if they exceeded the parsing threshold we had set. This was primarily because these numbers appeared repeatedly across log lines, so our method would indicate their category as static fields. To tackle this issue, we could incorporate a new heuristic into our parsing methodology, which would incorporate the parsing of numbers alongside the frequency dictionary and English vocabulary.

At present, our industry partner has implemented our log parser to extract templates and variables from error lines within software build logs. The templates extracted from the logs are then utilized as main input for error deduplication and error grouping processes. Looking ahead, our partner envisions expanding the utilization of the log parser to encompass anomaly detection and error identification across diverse log types.

4.6 Related Work

As far as we know, our method is the first to utilize Transformers and word analysis in an unsupervised manner for log file parsing. However, there exist several other approaches that employ log parsing or Transformers to accomplish the same objective, albeit with different methodologies. To support our research, we conducted an extensive literature review and identified two primary categories of related work for our method:

4.6.1 Log parsing

In the domain of log parsing, various strategies can be employed to parse log lines. In addition to the previously mentioned articles, in [31], the authors utilize a log abstraction technique that identifies the internal structure of each log line, enabling summarization and

categorization of log lines. In [146], data clustering and line pattern mining techniques are implemented for textual event logs. In [147], a log pattern extraction algorithm based on a weighted word matching rate is employed, utilizing longest common subsequences (LCS) to generate templates for new log lines in real-time. In [148], a log parser is developed that assigns one-to-one matching between rules and token types and parts of speech during the matching process. In [149], log messages are transformed into vectors, and the similarity between two log messages is measured based on the distance between their corresponding vectors. Clustering is then applied to the remaining log lines, and log templates are extracted from the resulting clusters. Lastly, in [150], an online log parsing method based on an evolving tree structure is proposed.

4.6.2 Transformers for log analysis

Regarding the utilization of transformers, they are primarily employed for log anomaly detection, as demonstrated in the works of [151, 152, 153, 154]. In the domain of log parsing, the availability of papers is relatively limited. Among these, a study conducted by [120] introduces a self-supervised learning model that tackles the parsing task through masked language modeling (MLM), enabling the extraction of summarizations from log data. Another significant contribution comes from [155], where a similar methodology to ours is employed. However, this approach does not take into account the presence of English words and does not offer the provision for incorporating regular expressions for pratical applications.

4.7 Conclusion

Our work presents a novel approach to log parsing, adopting the power of Transformers and incorporating a user-friendly word analysis mechanism, using our parsing threshold, so as to fine-tune the method to diverse datasets. Through our rigorous experimentation, we achieved the highest parsing accuracy among LogHub's calculations in four distinct datasets. Furthermore, the practical implementation of our parser within our industry partner's environment demonstrated its viability and provided valuable insights for potential enhancements in future iterations. Our findings underscore the potential of our method to address the log parsing challenges faced by real-world users and pave the way for further advancements in the field.

In terms of future work, firstly, we can further enhance our log parser by incorporating the valuable feedback and improvements suggested by our industry partner. Additionally, it would be beneficial to evaluate the performance of our parser across a diverse range of real-life datasets, encompassing various log syntaxes.

CHAPTER 5 ARTICLE 2: CLUSTERING TEXTUAL FEATURES FOR LOG SUMMARIZATION IN LARGE SOFTWARE SYSTEMS

Authors

Vithor Bertalan and Daniel Aloise. Submitted to Software: Practice and Experience on 2025-12-12.

Abstract

Identifying which lines deserve attention within large software log files can be a challenging task. Log files have consistently increased, reflecting the growth of software development platforms that are becoming larger and more integrated. However, software engineers rarely have the time to thoroughly analyze these files to identify important information. To address this issue, data mining methods have been proposed with the intent of summarizing log lines within large log datasets. In this work, we propose a supervised log summarization method based on clustering, which groups log data by using integrated information from (i) log line embeddings, (ii) identified variables extracted from parsed log lines, and (iii) the proximity between log lines. From the obtained clusters, we apply methods for topic modeling and word analysis to summarize and indicate which lines deserve more attention in a log file. In particular, we rely on a topic modeling framework that builds on BERT-based transformer embeddings, thereby integrating large language models (LLMs) into the summarization pipeline. Our quantitative analysis on various log datasets demonstrates that our approach outperforms state-of-the-art text summarization methods, thereby showing that the clustering method and the combination of (i)-(iii) are crucial in achieving high accuracy scores for diverse log structures. Finally, we outline the implementation of our method with our corporate partner, highlighting the feedback received and the adjustments made to enhance its practical use.

Keywords

Log Summarization, Machine Learning, Software Mining

5.1 Introduction

Software logs play an essential role in ensuring the reliability and stability of contemporary computer systems. As shown in [156], logs function as a comprehensive record of events

within a system, offering developers and administrators crucial insights into program functionality and the ability to pinpoint potential issues. Moreover, analyzing log data enables the extraction of information ranging from user behavior to resource utilization, rendering logs indispensable for troubleshooting and optimization purposes. A fundamental advantage of logging lies in its capacity to furnish detailed information about events that may occur during program execution, such as date-time stamps and compilation exceptions [157]. According to [158], approximately 80% of the total hours spent in a software system development life cycle are dedicated to identifying and correcting bugs. Therefore, on a daily basis, by honing in on specific log entries, developers can swiftly isolate the events surrounding an issue, saving precious time in the diagnostic phase.

With the intent of helping with the task of finding relevant log lines, one of the tools that we can use is topic modeling. In modern software development, where projects involve numerous contributors and evolving code bases, topic modeling techniques might help unveil latent thematic structures within textual data [159]. This capability proves instrumental in tasks such as code comprehension, bug tracking, and documentation management [160, 161, 162].

Several topic modeling methods adopt clustering approaches, such as [163], [164], and [165]. Clustering is fundamental for topic modeling, because it helps to identify and group words with similar meanings, allowing for the discovery of coherent and distinct topics within large text datasets. Through clustering, words that appear together in similar contexts are grouped around a common topic. Additionally, rather than using the entire text as input for topic modeling, working with smaller clusters proves more effective for extracting relevant knowledge from a dataset. By applying topic modeling to smaller clusters, we achieve higher granularity. Thus, words with subtle differences in meaning are more likely to be grouped into distinct clusters.

Nevertheless, to capture the peculiarities of the dataset, clustering models are highly dependent on how similarities are perceived among the clustered objects. In this paper, we investigate the positive impact of using more informative similarity matrices in clustering models for topic modeling, specifically in the context of log summarization in software engineering projects.

From a practical standpoint, our research is motivated by two complementary gaps. First, most existing log summarization techniques either rely on handcrafted linguistic heuristics, frequent pattern mining, or generic extractive summarizers that treat logs as ordinary natural language. These approaches often ignore log-specific structure such as parsed variables and the temporal/positional relationship between lines. Second, modern deep learning summarization models require large amounts of task-specific training data and substantial

computational resources, which are rarely available for log data, and they typically provide limited interpretability for practitioners. This motivates a method that (i) explicitly exploits multiple structural aspects of logs; (ii) remains lightweight and comprehensive enough for industrial deployment; and (iii) can be systematically analyzed in terms of how its components contribute to summarization quality.

The main step of our methodology consists of constructing a unified dissimilarity matrix between log lines that combines information on: (i) Cosine distances obtained from TF-IDF embeddings, (ii) Jaccard distances between the variables of the parsed log lines, using the Drain parsing method [29], and (iii) the proximity between log lines inside log files. Once log lines dissimilarities are computed, clusters are found using a k-medoids clustering model. Then, from each of the obtained clusters, the main topic is extracted by means of BerTopic [61], a technique designed for topic extraction. BerTopic internally relies on BERT-based transformer embeddings [166], which are derived from large language models (LLMs), thus effectively incorporating deep neural language representations in the topic modeling stage. Lastly, for each cluster, we identify the log lines that share the most tokens with the main extracted topic. These log lines represent the most relevant instances related to the topic and are referred to as the *log summaries*.

We evaluate the performance of our method against practical baseline approaches, as well as against state-of-the-art log summarization techniques proposed in the literature. For these evaluations, we used the datasets provided by [167], which contain distinct datasets of log lines adapted from [168], but tailored for log summarization.

In contrast to prior work, our method is explicitly designed to answer how much can be gained by carefully modeling the structure of log data, through textual similarity, parsed variables, and positional closeness, before applying topic modeling, and to what extent this structured representation competes with or complements more generic summarization techniques. By framing these questions and answering them empirically on multiple benchmark datasets, we aim to provide new insights rather than simply reusing existing tools.

The main scientific contributions of our work are:

- We propose a unified dissimilarity formulation for log lines, in which a single matrix $M = \alpha M_{\text{text}} + \beta M_{\text{var}} + \gamma M_{\text{close}}$ jointly captures (i) textual similarity through TF-IDF embeddings, (ii) similarity in parsed variables via Jaccard distances, and (iii) positional closeness within log files. This formulation allows us to systematically study how different aspects of log structure contribute to summarization quality across heterogeneous datasets.

- We design a clustering–topic modeling pipeline tailored to log summarization: instead of applying topic modeling directly to the full logs, we first cluster log lines using k-medoids on M and then apply BerTopic *within each cluster* to obtain cluster-specific topics and representative log summaries. To the best of our knowledge, this is the first work that combines parsed variables, temporal proximity, and transformer-based topic modeling in a unified framework for log summarization. Since BerTopic builds on BERT-based LLM embeddings, our method benefits from large pre-trained neural language models while preserving interpretability in the clustering and summary selection stages.

- We conduct an extensive empirical study on six benchmark log datasets, comparing our method with (i) simpler topic-modeling baselines and (ii) widely used extractive summarization methods (LDA, LexRank, LSA, Luhn, TextRank). Our results show that the proposed method achieves the highest ROUGE F1 scores on four out of six datasets and that different combinations of (α, β, γ) align with distinct lexical and structural properties of the logs. This analysis explicitly quantifies the impact of individual dissimilarity components.

- We report insights from an industrial deployment of our method in a large-scale software development environment. In particular, we show how the closeness component M_{close} and the tuning of model parameters (number of clusters, weights of M_{text}, M_{var}, and M_{close}) emerged from real-world constraints on performance and developer feedback, providing practical guidelines for configuring log summarization pipelines.

To guide our investigation, we formulate the following research questions:

- **RQ1.** Does combining textual, variable, and positional information in a unified dissimilarity matrix improve log summarization quality over simpler representations that rely on a single source of information?

- **RQ2.** How do different components of the unified matrix M and the main hyperparameters of the pipeline (e.g., the number of clusters and the number of topic tokens) affect summarization performance across heterogeneous log datasets?

The rest of the paper is organized as follows. Section 2 provides related work on the present research. Section 3 describes our method. Section 4 presents our computational results, comparing our method to baselines and state-of-the-art log summarization techniques proposed in the literature. Section 5 presents the report on the practical application of our

method with our corporate partner. Section 6 draws the conclusions and presents future work possibilities.

5.2 Related Work

Log summarization, as defined by [167], is the act of extracting the most important tokens inside log lines, subsequently ranking them based on a predefined metric. These selected tokens aim to represent the original lines, keeping their original meaning while reducing their representation size.

Different strategies have been used in the literature for log summarization. The method proposed in [169] parses raw log files, compiles the system runtime execution paths, eliminates duplicate events and uses n-grams to identify the most common log sequences, having the n-grams of those sequences summarize the initial log lines. The method reduces the amount of logs to be analyzed by 99%, effectively reducing the time spent for this task. In [170], a linguistic strategy based on fuzzy sets is adopted. The authors are able to reduce the amount of log lines that have to be analyzed by at least 80%.

Beyond reducing the raw volume of log data, other works focus on understanding which information makes a log line relevant and how to prioritize it for developers. In [171], the authors analyze log lines from Hadoop, Cassandra, and Zookeeper datasets, and identify that the five main categories of knowledge required to classify a line as relevant are *meaning*, *cause*, *context*, *impact* and *solution*. In [172], the authors query SAS log lines to find key tokens, and classify their severity based on the combination of those tokens inside the lines.

Moreover, the very intent of log summarization may be quite diverse. As examples, not only do we find studies focusing on log summarization for identifying bugs on distributed systems, such as in [173], which focuses on tracking start and end events to create summaries of all actions; but also on summarizing logs to better understand the behaviour of a software system, using execution traces, such as in the work of [174].

Also, in the work of [175], log summaries are used as vector embeddings for log parsing. In [176], the purpose is to reduce the storage size occupied by the log lines, using time series representations. In their work, the authors classify all log lines using a tuple of actions, users, objects, time interval and covering indexes, building a taxonomy graph from those tuples and finding which paths provide an information gain.

Machine Learning is also a viable tool to summarize log lines, as we can see in the next examples. In [177], the authors use frequent itemsets and Apriori to track the most common terms from web access logs so as to identify the users' browsing patterns. In [178], LSTM is the

technique adopted to summarize industrial log files, creating layers of hierarchical networks of feature representation. Finally, in [179], the research evaluates how well ChatGPT can be used to summarize log files. The authors show that while the tool shows good performance on other fields, such as anomaly detection and log parsing, ChatGPT still does not offer good results on the summarization of raw log lines.

Beyond log-specific pipelines, in the field of software engineering, researchers have increasingly turned to topic modeling techniques to extract valuable insights from the vast amount of textual data generated throughout the software development lifecycle. As we can see in the work of [180], software engineering is one of the most popular domains in which topic modeling has been adopted. The foundational work of [181] explored the application of topic modeling in code repositories, demonstrating its efficacy in identifying prevalent themes and common coding patterns. Other studies by [161] delved into the use of topic modeling for bug tracking, revealing how it is possible to use the method to detect duplicates in bug reports.

Furthermore, the convergence of natural language processing and software engineering has garnered significant attention. The work in [182] presented topic modeling techniques tailored for extracting semantic topics from software source code, aiding in knowledge transfer and project comprehension. Relevant contributions by [183] highlight advancements in using topic modeling to track the evolution in development topics across software engineering. We can also list applications such as the work of [184], using topics to find specific features inside raw source code, and the work of [185], that adopts LDA to identify which manual test cases are prioritized for testing. Topics are also used by [186] to identify the quality of features inside version control systems, and by [187] so as to get an architectural view of the artifacts and components of a corporate system.

In this emergent scenario of natural language processing, the integration of transformers for topic modeling in software engineering is becoming a very useful field of expertise, being put in place by the introduction of BerTopic [61]. In their model, the author creates a workflow in which the raw content is processed using Transformer language models, followed by clustering of the content and generation of topic representations utilizing the TF-IDF token measure. Crucially, BerTopic is typically instantiated with BERT or related transformer encoders [166], which are large language models pre-trained on massive text corpora. As a result, topic representations in BerTopic inherit rich semantics from LLM embeddings, even though the clustering and cTF-IDF steps remain relatively lightweight and comprehensive. BerTopic has been used in several areas of software engineering, such as requirements analysis [188], API documentation [189], bug tracking [190] and recommendation of software packages [191]. However, we are unaware of works using BerTopic for log summarization. Considering the

importance of comprehension and readability in log files, leveraging cutting-edge technologies like Transformers can significantly alleviate the manual workload for analysts engaged in this task.

Against this background, recent work has started to explore both classical and neural approaches to log summarization. For instance, [192] introduces a triplet-based representation of logs and an unsupervised summarization framework that highlights important messages by extracting and scoring (subject, predicate, object) triplets, showing consistent ROUGE improvements over prior tools. In parallel, a growing ecosystem of LLM-based systems, such as [193], which uses few-shot learning and reinforcement learning from human feedback to train a summarizer on human-written log summaries, treat log summarization primarily as a sequence-to-sequence generation problem. These efforts demonstrate that both structure-aware representations and neural models can substantially improve the readability and utility of raw logs, but they typically operate either at the level of individual log lines or small batches and do not explicitly model relationships between log entries in terms of content, variables, and positional closeness within a unified framework.

In the security domain, several systems leverage proprietary LLM APIs to assist analysts. For instance, [194] fine-tunes GPT-3.5-based models to build a conversational agent capable of summarizing uploaded logs, detecting events, and providing operational guidance for cybersecurity tasks, while a recent platform integrates Gemini 1.5 Flash into a web application to deliver low-latency summaries of heterogeneous security logs, highlighting anomalies such as brute-force attacks or suspicious IP activity [195]. Beyond summarization systems, [196] proposes an LLM-based, reference-free evaluation metric that scores log summaries along relevance and coherence dimensions without gold references, while other study provides a comprehensive benchmark to compare LLMs across log parsing, anomaly detection, fault diagnosis, and summarization tasks [197]. Together, these works show that large language models can effectively support log analysis and that the community is beginning to standardize evaluation; however, they generally treat logs as generic text, rely on proprietary LLM endpoints or fine-tuning pipelines, and do not systematically investigate how different similarity signals over logs interact when used as inputs to clustering-based summarization.

In summary, existing work on log summarization either focuses on linguistic heuristics, frequent patterns, storage reduction, or generic extractive text summarization, while topic modeling in software engineering has mostly targeted artifacts such as source code, bug reports, and requirements. None of these approaches jointly (i) model textual content, parsed variables, and positional closeness in a unified dissimilarity matrix, (ii) use this representation to drive k-medoids clustering of log lines, and (iii) apply transformer-based topic modeling

locally within clusters to select representative log summaries.

Building upon these observations, this combination, together with a systematic analysis of how the individual components of the matrix M affect ROUGE scores across six benchmark datasets, constitutes the core scientific contribution of our work beyond merely stacking existing methods. Our method, by contrast, builds on lightweight components (TF-IDF, Jaccard, k-medoids) combined with a transformer-based topic model (BerTopic) and is designed to be easily deployed in industrial environments where interpretability and resource constraints are important. The use of BERT-based LLM embeddings inside BerTopic ensures that our approach still leverages modern deep language representations.

5.3 Methodology

In this section, we describe our proposed methodology for log summarization, outlining the steps from raw log processing to the generation of results. Our code is publicly available for external evaluation at `https://doi.org/10.5281/zenodo.13921642`. In accordance with the journal's Artificial Intelligence Generated Content (AIGC) policy, we used AIGC tools to perform grammatical correction and improve the readability of selected parts of the manuscript.

Figure 5.1 provides an overview of the complete pipeline, from log collection to the evaluation of the resulting summaries. This work extends our previous preprint on log summarization [198].

5.3.1 Collecting Data

In the first step of our workflow, we collect raw log data from the origin systems. One of our objectives is to avoid any manual influence during this process. Therefore, in this step, we refrain from any type of preprocessing. This approach ensures that the key users of the workflow do not need to possess any previous knowledge of the log structures, making the method flexible enough to handle diverse log files. Our workflow, using Python libraries, can handle various log types, including text files, CSV spreadsheets, JSON files, and SQL databases.

5.3.2 Computing dissimilarity between log lines

Once raw log data is collected, we proceed by computing dissimilarity between the log lines. This is a key step of our methodology as these dissimilarities are later used by clustering methods to identify patterns and group similar log entries.

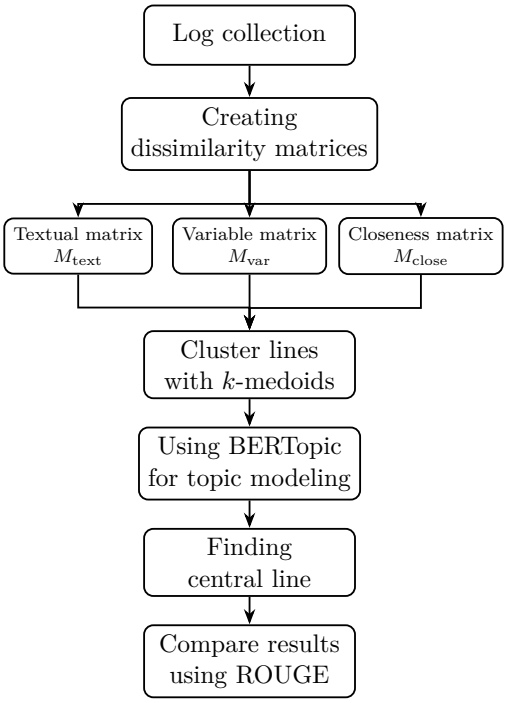

Figure 5.1 Sequential workflow of the proposed log summarization method.

The dissimilarity matrix M computed in our methodology is composed of the three main components that enrich the information contained in M. We illustrate our approach with the logs exhibited in Table 5.1.

Table 5.1 Example of log lines

Line Number	**Log Line**
1	make compile /opt/shell.bat
2	make compile /usr/main/code.appimage
3	buffer overflow /opt/shell.bat
4	buffer overflow /usr/main/code.appimage
5	cleaning memory cache

We can observe from the table five sequential hypothetical lines inside a log file. In our methodology, the dissimilarity matrix M between these lines is composed from three other matrices, each one representing a distinct aspect concerning the compared log lines.

The first one, denoted M_{text}, stores at M_{text}^{ij}, the *textual dissimilarity* between the tokens shared by log lines i and j. In the illustrated log file, lines 1 and 2, and lines 3 and 4, are similar, since they share two tokens. Here, our goal was to capture the semantic difference between the lines, clearly signaling the textual proximity between the log lines. Thus,

lines that are identical will show no dissimilarity (i.e., a value of 0), whereas lines that are completely different will show the highest possible dissimilarity value.

However, this is not the only way of computing dissimilarities between log lines. We notice that lines 1 and 3, and lines 2 and 4, are related to the same process (indicated by the last token in each line). Even though their text is somehow different, their variables are the same, showing that they are connected to the same subjects. Here, our reasoning is that shared variables may carry a significant meaning in bringing lines together. That is, even if two lines show many different tokens, if they share the same parsed variable, they should carry a logical connection. This is captured by the dissimilarity matrix M_{var}.

Finally, we can observe in Table 5.1 that line 5 could be a direct response to the errors found in lines 3 and 4. Since lines 3 and 4 showed buffer overflows, a logical reaction of the operating system would be cleaning the memory cache, so as to allow other processes and threads to take place. This means that those 3 lines are similar, since they have a logical connection of action/reaction. However, their text is completely different, and they do not show any shared variables. As log lines in log files can be considered as a form of time series, lines that are placed together are more likely to be connected than lines that are positioned far apart in a log file. This type of information is represented in the dissimilarity matrix M_{close}.

We detail next how each one these matrices are computed from the collected log files data.

Computing the textual matrix (M_{text})

From the collected data, we employ parsing algorithms to distinguish static tokens from variable tokens. Static tokens are those which carry the action being described in the lines. As an analogy, they could be classified as the subjects of a natural language. Variable tokens are the parts that represent the targets that are affected by those actions, being compared to the objects of standard languages.

As mentioned in [29], log parsing is the process of converting unstructured log lines into structured events. This involves separating static tokens—those that are part of a system operation template—from variables, such as timestamps and verbosity levels. In our work, we used the DRAIN log parsing method developed in [29].

Table 5.2 presents an example of parsed lines. In the first line, the tokens `visible` and `is` are classified as static, whereas the sequence `system.charge.show` is classified as a variable, and replaced by a wildcard (in our example, a `<*>`. If a line contains multiple variable tokens, they are all replaced by the wildcard, as in the third line of Table 5.2.

It is essential to note that, in our research, we do not discard variables; instead, we store

Table 5.2 Example of parsed lines

Original Log Line	Parsed Log Line Template
visible is system.charge.show	visible is $< * >$
visible is system.exit	visible is $< * >$
now starting system.charge.show gt0	now starting $< * > < * >$

them in a separate table for use when computing M_{var} (see Section 5.3.2).

After parsing, all log lines are converted to vector embeddings using TF-IDF [199], thus measuring the impact of a word inside a corpus, while considering its frequency. Therefore, if a token appears in all sentences, its TF-IDF value will approach zero. Otherwise, it will tend to 1. The matrix M_{text} is then obtained by computing the cosine distance between each pair of such vectors. We remark that M_{text} serves as a comprehensive representation of the pairwise dissimilarities among the lines in our dataset, mathematically indicating their textual similarity. In this scheme, a null cosine distance signifies complete similarity between two lines, i.e., they present exactly the same tokens, representing textual equality. Indeed, software log lines that discuss similar code aspects, show similar commands or instructions, share related design patterns, or cover comparable subject matters are likely to have lower cosine distances, thus exhibiting similar themes.

Computing the variable matrix (M_{var})

The log parsing procedure explained in Section 5.3.2 also entails a boolean matrix F for the presence of variable tokens across the entire set of log lines. Within this matrix, each cell stores a value of 1 if the token is present in the corresponding log line, and 0 otherwise. An illustrative example of such matrix is presented in Table 5.3.

Table 5.3 Example of boolean matrix F for the parsed variable tokens

Log Line	system.charge.show	system.exit	gt0
visible is system.charge.show	1	0	0
visible is system.exit	0	1	0
now starting system.charge.show gt0	1	0	1

The variable matrix M_{var} is obtained by computing the Jaccard dissimilarity between the lines of F. In this process, lines sharing precisely the same variables are assigned a value of 0, while lines with entirely different variables are assigned a value of 1. It is important to highlight that our intention is not to evaluate the variables textually, but to assert their presence or absence. This primary motivation for this representation of differences is to

maintain the relationships between lines that, although they possess distinct token structures, are associated with the same variables.

Computing the closeness matrix (M_{close})

Matrix M_{close} gathers information about the actual "distance" between two log lines inside the log file they belong to. Thus, log lines that are closely arranged in the log file will be less dissimilar than log lines that are far apart in the log file.

Table 5.4 provides an illustrative example of a closeness matrix M_{close} for a log file containing 5 log lines. The values in M_{close} are normalized by the total number of lines in the log file so that all values lie between 0 and 1. Thus, each line exhibits a dissimilarity of 0 with itself and a small dissimilarity with its immediate neighbors. As lines become more distant from each other, the dissimilarity values gradually increase.

Table 5.4 Example of a closeness matrix

	Line 0	**Line 1**	**Line 2**	**Line 3**	**Line 4**
Line 0	0	0.25	0.5	0.75	1
Line 1	0.25	0	0.25	0.5	0.75
Line 2	0.5	0.25	0	0.25	0.5
Line 3	0.75	0.5	0.25	0	0.25
Line 4	1	0.75	0.5	0.25	0

The fundamental concept behind M_{close} is to assign lower values to lines that are in close proximity, recognizing the significance of their sequencing in the log files. In the context of log analysis, the logical flow of events is crucial, and this closeness matrix aims to accentuate the importance of preserving the sequential arrangement of lines for a more meaningful interpretation.

Computing the dissimilarity matrix M

The final dissimilarity matrix M, which is used by the clustering procedure of our method, combines the three computed matrices M_{text}, M_{var} and M_{close} by making:

$$M = \alpha M_{text} + \beta M_{var} + \gamma M_{close}, \quad \text{such that } \alpha + \beta + \gamma = 1. \tag{5.1}$$

Thus, depending on the characteristics of the log files, the user may adjust the weights α, β, and γ in (5.1), ensuring they conform to the constraint of a convex combination, in order to emphasize one or more specific dissimilarity aspects when computing the final matrix M. As

an example, some log files contain datetime stamps, while others do not. We describe more characteristics of the textual structure of each log dataset in Section 5.4.2.

5.3.3 Clustering log lines

Once we process the data in the unified matrix M, we proceed to the clustering step where log lines are grouped according to their dissimilarity. Clustering is a powerful tool for automated analysis of data [200]. It addresses the following general problem: given a set of entities, find subsets, denoted clusters, which are homogeneous and/or well separated. Homogeneity means that entities in the same cluster must be similar and separation means that entities in different clusters must differ one from another. Hence, homogeneity and separation are properties that depend on how similarity (equiv. dissimilarity) is perceived among the observed entities.

In our work, the clustered entities correspond to the analyzed log lines. By representing $L = \{\ell_1, \ldots, \ell_n\}$ as the set of log lines, our clustering method utilizes the $n \times n$ dissimilarity matrix M, where each entry M_{ij} captures the dissimilarity between pairs of log lines (ℓ_i, ℓ_j) in L.

We partition the log lines using the k-medoids clustering model [201], which aims to partition the log lines into clusters such that the sum of the dissimilarities from each log line to the central exemplar of its cluster (i.e., its medoid) is minimal. Thus, the k-medoids model aligns well with our primary objective of identifying the most relevant log line in a cluster. Moreover, The k-medoids model is versatile, capable of clustering metric data as well as more abstract data with notions of similarity and dissimilarity. This versatility accounts for the wide range of applications of the model [202]. The k-medoids model solved in this work can be expressed as:

$$\min \quad \sum_{i=1}^{n}\sum_{j=1}^{n} M_{ij}x_{ij} \tag{5.2a}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} x_{ij} = 1 \qquad \forall i \in \{1, \ldots, n\}, \tag{5.2b}$$

$$x_{ij} \leq x_{jj} \qquad \forall i, j \in \{1, \ldots, n\}, \tag{5.2c}$$

$$\sum_{j=1}^{n} x_{jj} = k, \tag{5.2d}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in \{1, \ldots, n\}. \tag{5.2e}$$

The hyperparameter k establishes the number of medoids, and by consequence, the number

of clusters to be found. The decision variables x_{ij}, for all $i, j \in \{1, \ldots, n\}$, correspond to the assignment of log line ℓ_i to the cluster for which its representative log line (i.e., its medoid) is log line ℓ_j. Constraints (5.2b) ensure that every log line must be assigned to one medoid. Constraints (5.2c) guarantee that a log line ℓ_i is assigned to line log ℓ_j only if the latter is a medoid. The constraint (5.2d) requires the selection of exactly k log lines as medoids. Finally, constraints (5.2e) specify the decision variables to be binary. We used here the PAM algorithm [203] to obtain heuristic solutions for the k-medoids model. Given that our method seeks to identify the most pertinent line within each cluster, it is noteworthy that the number of clusters is an important parameter of our method which is subject to fine-tuning. In our empirical study, we analyze the impact of this parameter by setting k to the number of unique templates produced by the Drain parser and by validating this choice against alternative values; the details of this analysis are reported in Section 5.4.2.

5.3.4 Bertopic modeling from clustered log lines

To proceed with the log summarization, our method executes BerTopic [61] for individual topic modeling on each of the clusters identified by the k-medoids algorithm. This approach deviates from running BerTopic across the entire log data, allowing for a more nuanced examination of distinct themes within each cluster. Since BerTopic is instantiated with BERT-based transformer encoders, this step injects LLM-level semantic information into our log clusters without requiring task-specific fine-tuning.

BerTopic works by sequential steps: first, the method starts by generating BERT textual embeddings [166]. BERT is a bidirectional transformer model pre-trained as a large language model on large-scale corpora, and its embeddings capture rich contextual semantics that are crucial for distinguishing subtle differences between log messages. Then, in order to reduce the size of the data set, the dimensionality reduction algorithm called UMAP is used [63]. Subsequently, the method creates clusters using the HDBSCAN algorithm [204]. These steps are skipped in our method since a clustering solution is already provided by our k-medoids algorithm.

From this moment on, our method follows the BerTopic workflow. It proceeds by using a bag-of-words representation f of the words in the original text (in our context, the log lines). Two BoW representations are then obtained: one for each token x with respect to the entire set of log lines, denoted (f_x); and another considering the frequency of the token inside each cluster, denoted ($f_{x,c}$), which is normalized to account for the varying number of log lines within each cluster. The BerTopic model proceeds by computing a customized TFIDF representation, called by the authors $cTFIDF$, to represent the weight of each token

inside its own cluster. The equation representing $cTFIDF$ is given by:

$$W_{x,c} = f_{x,c} \log\left(1 + \frac{a}{f_x}\right) \tag{5.3}$$

where:

- $W_{x,c}$ = the weight of token x in cluster c
- $f_{x,c}$ = the frequency of token x in cluster c
- a = the average number of tokens per cluster
- f_x = frequency of token x in all clusters

Based on Equation (5.3), BerTopic identifies one topic for each detected cluster. Each topic is composed of the top-N tokens with the highest weights W, where N is a hyperparameter set by the user. Thus, each topic represents a semantically coherent group of log lines.

The output of a BerTopic execution shows the id of the topic and the list of its associated tokens, which are the most representative for that topic. Table 5.5 illustrates an output from a BerTopic execution, extracted from [167], showing five topics, one for each identified cluster, along with their top-4 most important tokens. For instance, topic 1 indicates tokens related to a fatal error interruption, while topic 2 indicates a correction executed in the cache parity.

Table 5.5 Example of topic generated by BERTopic from the `BGL` file.

Topic	**Most Significant Tokens**
1	data, interrupt, fatal, error
2	cache, parity, corrected, error
3	tree, wire, unit, bit
4	vpd, check, card, node
5	stopping, panic, execution, fatal

5.3.5 Choosing the log summaries

After the topics and their most representative tokens have been identified, BerTopic compares these lists of terms to every log line within each cluster. The log line sharing the greatest number of tokens with the most representative tokens of the topic is chosen as the log summary of the log lines in that cluster. This approach ensures that the summary line

encapsulates the essence of the reference topic, thereby providing a concise yet informative representation of the cluster's thematic content.

This represents a novel approach in contrast to other log summarization methods, which typically rely on linguistic approaches for log lines summarization, such as the Luhn algorithm, that uses frequencies of words, and their cumulative weights, to summarize text [205].

By explicitly separating the roles of M_{text}, M_{var}, M_{close}, the k-medoids clustering, and the BerTopic topics, our framework also facilitates a component-wise interpretation of the pipeline. In particular, we can ask whether performance gains come primarily from better similarity modeling, from clustering, or from topic-based selection of representatives, and we address these questions empirically in Section 5.4. The use of BERT-based LLM embeddings inside BerTopic ensures that improvements cannot be attributed solely to classical techniques, but also to the semantic representations provided by large language models.

5.4 Empirical Evaluation

In this section, we assess the efficacy of our method against baseline approaches aiming to investigate the impact of our enriched unified matrix M presented in Section 5.3.2. Moreover, we evaluate our log summarization approach against state-of-the-art methods proposed in the literature. We use a supervised approach, in which we use 80% of the datasets to find the optimal hyperparameter combination, and the remaining 20% to assess our Rouge scores.

To make the evaluation protocol explicit, all experiments follow the same procedure. For each dataset, we randomly split the 2,000 log lines into a training/validation partition (80%) and a held-out test partition (20%). The training/validation partition is used solely to select hyperparameters, including the weights (α, β, γ) of the unified matrix M, the number of clusters k, and the number N of tokens per topic. Once the hyperparameters are fixed, we retrain the models on the full 80% partition and evaluate the final summarization performance on the remaining 20% test lines. All ROUGE scores reported in this section are computed on this held-out test partition, ensuring that the evaluation is not biased by the hyperparameter search.

5.4.1 Summarization Metrics - Rouge Score

In this work, we used the Rouge-N metric [206] to perform our empirical evaluation. Rouge-n is a coverage measure of n-grams between a created candidate summary (also called a *hypothesis*), and a set of *reference* summaries, which serves as the ground truth.

In our context, true positives correspond to tokens that are present in both the reference and the hypothesis, whereas false positives are tokens present only in the hypothesis. Finally, false negatives represent tokens that are present in the references, but not in the hypotheses. Thus, the ROUGE F1 score corresponds to the harmonic mean between the *Recall* and *Precision* measures computed as:

$$Recall = \frac{\text{Number of matching n-grams}}{\text{Number of n-grams in the reference}} \tag{5.4}$$

$$Precision = \frac{\text{Number of matching n-grams}}{\text{Number of n-grams in the hypothesis}} \tag{5.5}$$

Table 5.6 illustrates an example of ground-truth log line summary, and two candidate (hypothesis) summaries. The computation of Rouge-1 for the two candidate summaries is given in Table 5.7. In the illustrated example, $H1$ correctly matches 4 out of 4 tokens in the reference summary, for a $Recall = 1$. However, the 4 matches are obtained from a total of 8 tokens, yielding a $Precision = 0.5$. Applying a harmonic mean between the *Recall* and *Precision*, we obtain a Rouge-1 score of 0.67.

Table 5.6 Example of summary references and hypothesis

Reference Summary	make compile completed successfully
Hypothesis Summary 1	the software make and compile was completed successfully
Hypothesis Summary 2	run make compile completed

Table 5.7 Example of summary metrics using the texts from Table 5.6

Hypothesis	**Recall**	**Precision**	**Rouge-1**
H1	$4/4 = 1$	$4/8 = 0.5$	$\frac{2\times1\times0.5}{1+0.5} = 0.67$
H2	$3/4 = 0.75$	$3/4 = 0.75$	$\frac{2\times0.75\times0.75}{0.75+0.75} = 0.75$

In our experiments, we focus on ROUGE-1 F1 as the main metric, since log summaries are typically short and dominated by single tokens rather than longer n-grams. All texts are lowercased and tokenized on whitespace and punctuation before computing ROUGE, and the same preprocessing is applied to both reference and candidate summaries. When multiple reference summaries are available for a given log line, we compute ROUGE-1 F1 against each reference and then average the scores, which yields a more robust estimate of summarization quality in the presence of alternative valid summaries.

5.4.2 Evaluating the unified matrix M for log line clustering

To evaluate the effectiveness of our k-medoids clustering model when using the unified dissimilarity matrix M described in Section 5.3.2, we compare our log summarization method introduced in Section 5.3 against two alternative baseline approaches.

In the work of [167], six log datasets are provided with their summaries, from distinct sources: `BGL`, a collection of runtime logs from BlueGene/L supercomputer system at Lawrence Livermore National Labs (LLNL) in Livermore, California; `HDFS`, an acronym for Hadoop Distributed File System, a distributed processing software; `HPC`, a collection of runtime logs from high performance computing clusters at the Los Alamos National Laboratories; `Proxifier`, a standalone software built to manage network applications through SOCKS or HTTPS proxy and chains; `Spark`, containing data from Apache Spark, a unified analytics engine for big data processing; and `Zookeeper`, a software service for configuring group services and maintaining cloud computing information. Each dataset is composed of 2000 lines, separated into groups, along with the ground-truth log summary for each group.

Each log dataset has its own characteristics and log lexicon. We can see some examples of lines from the datasets, along with their log summaries in Table 5.8, extracted from [167]. With the exception of the first line, all other log lines have multiple summaries. Moreover, Table 5.9 presents the number of unique words, the average sentence length and the number of *hapax legomena* of each dataset. `HDFS`, for example, has a high average sentence length and a high number of *hapax legomena*, indicating a more complex textual structure. On the other hand, `Zookeeper` has fewer unique words and *hapax legomena*, and also a lower average sentence length.

Across our analysis of the log summaries, we remarked that the division of the log files provided by [167] consisted of an exact partition of 20 log lines per cluster. As each log file provided by [168] has 2000 lines per file, which accounts for 100 clusters per file. To help in this distribution, as shown in Table 5.8, some clusters have more than one line summary. In Table 5.8, we observe that all log lines but the first have two line summaries (separated by semicolons). That happens because the straight division of groups of 20 lines might include lines with different characteristics and textual structures.

We assessed our log summarization approach presented in Section 5.3 against two baselines described below:

Table 5.8 Example of ground truth summaries

Log Line	Dataset	Line Summaries
RAS KERNEL FATAL data storage interrupt	BGL	data storage interrupt
INFO dfs.DataBlockScanner: Verification succeeded for blk_-84236234690192	HDFS	Receiving block src; blockMap updated; Verification succeeded
node status 1075931221 0 configured out	HPC	node status configured out; node status not responding
proxy.cse.cuhk.edu.hk 5070 open through proxy proxy.cse.cuhk.edu.hk 5070 HTTPS	Proxifier	open through proxy; bytes sent; bytes received; close
INFO storage.MemoryStore Block broadcast_5_piece289 stored as bytes in memory estimated size 4.0 MB free 563.0 MB	Spark	Block stored in memory; estimated size
INFO Closed socket connection for client /10.10.34.11 : 52893 which had sessionid 0x14ed93111f2009b	Zookeeper	Closed socket connection; caught stream exception

Baseline A - Topic modeling from predefined clusters

In this baseline, each of the log datasets are first converted to numerical embeddings using BerTopic's standard MiniLM embedding language model [62]. MiniLM is a distilled transformer encoder derived from BERT, and thus also belongs to the family of large language models, although in a compressed form. Each cluster provided by [167] is then analyzed using BerTopic to determine its corresponding main topics. As a result, we obtain 20 topics, each representing the top topic from the cluster. In the sequel, log summaries are obtained for each cluster as detailed in Section 5.3.5.

Table 5.9 Comparison of lexicon of log datasets

Dataset	Unique Words	Avg Sentence Length	Hapaxes
BGL	1703	16.51	1072
HDFS	2308	16.11	1508
HPC	1440	9.39	930
Proxifier	1451	17.35	943
Spark	1861	12.04	1089
Zookeeper	1306	10.32	735

Baseline B - End-to-end BerTopic

In this baseline, BerTopic is applied directly over the raw log datasets as described in [61]. Once the topics are found, log summaries are computed as explained in Section 5.3.5. To ensure a fair comparison, the embedded log lines generated by BerTopic's standard MiniLM are clustered using the k-medoids algorithm with Euclidean distance as the dissimilarity metric.

The summaries generated by our method are compared to those produced by baselines A and B using the ground truth provided by [167]. Additionally, both our method and baseline B use the same number of clusters as input to their underlying k-medoids model to ensure that this parameter does not affect the partitioning of the log lines. This allows us to isolate the impact of the enriched dissimilarity matrix M used for clustering on the quality of the final log summaries. Following preliminary experiments, the number of clusters was set to the number of unique templates identified during Drain's parsing steps. The hyperparameters of Drain for each dataset were selected based on the values given in [29]. Finally, since the computational experiments presented in [29] do not include the dataset `Spark` in their experiments, we used the values from the dataset `BGL` for the tests with `Spark`, given their similar values of unique words and *hapax legomena* reported in Table 9.

As an ablation study and a test of the number of parameters, we explicitly analyze the impact of the main hyperparameters of our pipeline. For each dataset, we perform a grid search over (α, β, γ) with a step size of 0.1 under the convexity constraint $\alpha + \beta + \gamma = 1$, and we evaluate each combination using ROUGE-1 F1 on the 80% validation partition. In addition, we explored different values for the number of clusters k (e.g., 0.5, 1.0, and 1.5 times the number of Drain templates) and for the number of tokens N per topic (from 5 to 15). We observed that the unified matrix M and the choice of (α, β, γ) have the largest impact on the results, whereas moderate changes in k and N lead to smaller variations in ROUGE scores, provided that they remain within a reasonable range. Based on these observations, we fix k to the number of templates and $N = 10$ tokens for all datasets in the final experiments reported below.

The setting of the matrix weights α, β and γ was performed with a 20% slice of each dataset, running all possible combinations of the parameters from 0 to 1, with 0.1 increments.

The best combinations of α, β, and γ found for each dataset are shown in Table 5.10.

We observe in the table that very often the best combination involves the use of the three matrices M_{text}, M_{var} and M_{close}, except for datasets `HDFS` and `Proxifier` for which only M_{var} and M_{text} are sufficient, respectively. However, we observe that, depending on the

Table 5.10 Best combinations of α, β, and γ for the unified matrix M in each tested dataset

Dataset	α	β	γ
BGL	0.6	0.3	0.1
HDFS	0	1	0
HPC	0.2	0.1	0.7
Proxifier	1	0	0
Spark	0.2	0.8	0
Zookeeper	0.4	0.6	0

dataset, some matrices may contribute more to the summarization process than others. This finding validates our strategy of combining and unifying the three matrices, each representing a different aspect of the log text into a single unified matrix. In our results, we found that some datasets, such as BGL, yield better results when prioritizing the standard TFIDF conversion (i.e., matrix M_{text}). This is likely associated to the case where the log lines are highly distinct from one another, resulting in better summarization when the lexicon of the text is emphasized. Other datasets, such as Spark and HDFS, are better summarized by augmenting the importance of the variables within the lines (i.e., matrix M_{var}). In our view, this suggests a text structure where the lines do not vary much, but the variables provide enough diversity to represent significant meaning. This is a case where a simple TFIDF embedding conversion, without additional heuristics, is not expected to effectively summarize the lines. Furthermore, we observe that in some cases, the closeness matrix (i.e., matrix M_{close}) delivers the best summarization output, as seen with the HPC dataset, where similar lines appear to be consistently placed adjacent to each other, with changes in the lexicon occurring over time.

Table 5.10 therefore provides a component-wise analysis of the unified matrix M and directly addresses the individual contribution of (i) log line embeddings, (ii) variables extracted from parsed log lines, and (iii) the proximity between log lines. Datasets for which α dominates benefit primarily from textual information; datasets where β is large rely more heavily on the structure of variables; and datasets with higher γ clearly profit from exploiting the positional closeness of log lines. The fact that different datasets select different combinations (α, β, γ) indicates that each of these three sources of information can be critical, depending on the characteristics of the underlying software system. This analysis empirically confirms that it is not enough to rely only on log embeddings or only on variables or proximity: combining them in a controlled way is what yields robust summarization performance across diverse scenarios.

While the current version of our method relies on manual (grid-based) tuning of (α, β, γ),

this can be seen as a limitation in terms of full automation. However, the coarse step size of 0.1 and the observation that several neighboring combinations achieve comparable ROUGE scores suggest that the method is not overly sensitive to small perturbations of these weights. In future work, we plan to replace this grid search by a data-driven procedure that learns the weights directly from the training data, for example by optimizing validation ROUGE through a bi-level or gradient-based optimization scheme.

After determining the weights of the component matrices M_{text}, M_{var} and M_{close} of M, our method is evaluated against baselines A and B on the entire dataset using the ROUGE F1 score. These results are compiled in Table 5.11.

Table 5.11 F1 ROUGE scores of our method against baselines A and B

Dataset	A	B	Our Method
BGL	0.093	0.318	**0.433**
HDFS	0.106	0.157	**0.189**
HPC	0.284	0.427	**0.560**
Proxifier	**0.122**	0.096	0.075
Spark	0.059	0.120	**0.127**
Zookeeper	0.067	0.101	**0.153**

We notice from the table that our method outperforms the other two baselines in 5 of the 6 datasets. In particular, we observe that for the Proxifier dataset, our method is outperformed by baseline A that uses the predefined clusters provided in [167]. We attribute this to the specific characteristics of the Proxifier dataset. First, our parsing methods frequently failed to identify many variables in the line structures of this dataset. Besides, the log lines are often repeated, with little variation. Because of this characteristic, our variable matrix M_{var} tends to be very sparse, as not many lines contain unique variables, rendering the matrix ineffective. Additionally, the repetitiveness of the summaries presents another challenge. For example, the line "open through proxy" is presented by [167] as a possible summary for 1,979 out of the 2,000 log lines.

Overall, the comparison with baselines A and B demonstrates that, once the hyperparameters are selected as described above, the unified matrix M brings a consistent improvement in ROUGE-1 F1 over simpler clustering and summarization strategies, which directly answers RQ1 for five of the six datasets.

5.4.3 Evaluating the Log Summarizations Against the State-of-the-Art

To evaluate the summary results of our log summarization method, we compared the ROUGE F1 score of our method with those obtained by state-of-the-art summarization techniques: LexRank [207], TextRank [208], LSA [209], and Luhn [205]. LexRank works by representing text as a graph, where sentences are nodes and edges represent the strength of their semantic similarity. It tracks the centrality of nodes in the graph and selects the highest scores as potential summaries. TextRank constructs a graph of words or sentences and computes the importance of each unit based on its connectivity to other units in the graph, selecting the most important nodes as the summaries. LSA works by creating a mathematical model of word co-occurrences in corpora and applies singular value decomposition (SVD) to identify underlying latent semantic structures, selecting sentences that are closely related to these structures as the summaries. Luhn assigns a weight to each word in the text based on its frequency, scores each sentence based on the cumulative weight of those words, and selects the sentences with higher weights as the summaries. Additionally, to establish a comparison with the category of topic modeling techniques, we compared our results with a baseline LDA [60], which, like our method, generates one topic per cluster. All tests were conducted using Python implementations, and the summaries limited to a maximum of $N = 10$ tokens per topic.

With the intent of comparing computational costs, we also measured the execution time of each method. All benchmarks were run on an Intel Core i7-7820X CPU @ 3.60GHz, with 32GB of RAM allocated. All execution times presented in Table 12 were calculated during the training phase. This means that the execution times for our algorithms were measured after the hyperparameter selection step, as this step is not required in a production environment. As we can observe, the CPU time spent by our method is comparable to that spent by the other state-of-the-art methods found in the literature.

Table 5.12 Execution time for the algorithms

Method	**Running Time (in seconds)**
LDA	11.11
LexRank	2.43
LSA	10.01
Luhn	1.35
TextRank	2.57
Our Method	9.08

For all baselines, we used publicly available Python implementations with their default parameter settings, except for the summary length, which we fixed to $N = 10$ tokens to match

our method and ensure a fair comparison. This means that the only tuned parameters in our study are those of the proposed method (and of baselines A and B defined in Section 5.4.2), while the state-of-the-art summarizers are evaluated in a standard, off-the-shelf configuration. We acknowledge that we do not compare against end-to-end deep learning summarization models. This is left as future work due to the lack of publicly available log-specific summarization corpora and the substantial computational resources required. Our focus here is to understand how far a carefully designed, comprehensive, and lightweight pipeline can go when compared with strong classical extractive baselines. Importantly, the use of BerTopic means that our approach already incorporates LLM-based transformer embeddings, even though the final summaries are obtained through clustering and cTF-IDF rather than sequence-to-sequence generation.

Table 13 presents F1 scores collected from our experiments and computed from the ground truth provided in [167]. As shown in lines 2 to 6 of Table 8, some log lines in the ground truth may have multiple possible summaries. In such cases, all methods were compared against each of the possible summaries, and we averaged the F1 scores for the set of summaries as the final result.

Table 5.13 F1 Scores for the summarization methods

	LDA	**LexRank**	**LSA**	**Luhn**	**TextRank**	**Our Method**
BGL	0.404	0.326	0.314	0.333	0.335	**0.433**
HDFS	0.149	0.165	0.162	0.163	0.165	**0.189**
HPC	0.553	0.301	0.284	0.322	0.357	**0.560**
Proxifier	0.179	0.058	0.079	0.079	**0.203**	0.075
Spark	**0.140**	0.086	0.089	0.099	0.137	0.127
Zookeeper	0.148	0.121	0.123	0.130	0.139	**0.153**

The results in the table indicate that our method achieved the highest F1 score for 4 out of 6 datasets.

The results demonstrate that our method is an effective approach for log line summarization when compared to other state-of-the-art techniques. Despite fluctuations in F1 scores across datasets, ranging from 0.075 to 0.560, our results consistently ranked among the top ones for all six tested datasets.

In our view, the variability in the results reflects the diversity of the log files. Although some characteristics, such as timestamps or file paths, might be consistent, each software system decides independently when and how to log the most significant events during execution. It is also important to note that log lines are not written in natural language. Each log template has a distinct lexical structure, making the application of natural language processing meth-

ods potentially challenging. This specialized terminology used in log files can potentially hinder the effectiveness of traditional summarization techniques.

Moreover, we observed that our method performed better on datasets with fewer multiple summaries per group in the ground truth, specifically `BGL` and `HPC`. Since our method averages the F1 score when multiple summaries are present, the final results were lower for datasets with a higher number of multiple summaries.

Nonetheless, for business applications, our scores confirm that our method is a practical alternative for summarizing and tracking the most relevant lines within a dataset. This approach helps save time that would otherwise be spent manually examining each log line. Taken together with the component-wise analysis in Table 5.10, these results answer RQ2 by showing that: (i) each of the three components ($M_{\text{text}}, M_{\text{var}}, M_{\text{close}}$) can dominate in different datasets; and (ii) appropriate parameter choices allow the proposed method to match or outperform classical summarizers in most scenarios, while remaining computationally competitive.

From an insight perspective, our experiments highlight that log summarization is not a one-size-fits-all problem: some systems are best summarized when textual content is emphasized, others when parsed variables carry most of the weight, and others when positional closeness is explicitly modeled. This provides concrete guidance for practitioners: by inspecting basic properties of a log dataset (e.g., vocabulary diversity, template variety, and the stability of variable fields), one can prioritize which component of the unified matrix M is likely to be most informative and tune (α, β, γ) accordingly.

5.5 Practical Application

After developing our method, we actively engaged in its practical implementation through collaboration with the developers of our industry partner. Given the multitude of parallel computer systems in their infrastructure, each generating significant volumes of log files, the utility of a log analysis method capable of effectively tracking the most relevant lines within extensive log files has proven to be an invaluable asset.

Our partner is one of the largest companies in their field, with thousands of employees. Their software development team handles numerous log files created by software build compilations within DevOps platforms, such as Jenkins. Some of these files are exceptionally large, reaching sizes of more than a gigabyte. Recently, our client established a dedicated machine learning team with the goal of extracting meaningful insights from these files.

We began by collecting raw log files from our client, which were then organized in JSON format. Our initial challenge stemmed from the size of the files, which exceeded several

gigabytes. Consequently, we had to optimize the computation of our matrices to efficiently handle large-scale calculations on the fly. This prompted us to design the matrices as sparse as possible, since working with dense matrices would have required more storage than was available. Additionally, our client had an existing in-house clustering method. Therefore, our method required fine-tuning to attempt to reproduce their clusters using our predefined matrices.

One noteworthy concern raised by the client's software developers was that an excessive number of clusters could negatively affect users by gradually decreasing overall system performance. While academic algorithms often strive to determine the optimal number of clusters through heuristics, it was essential to incorporate the developers' feedback to ensure maximum satisfaction. As mentioned earlier, we defined the optimal number of clusters by analyzing the previous clustering method used by our client, rather than relying on any specific rule. Additionally, our method allowed the client's software development team to validate different scenarios and review the final summaries without needing to manage machine learning hyperparameters directly.

Initially, our method lacked a closeness matrix. The partner was able to identify cases where closely positioned lines had a unified meaning, and relying solely on textual analysis was insufficient, as it overlooked the spatial arrangement of lines within the log file. Moreover, the need for user feedback regarding clustering constraints of type must-link and cannot-link led to the integration of a heuristic method [210] for a semi-supervised k-medoids. As part of the ongoing project, our client plans to develop a web application to collect real-time feedback from the users about the obtained clusters.

The entire implementation process spanned about a year, and our final method was converted into an API and integrated into our client's machine learning platform. Currently, we are working with our industry partner to determine the optimal number of clusters, aiming to improve the identification of relevant lines within software build log files.

5.6 Conclusion

This paper introduces a novel method that automatically summarizes and identifies the most relevant log lines within large, heterogeneous log datasets, enhancing the knowledge gained from classic log analysis. Our approach uses clustering as an intermediate step to group log data based on three key types of information: (i) log line embeddings, (ii) variables extracted from parsed log lines, and (iii) the proximity between log lines. From these clusters, we apply topic modeling and word analysis to summarize the logs and highlight lines that require closer

attention. Through quantitative evaluations on multiple log datasets, we demonstrate that our approach improves the accuracy and relevance of the generated summaries as compared to state-of-the-art text summarization methods.

Traditional log analysis methods are often overwhelmed by the scale and diversity of log data, which can vary greatly between different systems and applications. Moreover, existing tools frequently require a level of domain expertise and manual intervention that not all software engineers possess. Our research addresses this gap by introducing a flexible unified log dissimilarity matrix, able to focus on (i) and/or (ii) and/or (iii) to effectively summarize log text. By allowing users to fine-tune the hyperparameters associated with (i)-(iii), users can modify the weight of each information source in the overall summarization, effectively adapting the method to reflect the lexicon of the analyzed log structure.

Selecting the most important lines in software log files is a strategic approach with many applications in the maintenance, monitoring, and optimization of computer systems. The applicability of our project is especially relevant in production environments, where response time to failures is critical. Large volumes of logs make it difficult to identify errors and events of interest, delaying problem resolution, and impacting the user experience. Automatically selecting the most relevant lines, such as error messages, warnings, and anomalous behavior, allows developers and operations teams to focus their efforts on priority areas, facilitating faster and more assertive analysis. By implementing this functionality, our method can directly contribute to increasing system availability, mitigating interruptions, and reducing the operational costs associated with downtime.

In addition to real-time system monitoring, our intelligent log selection plays a central role in the debugging process during software development. During tests and executions of applications in the construction phase, log files tend to contain a heterogeneous mix of information: debug messages, flow events, integration results, and error records. Manually analyzing this content can be inefficient and time consuming. In this sense, the use of filtering algorithms based on Natural Language Processing makes it possible to identify the most relevant records and facilitate bug tracking. This process improves not only the productivity of the developer, but also the quality of the code delivered.

Another important aspect of applicability of our project is in predictive analysis and fault diagnosis, a field in which integration with Machine Learning techniques is promising. Subtle patterns in logs that indicate imminent failures or performance degradation are not always evident through traditional analysis. Automated methods can be used to detect anomalies or recurring behaviors associated with future problems. Identifying these patterns automatically allows our method to anticipate incidents before they impact the system, promoting a

proactive approach to software maintenance.

In addition, the relevance of our project like this extends to the storage and auditing of logs. With the growth of systems and the requirement to retain records for regulatory compliance or security purposes, storing logs in their entirety can represent a significant cost. Selective filtering makes it possible to generate log summaries, ensuring that only the most important entries are stored for future analysis.

Moreover, in complex systems, such as those based on distributed architectures and microservices, selecting important logs can also facilitate system observability. In these architectures, different services and components generate logs simultaneously, making it difficult to correlate events and track down problems in distributed flows. Our project, by identifying and relating the most relevant lines between different files, can offer a cohesive view of the system's behavior, simplifying problem analysis, and improving traceability.

In short, selecting the most important lines in log files transcends simple data filtering, positioning itself as an essential component for observability, proactive maintenance and optimization of modern software systems. Its applicability is vast, ranging from real-time monitoring to predictive failure analysis, directly reflecting operational efficiency, cost reduction, and the quality of the solutions developed. This project, especially when combined with modern artificial intelligence and machine learning techniques, offers not only practical solutions to current challenges, but also innovative perspectives for the intelligent management of complex systems.

Through rigorous experimentation, we achieved the highest summarization accuracy across four of the six benchmark datasets used in the literature. Our findings highlight the potential of our method to address the log analysis challenges faced by real-world users, paving the way for further advancements in the field. As future work, we intend to refine our method by incorporating statistical techniques for sentence selection, rather than relying solely on the number of shared tokens.

From a scientific perspective, our main contribution lies in showing that a unified, parameterized dissimilarity representation of log lines (combining textual embeddings, parsed variables, and positional proximity) can be used to systematically study and improve log summarization. By explicitly analyzing the impact of the main hyperparameters and by isolating the individual contribution of each component of M, we go beyond simply stacking existing methods and provide a principled framework for understanding how different aspects of log structure affect summarization quality.

At the same time, we acknowledge the existing limitations of our current work. First, we

rely on manually tuned weights (α, β, γ) rather than learning them automatically from data. Second, our experimental comparison focuses on strong classical summarizers and topic-modeling baselines, and does not include large, supervised deep learning summarization models adapted to logs. As discussed above, this is due to practical constraints on data and compute as well as the lack of standardized log summarization corpora, but it also opens an interesting avenue for future research: using our unified matrix and clustering framework as a comprehensive, log-specific front-end for more powerful neural summarizers. Fundamentally, the use of BerTopic, resting on BERT-based large language models, already positions our approach within the broader landscape of LLM-based techniques, while maintaining a clear separation between semantic representation (via LLM embeddings) and controllable, explainable summarization steps (via clustering and cTF-IDF).

CHAPTER 6 ARTICLE 3: EXTRACTING CAUSAL RELATIONS FROM LOG SEQUENCES USING CAUSAL LANGUAGE MODELS

Authors

Vithor Bertalan, Fateme Faraji Daneshgar and Daniel Aloise. Published at the 33rd IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER 2026). Published on 2025-12-18. Not yet available for public access.

Acknowledgements

We would like to gratefully acknowledge the Natural Sciences and Engineering Research Council of Canada (NSERC), Ericsson, Ciena, and EffciOS for funding this project. Moreover, this research was enabled in part by the support provided by Compute Canada.

Abstract

Understanding causal relationships in system logs is crucial for diagnosing complex software behaviors. Traditional causality extraction methods fall into two categories: constraint-based and time-series-based. Constraint-based approaches, such as the PC algorithm, assume fixed variables and well-defined joint distributions, which conflict with the sequential and dynamic structure of log data. Time-series-based methods, like Granger causality, can be applied to transformed log sequences but suffer from extreme sparsity, resulting in unstable and unreliable causal estimates. We propose a novel framework that leverages causal language models (CLMs) to estimate conditional probabilities among log events, enabling robust conditional independence testing directly on sequential data. By predicting event probabilities given preceding contexts, our method captures long-range and nonlinear dependencies that conventional frequency-based techniques often miss. Experiments on real-world logs demonstrate that this approach reveals meaningful causal relations, offering a promising direction for reliable causal inference in software log analysis.

IEEE Keywords: Large language models, Cause effect analysis, Software

6.1 Introduction

Understanding causal relationships among events is fundamental to diagnosing and predicting system behavior in complex software infrastructures. Logs generated by large-scale systems provide a detailed chronological record of operational activities, failures, and dependen-

cies among components. Extracting causal links from these logs enables fault localization, anomaly explanation, and proactive system management. However, despite their promise, **traditional causal discovery algorithms**, such as the PC algorithm or its derivatives, face substantial challenges when applied to log data.

In Table 6.1, an example of causality between logs is presented. In the first case, the source line indicates a request to load an authentication module and the target line represents that the connection did not end successfully. In the second case, a request to save the current settings is successfully completed in the target line. Finally, in the third line, user authentication fails because the user is not known to the identity management system. In all of those cases, the second line could not have happened without the first. Therefore, we can conclude that there is a causal link between the two. It is important to highlight that the first messages might have different outputs. However, the target lines need necessarily to be accompanied by the presence of the sources.

Table 6.1 Example of causal links between logs

Source Log Line	**Target Log Line**
Loading authentication module	Authentication module failed to initialize
User clicked "Save Settings"	Settings updated successfully
Authenticating user *jdoe*	User *jdoe* not found. Access denied

Conventional causal discovery methods are typically designed for tabular, independently, and identically distributed (i.i.d.) data, where each variable represents a random quantity with sufficient sampling across observations. These algorithms rely on statistical independence tests (e.g., chi-square or conditional independence tests) to prune and orient edges in a causal graph. In contrast, log data are inherently sequential, sparse, and context-dependent.

Traditional causal discovery methods can generally be divided into two categories: **constraint-based** and **time-series–based** approaches. Constraint-based methods, such as the **PC algorithm** [4], require tabular data and assume that all variables are observed across many i.i.d. samples. Converting log sequences into a tabular form for these algorithms inevitably destroys or obscures crucial ordering information, leading to unreliable independence tests and incorrect causal structures. Furthermore, the high number of variables (log templates) commonly found in industrial datasets makes constraint-based approaches particularly ineffective. PC must test independence between every pair of variables while conditioning on **all possible subsets** of the remaining variables, an operation that grows combinatorially with dimensionality. In practice, on real industrial log data, this results in prohibitive runtime and, in many cases, failure to produce any result at all.

Time-series–based methods, such as **Granger causality** [5], preserve temporal dependencies but are highly sensitive to data sparsity and require dense, regularly observed signals. Industrial log data, however, are typically sparse and event-driven, making these methods unstable and often ineffective.

To address these issues, we propose a novel framework that integrates **language-model-based probability estimation** into the causal discovery process. Instead of relying on raw event counts, our method leverages the **probabilistic knowledge encoded in a trained causal language model (CLM)**, which captures sequential and contextual dependencies among log events, to estimate the **conditional mutual information (CMI)** between events. By computing CMI from the model's predicted probabilities, we obtain a continuous, context-aware measure of dependency that reflects how the presence of one event influences the likelihood of another, conditioned on relevant contextual events.

Although training a large language model (LLM) requires computational effort, it offers several long- term advantages over traditional causal discovery methods. First, the model is trained **once** and can be reused for future analyses without repeating expensive computations. Second, industrial environments generate massive volumes of log data, and leveraging more data generally leads to better estimates of statistical relationships. LLM-based approaches naturally benefit from this scale, as they can be trained on extremely large corpora of logs, whereas traditional causal discovery algorithms struggle to operate on high-volume, high-dimensional datasets due to their reliance on repeated statistical tests. Moreover, many companies today already maintain pretrained LLMs derived from their own log sequences. Developing methods that repurpose these existing models for downstream tasks, such as causal analysis, adds significant practical value and aligns with current industrial trends.

In summary, our contributions are threefold:

- **A probability-driven causal discovery framework**: We introduce a novel approach that replaces raw event-frequency statistics with probability estimates obtained from a trained causal language model. This enables computation of conditional mutual information that is continuous, context-aware, and robust to data sparsity.
- **A scalable solution for high-dimensional log data**: By leveraging the expressive power and scalability of LLMs, our method can effectively utilize massive volumes of industrial log data, far beyond what traditional constraint-based or time-series–based causal discovery methods can handle. This allows reliable causal inference even in settings with thousands of log templates and highly sparse event occurrences.
- **A practical way to reuse existing pretrained models**: We demonstrate how pre-

trained LLMs, already deployed in many industrial environments for tasks such as log parsing or anomaly detection, can be repurposed for causal analysis with minimal additional cost. This creates a highly practical and value-adding pathway for organizations to extract new causal insights from models they already maintain.

The code for our method is openly available for public scrutiny [1].

6.2 Bibliographical Review

The recent literature on log-based causality in software engineering depicts a rapidly emerging but still fragmented research area, spanning cloud microservices, networks, build systems, and scientific modelling software. A broad systematic review conducted by [211] shows that most work concentrates on testing and root cause localization, often relying on informal causal reasoning via manually constructed causality graphs rather than strict statistical inference or formal interventions. Within this landscape, logs appear as one of the main observable artefacts of complex systems, and a growing line of research focuses explicitly on turning long, semi-structured log streams into representations suitable for causal discovery and diagnosis.

A first major line of work models logs as multivariate time series and applies Granger causality to uncover dependencies between components. In [212], the authors propose to transform discrete microservice logs into time series and then use both linear and neural autoregressive models to infer causal structure among microservices, reducing the need for intrusive instrumentation. The authors report that neural Granger models tend to outperform traditional linear methods on non-linear data, while linear Granger remains efficient and accurate on large, predominantly linear time series.

In a related but more reliability-oriented setting, the authors of [212] use Granger causalities extracted from logs to predict time-to-failure in long-running software and to diagnose degradation causes. They compare their framework with classical ARMA-based baselines and report higher prediction accuracy, indicating that Granger-style causal structure can be leveraged for predictive maintenance. Building on these ideas, [213] introduces a neural approach for root cause analysis in microservices.

Prior work in this space had applied structure learning algorithms such as the PC algorithm to logs, but these methods largely ignore temporal order and struggle with patterns where an anomaly (e.g., a CPU spike) precedes a downstream effect (e.g., increased latency). The approach in [213] tackles this limitation by embedding temporal context through a forecasting backbone and by framing causal discovery as neural Granger analysis combined with

[1] `https://github.com/vbertalan/EnhancedPC`

contrastive learning. It then uses a personalized PageRank variant to rank likely root causes. Experiments on synthetic and real-world microservice datasets report improvements over earlier root cause analysis techniques and indicate benefits of explicitly modelling temporal dynamics when reasoning about causal structure from logs.

A second cluster of work focuses on network logs such as syslog data in large-scale infrastructures. In [214], the authors apply causal inference to reconstruct causal relationships among network events over long time periods, with the explicit goal of filtering out spurious correlations that affect traditional correlation-based monitoring. Using 15 months of syslog data from a nationwide academic network, the method shows that causal inference can reduce pseudo-correlated events and help operators identify failures in practice.

Complementing this, the work of [215] compares the PC algorithm with MixedLiNGAM for network log analysis. The authors argue that PC often generates many undirected, unweighted edges that are hard to interpret in practice, whereas MixedLiNGAM produces weighted directed acyclic graphs, where directed, weighted edges provide clearer insight into which events are likely causes and which are effects. Together, these works suggest that causal discovery methods can complement correlation-based dashboards with causal explanations in network log analysis.

A different but related line of research addresses log causality from the standpoint of execution provenance and structural dependencies rather than purely statistical time-series models. In [216], the authors tackle the challenge of reconstructing causally consistent traces in distributed systems where logs are massive, interleaved, and affected by unsynchronized clocks. The approach in [216] uses kernel-level probes to link application-level log messages across components, encodes events as a DAG, and stores this graph in a database to enable causal queries. Evaluations on a complex ticket-booking microservice benchmark show that this approach can pinpoint root causes of anomalies with higher accuracy and better performance than the log-analysis systems used as baselines.

In a complementary domain, the work of [217] uses system-call traces as a uniform interface to capture build processes across environments. It then performs causality analysis over read/write and parent/child dependencies, coupled with differential analysis and similarity filtering, to build dependency graphs and locate the commands and files responsible for unreproducible builds. The evaluation over real-world software packages reports high accuracy in identifying both commands and files associated with unreproducible builds.

At a more conceptual level, in [218] the authors propose a formal model of "contextual event causal dependency" that unifies concepts from distributed and operating systems (namely, Lamport [219] and d'Ausbourg [220]) to reason about causality among heterogeneous logs

across abstraction layers (network, system, application). This model provides a theoretical foundation for reasoning about causal relationships among logged events in settings where events originate from multiple layers and sources.

Human-in-the-loop frameworks form a third major strand of the literature, seeking to combine automated causal inference with domain expertise. The work in [221] provides an end-to-end pipeline that transforms messy logs into variables aligned with chosen "causal units" (e.g., user, machine), aggregates and names them, and then supports what the authors call Exploration-based Causal Discovery. The framework in [221] proposes candidate causal edges (e.g., via average treatment effect estimates), and the user iteratively accepts or rejects them to construct a causal graph. The user interface extension surfaces key assumptions that drive each causal conclusion, helping users assess whether the answers are appropriate and mitigating the cascading effect of early mistakes. This design assumes that logs are partial, noisy observations of complex systems and that expert feedback is important for causal modelling.

Finally, in [6], the authors generalize the human-in-the-loop idea through a framework that uses causal inference over log-derived data to diagnose problems and reason about hypothetical interventions. This framework includes a log-conversion component to transform raw logs into tabular data, a candidate-cause ranking module that identifies likely causes of a variable of interest without requiring a full global causal graph, and an interactive graph-refinement module that focuses user judgements on the parts of the graph most relevant to the effect under study.

Evaluations on synthetic and real logs report improved precision and a reduction in the amount of manual causal judgement required, while remaining scalable to large log datasets. In parallel, the authors of [222] illustrate that similar causal-testing ideas can support software testing of complex scientific models, by using causal inference to infer metamorphic test outcomes from confounded historical data. Taken together with the systematic review of causal inference in software engineering, these works suggest a trend from early, often informal causal reasoning over logs toward combinations of statistical causal discovery, provenance-based graph construction, and interactive tools that incorporate expert knowledge to manage the noise, partial observability, and scale of real-world logging data.

6.3 Methodology

Figure 6.1 illustrates the two workflows we compare: the standard PC algorithm and our proposed LLM-augmented PC for extracting causal relationships from a collection of log

sequences.

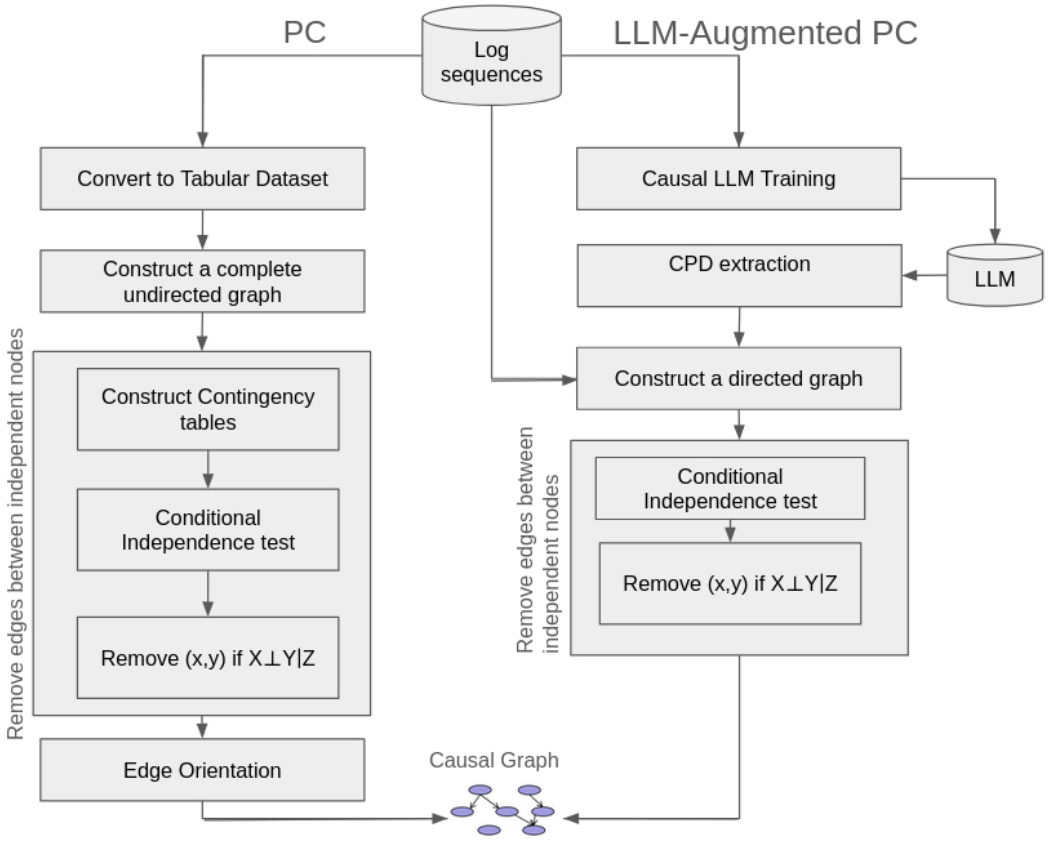

Figure 6.1 Comparison of workflows: PC and LLM-augmented PC

In the classical PC pipeline, a set of categorical variables (log templates) is obtained by converting each input sequence into a tabular representation. This tabularization typically discards some ordering information inherent in sequences. The PC algorithm then begins with a complete undirected graph over variables and iteratively removes edges by performing conditional independence tests: for each candidate edge (X, Y) and for conditioning sets $Z \subseteq V \setminus \{X, Y\}$, the test evaluates whether

$$X \perp\!\!\!\perp Y | Z \tag{6.1}$$

If independence is accepted for some Z, the edge (X, Y) is removed. The remaining skeleton is then oriented into a causal graph using standard orientation rules [4]. In domains with very many variables (as is common in software logs, where each log template corresponds to a node), the combinatorial number of conditioning sets Z makes PC computationally expensive

and statistically unstable: the number of subsets of size k grows as

$$\binom{|V|}{k} \tag{6.2}$$

and reliable conditional independence testing requires sufficient sample support for each conditioning cell.

To mitigate these issues, we propose an LLM-augmented approach. We first train a causal LLM on raw sequences of log templates so that the model learns sequence-level conditional distributions. From the trained model we extract conditional probability distributions (CPDs) $p(v \mid pa)$ between templates. Concretely, for two templates X and Y we use the model's estimate of $p(X \mid Y)$ (or $p(Y \mid X)$) as a prior score of a possible causal relation. We add a directed candidate edge $(X \to Y)$ whenever

$$p(X \mid Y) \geq \tau \tag{6.3}$$

for a chosen threshold τ. This thresholding step focuses downstream computation on node pairs with high probabilistic dependence according to the LLM, thereby dramatically reducing the number of pairs that require expensive statistical testing.

After constructing the candidate directed graph from CPDs, we apply a refined conditional independence test that uses the extracted CPDs directly. For each candidate pair (X, Y) and a chosen conditioning set Z (we use the set of events that appear before both X and Y in the input sequences as a natural, sequence-informed choice for Z), we compute the conditional mutual information, CMI [4]:

$$I(X; Y \mid Z) = \sum_{x,y,z} p(x, y, z) \log \frac{p(y \mid x, z)}{p(y \mid z)}. \tag{6.4}$$

Low values of $I(X; Y \mid Z)$ indicate that, once we know Z, knowledge of X provides little additional information about Y; therefore (X, Y) can be considered conditionally independent given Z and the candidate edge is removed. Intuitively, using CPDs from the LLM replaces raw frequency counts with smooth, model-based estimates of $p(\cdot)$, which is helpful in sparse regimes typical for log data. The CMI is always non-negative in expectation (it is a Kullback–Leibler divergence between joint and product conditional distributions) [5]; in practice we use a small threshold ε to accept independence when $I(X; Y \mid Z) \leq \varepsilon$.

The final directed graph is obtained after removing edges with low CMI and taking the candidate directions produced by the CPD thresholding as initial orientations. This hybrid

design therefore combines the strengths of data-driven causal discovery (the PC skeleton) with the sequence-sensitive, high-capacity modeling of LLMs to: (i) prioritize promising edges, (ii) provide smooth CPD estimates in sparse regimes, and (iii) reduce the computational burden of exhaustive conditioning searches. In the following subsections, each step is described in greater detail.

6.3.1 Representation of Log Sequences

The dataset consists of sequences of log templates extracted from system execution traces. Each template represents an abstracted form of a log message, independent of variable parameters. The causal LLM is trained using these sequences under a causal language modeling objective, where the model learns to predict the next event in a sequence given its historical context. Formally, for a sequence of templates $S = (t_1, t_2, \ldots, t_n)$, the model learns $P(t_{i+1} \mid t_1, t_2, \ldots, t_i)$. After training, this model captures the transition probabilities between templates that implicitly encode directional dependencies among events.

6.3.2 Conditional Probability Distribution (CPD) Extraction

In the proposed LLM-augmented causal discovery approach, we aim to estimate the conditional probability distributions (CPDs) among log templates directly from a pretrained causal language model (CLM). Unlike classical causal discovery algorithms such as PC [4], which rely on frequency-based contingency tables to approximate dependencies, our approach leverages the predictive capacity of an LLM to infer probabilistic relationships among events. When a causal language model receives a context sequence $X = [x_1, x_2, \ldots, x_t]$, it processes it through a series of transformer layers and outputs a logit vector.

$$\mathbf{z} = f_\theta(X) \in \mathbb{R}^{|V|} \tag{6.5}$$

where f_θ denotes the model's forward function parameterized by θ, and $|V|$ is the vocabulary size. Each element z_i in this vector represents an unnormalized score that reflects how likely token v_i is to appear next, given the current context. To transform these raw logits into interpretable probabilities, the model applies the softmax function:

$$\pi_i = \frac{\exp(z_i)}{\sum_{j=1}^{|V|} \exp(z_j)} \tag{6.6}$$

where π_i denotes the normalized probability assigned to token v_i. This softmax distribution provides a ranking over all possible next tokens. To efficiently approximate the joint prob-

ability over multi-step event transitions, we recursively expand the most probable paths up to n steps ahead using a top-k exploration strategy. This process can be represented as a probability tree, where each node corresponds to a log event and each branch corresponds to a predicted next event with its associated probability (see Figure 2). The cumulative probability of a path $\pi = (x_1, x_2, \ldots, x_t, y)$ is given by

$$P(\pi) = \prod_{i=1}^{t} P(x_{i+1} \mid x_1, \ldots, x_i) \tag{6.7}$$

During the recursive search, branches with probabilities below a threshold τ are pruned to ensure computational efficiency. The algorithm caches intermediate model states (i.e., past key values) to avoid redundant forward passes, significantly reducing inference time. For each pair of tokens (x, y), the final CPD is computed as the sum of all path probabilities that terminate in y, normalized over the explored space.

$$P(y \mid x) = \sum_{\pi \in \mathcal{P}_{x \to y}} P(\pi) \tag{6.8}$$

where $\mathcal{P}_{x \to y}$ denotes the set of all paths from x to y discovered within n recursive expansion steps. The resulting probabilities are stored in an edge probability dictionary, where each key (x, y) represents a directed relation and the value corresponds to $P(y \mid x)$. These CPDs serve as the structural foundation for the subsequent conditional independence testing and causal graph construction. An illustration of this process is shown in Figure 6.2.

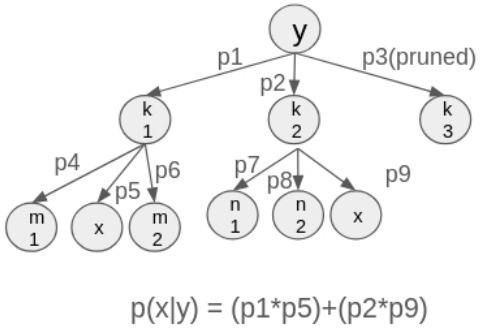

Figure 6.2 Example of a $\mathcal{P}_{x \to y}$ calculation

6.3.3 Conditional Mutual Information for Independence Testing

To evaluate whether two events X and Y are conditionally independent given a third event Z, we employ the conditional mutual information (CMI) defined as:

$$I(X;Y \mid Z) = \sum_{x,y,z} p(x,y,z) \log \frac{p(y \mid x,z)}{p(y \mid z)}. \tag{6.9}$$

In this context, $p(x, y \mid z)$ denotes the probability of observing the pair (x, y) in the presence of conditioning event z.

We know that:

$$p(x,y \mid z) = p(x \mid z)\, p(y \mid x,z). \tag{6.10}$$

Also:

$$p(x,y,z) = p(z)\, p(x,y \mid z) = p(z)\, p(x \mid z)\, p(y \mid x,z). \tag{6.11}$$

Substitute into the formula:

$$I(X;Y \mid Z) = \sum_{x,y,z} p(z)\, p(x \mid z)\, p(y \mid x,z) \log \frac{p(y \mid x,z)}{p(y \mid z)}. \tag{6.12}$$

Given the CPDs, we can compute the conditional mutual information (CMI) for each pair of variables (X, Y). For the conditioning set Z, we only include the log templates that have appeared in sequences prior to both X and Y. This results in a much smaller and more focused set of controlling variables compared to the standard PC algorithm.

The computed $I(X;Y \mid Z)$ value quantifies the degree of dependence between X and Y after accounting for the influence of Z.

- A high CMI value indicates that knowing Y reduces the uncertainty about X even when Z is known, implying a potential causal dependence.
- A low or near-zero CMI value suggests that X and Y are conditionally independent given Z, warranting the removal of a causal link between them.

Pairs with low CMI are then removed from the graph, effectively pruning weak or spurious dependencies.

6.4 Experimental Evaluation

We evaluate the effectiveness of the proposed LLM-augmented causal discovery framework by comparing it against the classical PC algorithm on real-world industrial log datasets. Our experiments are designed to answer the following research questions:

- **RQ1**: Does the LLM reduce the number of independence tests required, thereby improving scalability?
- **RQ2**: Does the proposed framework improve the accuracy of discovered causal relationships compared to traditional causal discovery approaches?

6.4.1 Datasets

To evaluate our approach, we used a proprietary dataset provided by our industry partner, composed of execution traces collected from build logs of multiple software products over a six-month period. Domain experts identified a ground truth set of 15 causal links between log templates. For confidentiality reasons, the log entries cannot be disclosed. However, they are structurally similar to those presented in Table 6.1. From this dataset, we randomly sampled 200,000 build logs while preserving the natural frequency of sequences containing the known causal patterns.

Each build log consists of an ordered sequence of log lines, which were abstracted into log templates. To ensure reproducibility without revealing sensitive content, each template was replaced by a unique numeric identifier. Since the templates are anonymized, our experiments focus on template-level probability estimation and scalability rather than semantic or contextual analysis.

After the sampling, the resulting dataset is characterized by:

- **200,000** log files
- **1,487** unique templates
- **Highly sparse sequences** (median length: 15)
- **9** known causal links

6.4.2 Baselines

We compare the following methods:

- **PC (Classical):**
  Standard constraint-based causal discovery using chi-square conditional independence tests and exhaustive conditioning.
- **Granger Causality:**
  A time-series baseline widely used for event-driven data.
- **LLM-PC (Our Model):**
  Full proposed approach using CPD thresholding plus CMI pruning.

6.4.3 Implementation

LLM architecture We use a decoder-only GPT-NeoX configured from scratch with a SentencePiece tokenizer wrapped by a `T5Tokenizer`. The tokenizer is trained on the raw log corpus and augmented with a dedicated line delimiter token. The model is instantiated with the following defaults: hidden size = 768, number of layers = 12, attention heads = 12, feed-forward size = 3072, and maximum positional embeddings = 2048. The vocabulary size equals the trained SentencePiece model (default 8,000), and the output projection is tied to this vocabulary. This setup yields a compact yet expressive decoder suitable for next-token causal language modeling directly over log streams.

Training We follow a standard causal language modeling objective (next-token prediction). Each non-empty log line receives the delimiter token sentinel. The corpus is tokenized, concatenated, and chunked into fixed-length blocks of size $L = 1024$. Labels mirror the inputs (teacher forcing). Unless otherwise specified, we train with: epochs = 3, per-device batch size = 8, gradient accumulation = 1, AdamW with learning rate 5×10^{-4} and weight decay 0.01. We also use Top-$k = 4$ and Beam depth = 5 for CPD extraction (probability tree parameters).

CMI Parameters We define the conditioning set Z with the events appearing before both X and Y. We also set the independence threshold with $\epsilon = 0.01$.

Hardware The script detects CUDA devices at runtime and can leverage bf16/fp16 and TF32 when available. In our runs we executed on a GPU-enabled server (64 GB RAM), using the script's defaults for AMP/TF32; the same code path supports single- or multi-GPU setups.

6.5 Results

RQ1: Scalability and Number of Independence Tests

In our dataset, we observed 1,487 unique log event templates, which implies a very large search space for constraint-based causal discovery methods such as the PC algorithm. In the worst case, PC evaluates conditional independence for every variable pair under every possible conditioning set. With $n = 1487$ variables, this results in

$$\binom{n}{2} = \binom{1487}{2} = 1{,}104{,}841$$

candidate pairs, and for each pair, up to 2^{n-2} possible subsets of conditioning variables. This yields approximately 1.18×10^{453} conditional independence tests — a number astronomically beyond any feasible computation.

The PC algorithm cannot handle our original dataset of over 200,000 log files due to the prohibitive number of conditional independence tests required. To make the evaluation computationally feasible while preserving the statistical characteristics of the data, we randomly sampled 2,000 log files, ensuring that the ratio of causal link occurrences in the sequences remained consistent with the full dataset.

Even when restricting the conditioning set to at most three variables, as is often done in practice to avoid exponential blow-up, the algorithm still requires roughly 6×10^{14} tests (over 600 trillion). Moreover, when we ran PC on a sampled subset of only 2,000 logs, the algorithm still produced 91 edges (over a reduced domain), confirming that even heavily downsampled data does not eliminate the computational burden. These calculations clearly illustrate that classical constraint-based causal discovery methods are fundamentally unable to scale to large-scale industrial log datasets.

In contrast, our approach uses an LLM-driven pruning strategy that greatly reduces the size of the search space before performing statistical tests. By extracting edge probabilities from the language model, we identify only 351 highly co-occurring candidate edges out of more than a million theoretical possibilities. After pruning low-probability edges, we need to investigate only 797 conditional independence tests in total — a reduction of over 11 orders of magnitude compared to the PC worst case (and 11 trillion times fewer than the practical, limited-PC scenario).

This results in a computational pipeline that is scalable, feasible to execute, and suitable for real industrial environments, where detecting meaningful causal relations matters more than exhaustively testing every possible dependency. Table 6.2 summarizes the results.

Table 6.2 Comparison of required independence tests and the number of edges

Method & Setting	Variables	Candidate Edges	Conditioning Strategy	Estimated CI Tests
PC (theoretical worst case)	1,487	1,104,841	All subsets (unbounded)	1.18×10^{453}
PC (practical limit: max 3 cond. vars)	1,487	1,104,841	Subsets of size ≤ 3	6×10^{14}
PC on reduced dataset (2,000 logs)	1,021 (reduced domain)	921,523 edges	Subsets of size ≤ 3	177,072,594
Our LLM-based method (full dataset)	1,487	351 pruned edges	Model-driven dependency	797 CI tests total

RQ2: Accuracy of Discovered Causal Structure

In this subsection, we evaluate the effectiveness of our proposed approach in recovering ground-truth causal relations from real industrial log data. We compare our method against two baseline families: (i) traditional constraint-based causal discovery (PC algorithm) and (ii) time-series–based methods (Granger causality).

As demonstrated in RQ1, the PC algorithm cannot handle our original dataset of over 200,000 log files due to the prohibitive number of conditional independence tests required. To make the evaluation computationally feasible while preserving the statistical characteristics of the data, we randomly sampled 2,000 log files, ensuring that the ratio of causal-link occurrences within sequences remained consistent with the full dataset. The PC results reported in this section are therefore based on this reduced dataset. Table 6.3 summarizes the outcomes for each approach in terms of true positives (TP), false positives (FP), precision, recall, and F1-score.

Table 6.3 Results obtained by each approach

Method	Precision	Recall	F-1 Score
PC	0.01	0.11	0.2
Granger	0	0	0
LLM-Augmented PC	**38.4**	**89**	**53.3**

6.5.1 PC results

The PC algorithm produced a dense causal graph containing over 100 directed edges, but only one of the nine ground-truth causal relations was recovered, and that edge appeared as *undirected* (211–212), meaning the direction could not be inferred. This corresponds to a precision of approximately 0.01 and a recall of 0.11. The remaining eight true causal links were entirely missed, while the algorithm introduced a large number of spurious causal edges. These results

demonstrate that PC is not well suited for log-derived sequences, where data is sparse, temporal dependencies are weak and irregular, and independence tests frequently become unreliable. This failure highlights a fundamental mismatch between constraint-based causal discovery and the statistical properties of real-world system logs; most notably, **constraint-based causal discovery is designed for dense observational data, not event streams**.

6.5.2 Granger results

To evaluate Granger causality on our dataset, we treated each detected causal edge with a non-zero score as a predicted causal link and compared these against the 9 ground-truth template-level causal relations provided by domain experts. Granger causality failed to recover any of the known causal links, resulting in a precision of 0.0 and a recall of 0.0. The maximum Granger score was approximately 0.006, far below commonly used thresholds for statistical significance. Although we experimented with multiple threshold settings, Granger causality was unable to identify any of the 15 true causal relationships. The strongest detected signal remained extremely weak (≈ 0.006), indicating no statistically meaningful temporal dependencies were captured. These findings illustrate a fundamental limitation of Granger causality in settings where sequences are sparse and events do not exhibit continuous or autocorrelated temporal behavior—a common characteristic of real-world system logs.

6.5.3 Our method's results - LLM-based CMI

Our approach recovered **8 of the 9** ground-truth causal links (recall ≈ 0.89) while producing 10 spurious edges (precision ≈ 0.38), resulting in an F1-score ≈ 0.53. This indicates that the method is highly sensitive and captures the vast majority of true causal relations in sparse, high-dimensional log data. From an industrial standpoint, high recall is far more critical than eliminating all false positives, because missing a true causal dependency can lead to failures that remain undiagnosed, cascading outages, and repeated system incidents. False positives can be filtered later through expert review or downstream validation, but false negatives directly hinder root-cause analysis and fault localization. The reduced false-positive count (10) combined with the nearly near-complete recovery of ground truth confirms that our LLM-based CMI approach successfully prioritizes meaningful causal edges while keeping noise at a manageable level, a key requirement in operational environments where reliability and failure diagnosis are paramount.

6.6 Threats to Validity and Limitations

6.6.1 Construct validity

Our operational notion of causality is instantiated through next-token probabilities from a decoder-only LLM and the induced conditional probability distributions (CPDs), followed by conditional mutual information (CMI) pruning. This captures *local* causal transitions in log sequences rather than full structural causal mechanisms with explicit interventions. As such, long-lag effects, latent confounding across services, and non-immediate dependencies may be underrepresented. Moreover, the chosen thresholds (e.g., CPD cutoffs, CMI ε) and beam parameters (Top-k, depth) introduce design degrees of freedom that may affect edge recall/precision; we mitigate this by reporting Top-k results and by keeping thresholds fixed across datasets, but residual sensitivity remains.

6.6.2 Internal validity

Method-level comparisons may be affected by configuration choices and input representations. Our LLM operates directly on raw text with subword structure, whereas PC and Granger consume a derived tabular/time-series encoding; this representational mismatch can advantage the LLM in capturing semantic regularities within lines. We restricted PC conditioning depth and selected a fixed Granger lag/regularization for tractability; more exhaustive tuning or alternative estimators (e.g., partial correlation tests, LiNGAM variants, neural Granger) could change baseline performance. Runtime comparisons are further influenced by hardware and software stacks (GPU acceleration, AMP/bf16, TF32 kernels) that benefit the LLM pipeline but are not universally available for all causal discovery libraries. We reduce these risks by (i) fixing hyperparameters *a priori*, (ii) reusing the same input corpus across methods, and (iii) reporting when a method (PC) could not complete at full scale, yet differences in engineering maturity and compute backends remain a confound.

6.6.3 External validity

Synthetic data allow quantitative evaluation against known ground truth, but necessarily simplify real logging ecosystems (template drift, heterogeneous formats, multi-service timing, and non-stationary deployments). Results on the industrial dataset are constrained by an evolving ground truth: causal labels were recently initiated across teams, may be incomplete or noisy, and can change direction after deeper inspection (e.g., $X \to Y$ later revised to $Y \to X$). Consequently, low absolute Top-k counts should be interpreted with caution: the model can surface plausible relations that are not yet annotated, and some annotated links

may later be revised. Generalization to other domains, log styles, and tooling depends on the availability of comparable training corpora and on governance processes (human-in-the-loop review) to adjudicate candidate edges. Future work should broaden real-world coverage (multiple partners, longer periods), include intervention-style tests where feasible, and assess robustness under template churn, clock skew, and missingness.

6.7 Industry Application

In our deployment with the industrial partner, the most immediate challenge was **identifying causal lines** in the first place. Although engineers were familiar with failure modes and routine operational patterns, there was no established procedure for curating *explicit* causal pairs from raw logs. Determining whether a given sequence reflected a true cause–effect relation or a recurring but incidental co-occurrence required cross-team arbitration (ML, software engineering, and operations). Early iterations revealed gaps in shared vocabulary (e.g., what constitutes a "trigger" vs. a "symptom"), inconsistent template usage across services, and limited access to system-level context, all of which slowed the creation of a reliable ground truth. This experience suggests that successful industrial adoption depends as much on *process*, clear annotation guidelines, ownership, and escalation paths, as on modeling choices.

A second, pervasive obstacle was the **high volume of noise** in production logs. Many lines encode boilerplate scaffolding, monitoring heartbeats, or verbose diagnostics that rarely carry causal signal for downstream effects. Such events inflate the state space, degrade classical independence tests, and can distract LLM-based scoring if not regulated by sensible thresholds. We mitigated this by (i) retaining a neutral, data-agnostic representation, (ii) using probabilistic edge prioritization (CPD thresholding) to downweight low-signal transitions, and (iii) pruning with conditional mutual information to filter spurious links. Even so, the operational lesson is that noise governance (template hygiene, log-level policies, and de-duplication) is a prerequisite for efficient causal analysis at scale.

Finally, **validating discovered relations** proved difficult because the partner lacked a mature procedure to verify whether proposed edges were truly causal. In practice, domain experts needed time to reproduce scenarios, consult service owners, and align on directionality. Cases initially labeled as $X \to Y$ were occasionally revised to $Y \to X$, and some important lines were later reclassified as noise. To cope with this reality, we adopted a human-in-the-loop workflow: the model surfaces *ranked* candidates (Top-k) and engineers adjudicate which links to accept, postpone, or discard. This iterative validation, combined with traceability (linking edges to exemplar sequences) and lightweight playbooks for on-call investigation,

turned out to be more effective than seeking one-shot ground truth. The outcome is a practical pathway: use the model to accelerate hypothesis generation and focus expert attention where it matters most, while institutionalizing feedback to refine labels and logging practices over time.

6.8 Conclusion

This paper introduced a data-agnostic framework that reframes causal discovery in software logs as a model-internal scoring problem: instead of engineering tabular features or running expansive independence tests, we probe a decoder-only LLM (GPT-NeoX style) trained with `SentencePiece` and a simple line delimiter to read off next-token scores and derive conditional probability distributions (CPDs). This direct use of the model's hidden state and output layer provides a principled way to prioritize edges, capture intra-line semantics and inter-line transitions in the same objective, and drastically reduce the search space prior to independence testing. Conceptually, this departs from both constraint-based and time-series baselines by exploiting the representational capacity of modern LLMs rather than relying on external solvers or rigid temporal aggregation.

Building on these probabilities, our pipeline couples a lightweight CPD threshold with conditional mutual information (CMI) pruning. The result is a hybrid procedure that preserves the strengths of data-driven discovery (explicit tests and orientations) while benefiting from the LLM's smooth estimates in sparse regimes typical of logs. To the best of our knowledge, prior work has not extracted causal links by systematically *probing decoder layers* to construct CPDs and then using those CPDs to steer independence testing.

Empirically, the approach delivers superior accuracy and efficiency on synthetic benchmarks with known ground truth: it achieves markedly higher Top-1/Top-5 recovery than Granger and PC while finishing end-to-end faster (and at scales where PC could not complete without downsizing the data). In practical terms, this means the method surfaces *more true causal links in less time*, improving the value of downstream triage and root-cause workflows. On real industrial logs, absolute Top-k counts are lower across all methods, reflecting an evolving, multi-team annotation process, but our model remains the only one to recover a non-trivial subset of labeled edges while plausibly proposing additional relations that merit expert review. These outcomes align with the design goal: maximize recall of meaningful candidates under operational constraints, leaving fine-grained adjudication to domain experts.

Finally, the framework is readily adoptable. It trains once, reuses the same decoder for multiple analyses, and exposes ranked causal suggestions that practitioners can select or filter

according to context and risk. The lessons learned with our partner: cross-team alignment, iterative ground-truth curation, and occasional direction reversals, underscore that real-world causality is a moving target. A scalable, semantics-aware method that accelerates hypothesis generation is therefore especially valuable. Looking ahead, we see opportunities to extend the approach with longer-lag reasoning, intervention-style evaluations, and active-learning loops that use human feedback to refine both the CPDs and the final graph, further strengthening the bridge between LLM representations and actionable causal understanding in software systems.

As a direction of future research, we plan to continue this research by investigating the applicability of this research to other types of logs than the ones examined by this study, such as runtime, event and audit logs, to assess the robustness of this approach with respect to real-time system execution patterns.

CHAPTER 7 CONCLUSION

The present thesis explores the application of language models to the analysis of software logs. It progressively builds on the capabilities of such models, moving from log parsing to topic modeling, and finally to causal analysis. In addressing those three issues, it attempts to bridge the semantic gap between low-level log data and higher-level language model mechanisms.

Section 7.1 summarizes the major contributions of the thesis. Section 7.2 discusses their possible drawbacks. Section 7.3 debates feasible improvements to be performed in the research. Finally, Section 7.4 concludes the discussion.

7.1 Summary of Works

In this section, we present a summary of our published papers, and some general details about their methodology and evaluation.

7.1.1 Article 1

In Section 4, we propose the development of a dataset-agnostic log parsing framework that integrates language-model-based embeddings with a word analysis mechanism in which the method computes the frequency and detects the vocabulary of tokens. Unlike traditional log parsers that rely heavily on handcrafted rules, preprocessing heuristics, or dataset-specific tuning, the proposed method operates in an unsupervised manner and avoids mandatory prior knowledge of log structure. By combining semantic embeddings, density-based clustering (HDBSCAN), relative token frequency within clusters, and verification against a large English vocabulary, the approach systematically separates static fields from variables while preserving flexibility across heterogeneous log formats.

This is due to the fact that the Transformer-based log parsing approach offers significant adaptability. Since our approach is able to effectively handle diverse datasets, it demonstrates high versatility in adapting to new log line structures. As mentioned before, log lines do not share the same vocabulary, often varying significantly in structure. Thus, having a method that is able to flex itself enough to accommodate different log lines can be a very useful asset in the log parsing process.

Finally, empirical evaluation shows competitive or superior parsing accuracy compared to state of the art methods, and industrial validation confirms its practical viability.

7.1.2 Article 2

In Section 5, a clustering-based log summarization model was proposed, which adopts the unified dissimilarity representation of log lines. Instead of dealing with log data as general natural language text, the proposed model leverages the convex combination of three different dissimilarity measures, namely, textual, structural, and positional similarities. These different types of similarities have been combined in a single representation, which offers an effective way of dealing with the heterogeneous nature of log data.

While most of the existing log summarization models focus on processing raw text, the proposed model first clusters the log lines based on their textual characteristics using the k-medoids clustering algorithm and then applies the BerTopic model for topic modeling within each of the clusters. The model is able to deal with different combinations of textual, structural, and positional similarities depending on the datasets. Through extensive experiments, the proposed model has achieved better evaluation metrics compared to the state of the art for most of the datasets.

Moreover, our method is data-agnostic, being able to adapt itself to several different log semantics. This framework proves especially valuable in complex environments where manual inspection of raw log data is practically unfeasible, providing log analysts with the main topics on which they can focus their efforts.

7.1.3 Article 3

In Section 6, we introduce a new approach to address the issue of causal discovery from software logs. The traditional approach, based on constraint-based methods like the PC algorithm, is not effective in high-dimensional settings, while time-domain methods like Granger causality are not robust to sparse, event-driven log data. To address these challenges, a new approach to causal discovery was introduced, where a language model is used to estimate conditional probability distributions among log templates.

Thus, unlike traditional approaches that use independence testing as a measure to select edges, our approach uses a language model to estimate conditional probabilities, which is then used to significantly prune the search space before conducting CMI pruning.

The experiments on industrial log data show that our approach is effective in significantly reducing the number of independence tests while increasing recall on ground-truth causal relationships compared to traditional approaches. In particular, our approach is able to discover a majority of known causal relationships where traditional approaches failed to scale or failed to identify meaningful dependencies.

The causal discovery framework enhances the interpretability of event dependencies, providing a more robust understanding of system behavior. Tracing the events that caused the current log lines that are being analyzed is a very useful strategy to prevent bugs from happening before they occur. Even if the analysts are looking at the data *post factum*, learning the root cause of events is still useful knowledge to have for future incidents.

These three contributions represent a cohesive sequence of contributions from robust structural extraction (*log parsing*), semantic abstraction (*log topic modeling*), and finally causal interpretation (*log causal discovery*). Collectively, the results presented in this thesis offer a unified view of log analysis: software logs are neither simple natural language nor simple time-series data, but rather semi-structured sequential data whose textual content, structural variables, and contextual dependencies need to be modeled together. The work presented in this thesis offers both methodological contributions and practical tools for large-scale industrial log analysis.

7.2 Limitations

The limitations of the first paper involve the method's reliance on token variability between sentences, which could hinder accuracy in datasets where variables do not frequently change. Additionally, flaws and subjectivity in the LogHub ground truths affect the precise validation of results, as shown in 4. The parser also could lose performance in large clusters when the specific lines containing those variables occur with low frequency.

The second paper notes that the method fails to identify variables in certain log structures, such as those in the Proxifier dataset, leading to an ineffective and sparse variable matrix. Furthermore, the study does not compare results against end-to-end deep learning summarization models due to a lack of standardized log summarization corpora, as mentioned in 5.

The third paper states that the analysis captures local causal transitions rather than full structural mechanisms, potentially underrepresenting the long-lag effects or latent confounding across services. The selection of specific thresholds and beam parameters could introduce design degrees of freedom that affect edge recall and precision. Finally, industrial validation is constrained by a ground truth specifically tailored to our partner needs, lacking a possible generalization.

As general remarks, despite the promising results achieved by the proposed approaches, there are still some limitations. First, although the unified dissimilarity framework for log summarization has been validated by promising results, the weights α, β and γ used in the

framework need to be manually tuned. Although the experiment results have confirmed that the performance is robust to moderate changes in the weights, it would be even better if the weights could be learned automatically from the validation objectives.

Second, it should be noted that even though contextual embeddings enhance semantic coherence, interpreting the discovered topics frequently necessitates deep domain expertise, in a human in the loop approach. This requirement stems from the fact that log messages often contain technical jargon, unique identifiers, or system-specific references that diverge from traditional linguistic semantics, which may be known only to key users of the software systems.

Third, we point out that the lack of standard evaluation benchmarks and causal log datasets represent a significant drawback, since it limits the ability to rigorously test models under controlled circumstances, which is necessary for proper evaluation. Additionally, confidentiality agreements and other clauses related to the release of industrial log datasets also limit their release to the public, thus hindering the reproducibility of many articles that deal with log causality in the literature.

Finally, one of the main limitations of the present work is the time and computational cost of training the models used in the experiments. It is known that models with high depth and large numbers of parameters require high-performance computers with high memory capacity, GPUs, long training times, and energy costs. These issues can hinder the reproducibility of the experiment, the number of iterations of the fine-tuning process, and the use of the proposed method in scenarios with limited computational capabilities.

7.3 Future Research

Dealing individually with the three published articles, we can detect possible ameliorations that could be added to their content. For the first paper,

For the second paper, developing an automated method to determine the three matrices weights, based on the textual characteristics, would offer significant gains to the overall process.

A first potential extension of the proposed work is the integration of the proposed summarization framework with the proposed causal discovery framework as a unified framework for log analysis. This integration could enable more powerful tools capable of detecting causal relationships among log lines, improving root cause analysis and predictive maintenance in complex distributed systems.

Moreover, the absence of publicly available, well-annotated datasets significantly limits re-

producibility and fair comparison across methods. Future work could focus on constructing synthetic yet realistic log corpora. In parallel, novel evaluation metrics tailored to causal structure learning in logs, going beyond traditional precision and recall, could be proposed to capture temporal consistency, robustness to noise, and scalability in distributed environments.

Another prominent avenue for future research is the exploration of counterfactual causality in log analysis, performing a *what-if* analysis of the system. Instead of simply determining the directional dependencies between various events in the log, future models may be able to determine how the system would have behaved if a certain event in the log had not been recorded or if it had been recorded under different circumstances.

One final potential direction is the creation of smaller, more efficient language models that are specialized for log analysis tasks. Although large models demonstrate strong representational power, the computational requirements and training times of these models may be limiting for deployment scenarios. Future work could investigate smaller models, efficient fine-tuning approaches, and domain-specific pretraining for achieving state of the art results with significantly reduced computational overhead.

7.4 Final Remarks

To conclude, we stress that this thesis has shown that language models can act as a unifying framework for various log analysis tasks. Despite the challenges with the language model approach to log analysis, especially with regard to computational costs, interpretability, and causal correctness, our results have shown that combining language models with structured reasoning approaches is a promising direction for intelligent log analysis in complex software systems.

We would like to emphasize that the log data used in our articles and the code are all publicly available in popular repositories.

REFERENCES

[1] H. Seo *et al.*, "Programmers' build errors: a case study (at google)," in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 724–734.

[2] F. Wu, P. Anchuri, and Z. Li, "Structural event detection from log messages," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 1175–1184.

[3] J. Qiu *et al.*, "A causality mining and knowledge graph based method of root cause diagnosis for performance anomaly in cloud applications," *Applied Sciences*, vol. 10, no. 6, 2020. [Online]. Available: https://www.mdpi.com/2076-3417/10/6/2166

[4] P. Spirtes, C. N. Glymour, and R. Scheines, *Causation, prediction, and search*. Cambridge, MA, USA: MIT Press, 2000.

[5] C. W. J. Granger, "Investigating causal relations by econometric models and cross-spectral methods," *Econometrica: journal of the Econometric Society*, pp. 424–438, 1969.

[6] M. Markakis *et al.*, "From logs to causal inference: Diagnosing large systems," *Proc. VLDB Endow.*, vol. 18, no. 2, p. 158–172, Oct. 2024. [Online]. Available: https://doi.org/10.14778/3705829.3705836

[7] J. Zhu *et al.*, "Tools and Benchmarks for Automated Log Parsing," *Proceedings - 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE-SEIP 2019*, pp. 121–130, 2019.

[8] S. Nedelkoski *et al.*, "Self-supervised Log Parsing," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 12460 LNAI, pp. 122–138, 2021.

[9] H. Guo, S. Yuan, and X. Wu, "LogBERT: Log Anomaly Detection via BERT," *Proceedings of the International Joint Conference on Neural Networks*, vol. 2021-July, 2021.

[10] A. Janes, V. Lenarduzzi, and A. C. Stan, "A continuous software quality monitoring approach for small and medium enterprises," in *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*, ser. ICPE '17 Companion. New York, NY, USA: Association for Computing Machinery, 2017, p. 97–100. [Online]. Available: https://doi.org/10.1145/3053600.3053618

[11] T. Barik *et al.*, "Do developers read compiler error messages?" in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 2017, pp. 575–585.

[12] I. H. Sarker, "Machine learning: Algorithms, real-world applications and research directions," *SN Computer Science*, vol. 2, no. 3, pp. 1–21, 2021.

[13] Y. Zhong and Y. Guo, "Design and implementation of log parsing system based on machine learning," *Journal of Computer Applications*, vol. 38, no. 2, p. 352, 2018.

[14] M. Du and F. Li, "Spell: Online Streaming Parsing of Large Unstructured System Logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 11, pp. 2213–2227, 2019.

[15] B. Debnath *et al.*, "Loglens: A real-time log analysis system," in *2018 IEEE 38th international conference on distributed computing systems (ICDCS)*. IEEE, 2018, pp. 1052–1062.

[16] N. Aussel, Y. Petetin, and S. Chabridon, "Improving performances of log mining for anomaly prediction through nlp-based log parsing," *Proceedings - 26th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, MASCOTS 2018*, pp. 237–243, 2018.

[17] A. Brown *et al.*, "Recurrent neural network attention mechanisms for interpretable system log anomaly detection," in *Proceedings of the First Workshop on Machine Learning for Computing Systems*, 2018, pp. 1–8.

[18] Q. Cao, Y. Qiao, and Z. Lyu, "Machine learning to detect anomalies in web log analysis," in *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*. IEEE, 2017, pp. 519–523.

[19] J. Inoue *et al.*, "Anomaly detection for a water treatment system using unsupervised machine learning," in *2017 IEEE international conference on data mining workshops (ICDMW)*. IEEE, 2017, pp. 1058–1065.

[20] S. He *et al.*, "Experience report: System log analysis for anomaly detection," in *2016 IEEE 27th international symposium on software reliability engineering (ISSRE)*. IEEE, 2016, pp. 207–218.

[21] T. Schindler, "Anomaly detection in log data using graph databases and machine learning to defend advanced persistent threats," *arXiv preprint arXiv:1802.00259*, 2018.

[22] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 1st ed. USA: Prentice Hall PTR, 2000.

[23] H. Hamooni *et al.*, "LogMine: Fast pattern recognition for log analytics," *International Conference on Information and Knowledge Management, Proceedings*, vol. 24-28-Octo, pp. 1573–1582, 2016.

[24] D. El-Masri *et al.*, "A systematic literature review on automated log abstraction techniques," *Information and Software Technology*, vol. 122, no. February, p. 106276, 2020. [Online]. Available: https://doi.org/10.1016/j.infsof.2020.106276

[25] J. Abrahams, "Code and parse trees for lossless source encoding," *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*, pp. 145–171, 1997.

[26] M. Mizutani, "Incremental mining of system log format," in *2013 IEEE International Conference on Services Computing*, 2013, pp. 595–602.

[27] T. J. VanderWeele and J. M. Robins, "Four types of effect modification: a classification based on directed acyclic graphs," *Epidemiology*, vol. 18, no. 5, pp. 561–568, 2007.

[28] A. Vervaet, R. Chiky, and M. Callau-Zori, "USTEP: Unfixed Search Tree for Efficient Log Parsing," *Proceedings - IEEE International Conference on Data Mining, ICDM*, vol. 2021-Decem, no. Icdm, pp. 659–668, 2021.

[29] P. He *et al.*, "Drain: An online log parsing approach with fixed depth tree," in *2017 IEEE International Conference on Web Services (ICWS)*, 2017, pp. 33–40.

[30] D. Hu, "An introductory survey on attention mechanisms in nlp problems," 2018. [Online]. Available: https://arxiv.org/abs/1811.05544

[31] Z. M. Jiang *et al.*, "An automated approach for abstracting execution logs to execution events," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 20, no. 4, pp. 249–267, 2008. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.374

[32] S. Messaoudi *et al.*, "A search-based approach for accurate identification of log message formats," in *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*. IEEE, 2018, pp. 167–16 710.

[33] A. Zhong *et al.*, "LogParser-LLM: Advancing Efficient Log Parsing with Large Language Models," in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. Barcelona Spain: ACM, Aug. 2024, pp. 4559–4570. [Online]. Available: https://dl.acm.org/doi/10.1145/3637528.3671810

[34] Z. Ma, D. J. Kim, and T.-H. Chen, "LibreLog: Accurate and Efficient Unsupervised Log Parsing Using Open-Source Large Language Models," Nov. 2024, arXiv:2408.01585 [cs]. [Online]. Available: http://arxiv.org/abs/2408.01585

[35] Z. Jiang *et al.*, "LILAC: Log Parsing using LLMs with Adaptive Parsing Cache," *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, pp. 137–160, Jul. 2024. [Online]. Available: https://dl.acm.org/doi/10.1145/3643733

[36] Z. Ma *et al.*, "LLMParser: An Exploratory Study on Using Large Language Models for Log Parsing," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. Lisbon Portugal: ACM, Apr. 2024, pp. 1–13. [Online]. Available: https://dl.acm.org/doi/10.1145/3597503.3639150

[37] C. Almodovar *et al.*, "LogFiT: Log Anomaly Detection Using Fine-Tuned Language Models," *IEEE Transactions on Network and Service Management*, vol. 21, no. 2, pp. 1715–1723, Apr. 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10414427/

[38] W. Guan *et al.*, "LogLLM: Log-based Anomaly Detection Using Large Language Models," Apr. 2025, arXiv:2411.08561 [cs]. [Online]. Available: http://arxiv.org/abs/2411.08561

[39] H. Guo *et al.*, "LogFormer: A Pre-train and Tuning Pipeline for Log Anomaly Detection," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 1, pp. 135–143, Mar. 2024. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/27764

[40] Y. Liu *et al.*, "LogPrompt: Prompt Engineering Towards Zero-Shot and Interpretable Log Analysis," in *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*. Lisbon Portugal: ACM, Apr. 2024, pp. 364–365. [Online]. Available: https://dl.acm.org/doi/10.1145/3639478.3643108

[41] E. Karlsen *et al.*, "Benchmarking Large Language Models for Log Analysis, Security, and Interpretation," *Journal of Network and Systems Management*, vol. 32, no. 3, p. 59, Jul. 2024. [Online]. Available: https://link.springer.com/10.1007/s10922-024-09831-x

[42] M. Spiliopoulou, "The laborious way from data mining to web log mining," *Computer Systems Science and Engineering*, vol. 14, no. 2, pp. 113–126, 1999.

[43] S. Valsamidis *et al.*, "A clustering methodology of web log data for learning management systems," *Journal of Educational Technology & Society*, vol. 15, no. 2, pp. 154–167, 2012.

[44] E. Khabiri *et al.*, "Industry specific word embedding and its application in log classification," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, ser. CIKM '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 2713–2721. [Online]. Available: https://doi.org/10.1145/3357384.3357827

[45] J. M. Luna, P. Fournier-Viger, and S. Ventura, "Frequent itemset mining: A 25 years review," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 9, no. 6, p. e1329, 2019.

[46] R. Vaarandi, "Mining event logs with slct and loghound," in *NOMS 2008 - 2008 IEEE Network Operations and Management Symposium*, 2008, pp. 1071–1074.

[47] A. A. O. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "Clustering event logs using iterative partitioning," in *Proceedings of the 15th ACM SIGKDD*, 2009, pp. 1255–1264.

[48] S. Kobayashi, K. Fukuda, and H. Esaki, "Towards an nlp-based log template generation algorithm for system log analysis," in *Proceedings of The Ninth International Conference on Future Internet Technologies*, ser. CFI '14. New York, NY, USA: Association for Computing Machinery, 2014. [Online]. Available: https://doi.org/10.1145/2619287.2619290

[49] S. Thaler, V. Menkonvski, and M. Petkovic, "Towards a neural language model for signature extraction from forensic logs," in *2017 5th International Symposium on Digital Forensic and Security (ISDFS)*, 2017, pp. 1–6.

[50] C. C. Aggarwal and C. K. Reddy, "Data clustering," *Algorithms and applications. Chapman&Hall/CRC Data mining and Knowledge Discovery series, Londra*, 2014.

[51] R. Vaarandi, "A data clustering algorithm for mining patterns from event logs," in *Proc. 3rd IEEE Workshop on IP Operations and Management*, 2003, pp. 119–126.

[52] M. Nagappan and M. A. Vouk, "Abstracting log lines to log event types for mining software system logs," in *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, 2010, pp. 114–117.

[53] P. He *et al.*, "Towards automated log parsing for large-scale log data analysis," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 6, pp. 931–944, 2017.

[54] L. Tang, T. Li, and C.-S. Perng, "Logsig: Generating system events from raw textual logs," in *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, ser. CIKM '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 785–794. [Online]. Available: https://doi.org/10.1145/2063576.2063690

[55] Q. Fu *et al.*, "Execution anomaly detection in distributed systems through unstructured log analysis," *Proceedings - IEEE International Conference on Data Mining, ICDM*, pp. 149–158, 2009.

[56] X. Ning *et al.*, "1hlaer: a system for heterogeneous log analysis," in *SDM Workshop on Heterogeneous Learning*, 2014.

[57] A. Shadrova, "Topic models do not model topics: epistemological remarks and steps towards best practices," *Journal of Data Mining & Digital Humanities*, vol. 2021, 2021.

[58] R. Churchill and L. Singh, "The evolution of topic modeling," *ACM Comput. Surv.*, vol. 54, no. 10s, Nov. 2022. [Online]. Available: https://doi.org/10.1145/3507900

[59] T. Hofmann, "Probabilistic latent semantic indexing," in *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, 1999, pp. 50–57.

[60] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[61] M. R. Grootendorst, "Bertopic: Neural topic modeling with a class-based tf-idf procedure," *ArXiv*, vol. abs/2203.05794, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:247411231

[62] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. [Online]. Available: https://arxiv.org/abs/1908.10084

[63] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," 2018. [Online]. Available: https://arxiv.org/abs/1802.03426

[64] R. J. Campello, D. Moulavi, and J. Sander, "Density-based clustering based on hierarchical density estimates," in *Pacific-Asia conference on knowledge discovery and data mining*. Springer, 2013, pp. 160–172.

[65] A. Vaswani *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[66] J. Alammar. (2018) The illustrated transformer. [Online]. Available: https://jalammar.github.io/illustrated-transformer/

[67] Q. Fournier, M. Caron, and D. Aloise, "A practical survey on faster and lighter transformers," *ACM Computing Surveys*, 2023, in press.

[68] W. X. Zhao *et al.*, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, vol. 1, no. 2, 2023.

[69] J. Zhang *et al.*, "Computer multimedia assisted language and literature teaching using heuristic hidden markov model and statistical language model," *Computers & Electrical Engineering*, vol. 98, p. 107715, 2022.

[70] P. F. Brown *et al.*, "Class-based n-gram models of natural language," *Computational linguistics*, vol. 18, no. 4, pp. 467–480, 1992.

[71] R. Rosenfeld, "Incorporating linguistic structure into statistical language models," *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 358, no. 1769, pp. 1311–1324, 2000.

[72] Y. Bengio *et al.*, "A neural probabilistic language model," *Journal of machine learning research*, vol. 3, no. Feb, pp. 1137–1155, 2003.

[73] T. Mikolov, Q. V. Le, and I. Sutskever, "Exploiting similarities among languages for machine translation," *arXiv preprint arXiv:1309.4168*, 2013.

[74] M. E. Peters *et al.*, "Deep contextualized word representations," 2018. [Online]. Available: https://arxiv.org/abs/1802.05365

[75] J. Devlin *et al.*, "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)*, 2019, pp. 4171–4186.

[76] A. Radford *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[77] J. Kaplan *et al.*, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.

[78] X. Wang *et al.*, "Self-consistency improves chain of thought reasoning in language models," *arXiv preprint arXiv:2203.11171*, 2022.

[79] J. Wei *et al.*, "Emergent abilities of large language models," *arXiv preprint arXiv:2206.07682*, 2022.

[80] G. Xu *et al.*, "Causality learning: a new perspective for interpretable machine learning," *arXiv preprint arXiv:2006.16789*, 2020.

[81] M. Kalisch and P. Bühlmann, "Causal structure learning and inference: A selective review," *Quality Technology and Quantitative Management*, vol. 11, no. 1, pp. 3–21, 2014.

[82] J. Pearl *et al.*, "Models, reasoning and inference," *Cambridge, UK: CambridgeUniversityPress*, vol. 19, no. 2, 2000.

[83] C. Squires and C. Uhler, "Causal Structure Learning: a Combinatorial Perspective," 2022. [Online]. Available: http://arxiv.org/abs/2206.01152

[84] C. Heinze-Deml, M. H. Maathuis, and N. Meinshausen, "Causal Structure Learning," *Annual Review of Statistics and Its Application*, vol. 5, pp. 371–391, 2018.

[85] N. Weinberger, "Faithfulness, coordination and causal coincidences," *Erkenntnis*, vol. 83, no. 2, pp. 113–133, 2018.

[86] C. Zhang *et al.*, "Understanding causality with large language models: Feasibility and opportunities," 2023. [Online]. Available: https://arxiv.org/abs/2304.05524

[87] M. Zečević *et al.*, "Causal parrots: Large language models may talk causality but are not causal," 2023. [Online]. Available: https://arxiv.org/abs/2308.13067

[88] T. Feng *et al.*, "On the reliability of large language models for causal discovery," in *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vienna, Austria: Association for Computational Linguistics, Jul 2025, pp. 9565–9590. [Online]. Available: https://aclanthology.org/2025.acl-long.471/

[89] Z. Jin *et al.*, "Cladder: Assessing causal reasoning in language models," in *Advances in Neural Information Processing Systems*, vol. 36. Curran Associates, Inc., 2023, pp. 31 038–31 065. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/631bb9434d718ea309af82566347d607-Paper-Conference.pdf

[90] Z. Wang, "CausalBench: A comprehensive benchmark for evaluating causal reasoning capabilities of large language models," in *Proceedings of the 10th SIGHAN Workshop on Chinese Language Processing (SIGHAN-10)*. Bangkok, Thailand: Association for Computational Linguistics, Aug 2024, pp. 143–151. [Online]. Available: https://aclanthology.org/2024.sighan-1.17/

[91] H. Chi *et al.*, "Unveiling causal reasoning in large language models: Reality or mirage?" in *Advances in Neural Information Processing Systems*, vol. 37. Curran Associates, Inc., 2024, pp. 96 640–96 670. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2024/file/af2bb2b2280d36f8842e440b4e275152-Paper-Conference.pdf

[92] E. Kıcıman *et al.*, "Causal reasoning and large language models: Opening a new frontier for causality," 2024. [Online]. Available: https://arxiv.org/abs/2305.00050

[93] T. Ban *et al.*, "Integrating large language model for improved causal discovery," 2025. [Online]. Available: https://arxiv.org/abs/2306.16902

[94] M. Takayama *et al.*, "Integrating large language models in causal discovery: A statistical causal approach," 2025. [Online]. Available: https://arxiv.org/abs/2402.01454

[95] H. Jiang *et al.*, "Llm4causal: Democratized causal tools for everyone via large language model," 2024. [Online]. Available: https://arxiv.org/abs/2312.17122

[96] K. Han *et al.*, "Causal agent based on large language model," 2025. [Online]. Available: https://arxiv.org/abs/2408.06849

[97] X. Liu *et al.*, "Large language models and causal inference in collaboration: A comprehensive survey," in *Findings of the Association for Computational Linguistics: NAACL 2025*. Albuquerque, New Mexico: Association for Computational Linguistics, Apr 2025, pp. 7668–7684. [Online]. Available: https://aclanthology.org/2025.findings-naacl.427/

[98] J. Ma, "Causal inference with large language model: A survey," in *Findings of the Association for Computational Linguistics: NAACL 2025*. Albuquerque, New

Mexico: Association for Computational Linguistics, Apr 2025, pp. 5886–5898. [Online]. Available: https://aclanthology.org/2025.findings-naacl.327/

[99] R. Guo *et al.*, "A survey of learning causality with data: Problems and methods," *ACM Comput. Surv.*, vol. 53, no. 4, jul 2020. [Online]. Available: https://doi.org/10.1145/3397269

[100] D. M. Chickering, "Learning equivalence classes of bayesian-network structures," *The Journal of Machine Learning Research*, vol. 2, pp. 445–498, 2002.

[101] I. Tsamardinos, L. E. Brown, and C. F. Aliferis, "The max-min hill-climbing bayesian network structure learning algorithm," *Machine learning*, vol. 65, no. 1, pp. 31–78, 2006.

[102] L. Klem, "Structural equation modeling." 2000.

[103] S. Shimizu *et al.*, "A linear non-gaussian acyclic model for causal discovery." *Journal of Machine Learning Research*, vol. 7, no. 10, 2006.

[104] D. Rothenhäusler *et al.*, "Backshift: Learning causal cyclic graphs from unknown shift interventions," *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[105] Y. Gao, L. Shen, and S. T. Xia, "DAg-GAn: Causal structure learning with generative adversarial nets," *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, vol. 2021-June, pp. 3320–3324, 2021.

[106] I. Goodfellow *et al.*, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[107] C. Xu and W. Xu, "Causal structure learning with one-dimensional convolutional neural networks," *IEEE Access*, vol. 9, pp. 162 147–162 155, 2021.

[108] Y. Yu *et al.*, "DAG-GNN: DAG structure learning with graph neural networks," *36th International Conference on Machine Learning, ICML 2019*, vol. 2019-June, pp. 12 395–12 406, 2019.

[109] I. Ng *et al.*, "A Graph Autoencoder Approach to Causal Structure Learning," 2019. [Online]. Available: http://arxiv.org/abs/1911.07420

[110] Y. Yu *et al.*, "DAGs with No Curl: An Efficient DAG Structure Learning Approach," in *Proceedings of the 38th International Conference on Machine Learning*, 2021. [Online]. Available: http://arxiv.org/abs/2106.07197

[111] K. Yu *et al.*, "Multilabel feature selection: A local causal structure learning approach," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–14, 2021.

[112] P. Mudgal and R. Wouhaybi, "An assessment of chatgpt on log data," 2023. [Online]. Available: https://arxiv.org/abs/2309.07938

[113] R. Hermawan *et al.*, "Benchmarking large language models for root cause analysis in train control software testing," in *2025 ACM/IEEE 28th International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, 2025, pp. 648–657.

[114] J. Xu *et al.*, "Openrca: Can large language models locate the root cause of software failures?" in *The Thirteenth International Conference on Learning Representations*, 2025. [Online]. Available: https://openreview.net/forum?id=M4qNIzQYpd

[115] Y. Tian, S. Ying, and T. Li, "Causallog: Log parsing using llms with causal intervention for bias mitigation," *Information Processing Management*, vol. 63, no. 4, p. 104609, 2026. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306457326000014

[116] P. Gupta *et al.*, "Scalable and efficient large-scale log analysis with llms: An it software support case study," 2025. [Online]. Available: https://arxiv.org/abs/2511.14803

[117] V. Bertalan and D. Aloise, "Using transformer models and textual analysis for log parsing," in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2023, pp. 367–378.

[118] S. He *et al.*, "Loghub: A large collection of system log datasets towards automated log analytics," 2020, online.

[119] J. Rand and A. Miranskyy, "On automatic parsing of log records," in *Proceedings - International Conference on Software Engineering*, 2021, pp. 41–45.

[120] S. Nedelkoski *et al.*, "Self-supervised log parsing," in *Lecture Notes in Computer Science (LNCS)*, vol. 12460, 2021, pp. 122–138.

[121] J. Zhu *et al.*, "Tools and benchmarks for automated log parsing," in *Proc. 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice*, 2019, pp. 121–130.

[122] L. Layer *et al.*, "Automatic log analysis with nlp for the cms workflow handling," *EPJ Web of Conferences*, vol. 245, p. 03006, 2020.

[123] N. Aussel, Y. Petetin, and S. Chabridon, "Improving performances of log mining for anomaly prediction through nlp-based log parsing," in *Proc. 26th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 2018, pp. 237–243.

[124] H. Guo, S. Yuan, and X. Wu, "Logbert: Log anomaly detection via bert," in *Proceedings of the International Joint Conference on Neural Networks*, 2021.

[125] Y. Tay *et al.*, "Charformer: Fast character transformers via gradient-based subword tokenization," Feb. 2022, arXiv:2106.12672.

[126] S. Serasinghe, H. Shen, and D. Chen, "Ilse: An intelligent web-based system for log structuring and extraction," in *Proc. Asia-Pacific Software Engineering Conference (APSEC)*, 2018, pp. 588–593.

[127] H. Lou and F. F. Hassanzadeh, "Asymptotic analysis of data deduplication with a constant number of substitutions," in *2021 IEEE International Symposium on Information Theory (ISIT)*, 2021, pp. 3296–3301.

[128] S. Lupton *et al.*, "Literature review on log anomaly detection approaches utilizing online parsing methodology," in *Proc. Asia-Pacific Software Engineering Conference (APSEC)*, 2021, pp. 559–563.

[129] D. El-Masri *et al.*, "A systematic literature review on automated log abstraction techniques," *Information and Software Technology*, vol. 122, p. 106276, 2020.

[130] Q. Fu *et al.*, "Execution anomaly detection in distributed systems through unstructured log analysis," in *2009 Ninth IEEE International Conference on Data Mining*, 2009, pp. 149–158.

[131] N. Wu *et al.*, "Deep transformer models for time series forecasting: The influenza prevalence case," Jan. 2020, arXiv:2001.08317.

[132] H.-T. Nguyen *et al.*, "Transformer-based approaches for legal text processing: Jnlp team - coliee 2021," *Rev Socionetwork Strat*, vol. 16, no. 1, pp. 135–155, 2022.

[133] Z. Qu *et al.*, "Dota: detect and omit weak attentions for scalable transformer acceleration," in *Proc. 27th ACM ASPLOS*, 2022, pp. 14–26.

[134] R. Powalski *et al.*, "Going full-tilt boogie on document understanding with text-image-layout transformer," July 2021, arXiv:2102.09550.

[135] W. Kwon *et al.*, "A fast post-training pruning framework for transformers," Oct. 2022, arXiv:2204.09656.

[136] K. Shima, "Length matters: Clustering system log messages using length of words," 2016, arXiv:1611.03213.

[137] H. Hamooni *et al.*, "Logmine: Fast pattern recognition for log analytics," in *Proc. 25th ACM CIKM*, 2016, pp. 1573–1582.

[138] M. Mizutani, "Incremental mining of system log format," in *2013 IEEE International Conference on Services Computing*, 2013, pp. 595–602.

[139] M. Du and F. Li, "Spell: Online streaming parsing of large unstructured system logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 11, pp. 2213–2227, 2019.

[140] Y. Liu *et al.*, "Roberta: A robustly optimized bert pretraining approach," July 2019, arXiv:1907.11692.

[141] S. Biswas and H. Rajan, "Fair preprocessing: towards understanding compositional fairness of data transformers in machine learning pipeline," in *Proc. 29th ACM ESEC/FSE*, 2021, pp. 981–993.

[142] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *Journal of Machine Learning Research*, vol. 9, no. 11, 2008.

[143] X. Li *et al.*, "Swisslog: Robust and unified deep learning based log anomaly detection for diverse faults," in *Proc. International Symposium on Software Reliability Engineering (ISSRE)*, 2020, pp. 92–103.

[144] M. Raynal, M.-O. Buob, and G. Quenot, "A novel pattern-based edit distance for automatic log parsing: Implementation and reproducibility notes," 2020.

[145] Z. A. Khan *et al.*, "Guidelines for assessing the accuracy of log message template identification techniques," in *Proceedings of the 44th ICSE*, 2022, pp. 1095–1106.

[146] R. Vaarandi and M. Pihelgas, "Logcluster - a data clustering and pattern mining algorithm for event logs," in *2015 11th CNSM*, 2015, pp. 1–7.

[147] X. Wang *et al.*, "Ltmatch: A method to abstract pattern from unstructured log," *Applied Sciences (Switzerland)*, vol. 11, pp. 1–15, 2021.

[148] Z. Chunyong and X. Meng, "Log parser with one-to-one markup," in *Proc. 3rd ICICT 2020*, 2020, pp. 251–257.

[149] T. Xiao *et al.*, "Lpv: A log parser based on vectorization for offline and online log parsing," in *Proc. IEEE International Conference on Data Mining (ICDM)*, 2020, pp. 1346–1351.

[150] A. Vervaet, R. Chiky, and M. Callau-Zori, "Ustep: Unfixed search tree for efficient log parsing," in *Proc. IEEE International Conference on Data Mining (ICDM)*, 2021, pp. 659–668.

[151] S. Chen and H. Liao, "Bert-log: Anomaly detection for system logs based on pre-trained language model," *Applied Artificial Intelligence*, vol. 36, no. 1, p. 2145642, 2022.

[152] Z. Zhao *et al.*, "Trine: Syslog anomaly detection with three transformer encoders in one generative adversarial network," *Applied Intelligence*, vol. 52, no. 8, pp. 8810–8819, 2022.

[153] S. Huang *et al.*, "Hitanomaly: Hierarchical transformers for anomaly detection in system log," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2064–2076, 2020.

[154] H. Guo *et al.*, "Translog: A unified transformer-based framework for log anomaly detection," 2021, arXiv:2201.00016.

[155] S. Tao *et al.*, "Logstamp: Automatic online log parsing based on sequence labelling," *SIGMETRICS Perform. Eval. Rev.*, vol. 49, no. 4, pp. 93–98, 2022.

[156] B. Deokar and A. Hazarnis, "Intrusion detection system using log files and reinforcement learning," *International Journal of Computer Applications*, vol. 45, no. 19, pp. 28–35, 2012.

[157] B. Hartmann *et al.*, "What would other programmers do: suggesting solutions to error messages," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 1019–1028. [Online]. Available: https://doi.org/10.1145/1753326.1753478

[158] G. Tassey, "The economic impacts of inadequate infrastructure for software testing," 2002.

[159] S. W. Thomas, "Mining software repositories using topic models," in *2011 33rd International Conference on Software Engineering (ICSE)*, 2011, pp. 1138–1139.

[160] T. Wang and Y. Liu, "Jsea: A program comprehension tool adopting lda-based topic modeling," *International Journal of Advanced Computer Science and Applications*, vol. 8, no. 3, 2017.

[161] A. T. Nguyen *et al.*, "Duplicate bug report detection with a combination of information retrieval and topic modeling," in *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, 2012, pp. 70–79.

[162] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor, "Software traceability with topic modeling," in *2010 ACM/IEEE 32nd International Conference on Software Engineering*, vol. 1, 2010, pp. 95–104.

[163] M. Menenberg *et al.*, "Topic modeling for management sciences: A network-based approach," in *2016 IEEE International Conference on Big Data (Big Data)*, 2016, pp. 3509–3518.

[164] P. Xie and E. P. Xing, "Integrating document clustering and topic modeling," in *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*, ser. UAI'13. Arlington, Virginia, USA: AUAI Press, 2013, p. 694–703.

[165] L. George and P. Sumathy, "An integrated clustering and bert framework for improved topic modeling," *International Journal of Information Technology*, vol. 15, no. 4, pp. 2187–2195, 2023.

[166] J. Devlin *et al.*, "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)*. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. [Online]. Available: https://aclanthology.org/N19-1423

[167] W. Meng *et al.*, "Logsummary: Unstructured log summarization for software systems," *IEEE Transactions on Network and Service Management*, vol. 20, no. 3, pp. 3803–3815, 2023.

[168] J. Zhu *et al.*, " Loghub: A Large Collection of System Log Datasets for AI-driven Log Analytics ," in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2023, pp. 355–366. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ISSRE59848.2023.00071

[169] S. Locke *et al.*, "Logassist: Assisting log analysis through log summarization," *IEEE Transactions on Software Engineering*, vol. 48, no. 9, pp. 3227–3241, 2022.

[170] R. Dijkman and A. Wilbik, "Linguistic summarization of event logs – a practical approach," *Information Systems*, vol. 67, pp. 114–125, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306437916303192

[171] W. Shang *et al.*, "Understanding log lines using development knowledge," in *2014 IEEE International Conference on Software Maintenance and Evolution*, 2014, pp. 21–30.

[172] M. Stojanovic, "Sas® log summarizer–finding what's most important in the sas® log," 2008.

[173] D. Gunter *et al.*, "Log summarization and anomaly detection for troubleshooting distributed systems," in *2007 8th IEEE/ACM International Conference on Grid Computing*, 2007, pp. 226–234.

[174] A. Hamou-Lhadj and T. Lethbridge, "Summarizing the content of large traces to facilitate the understanding of the behaviour of a software system," in *14th IEEE International Conference on Program Comprehension (ICPC'06)*, 2006, pp. 181–190.

[175] S. Nedelkoski *et al.*, "Self-supervised Log Parsing," *Lecture Notes in Computer Science*, vol. 12460 LNAI, pp. 122–138, 2021.

[176] E. Gentili, A. Milani, and V. Poggioni, "Data summarization model for user action log files," in *Computational Science and Its Applications – ICCSA 2012*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 539–549.

[177] H. Kato, H. Hiraishi, and F. Mizoguchi, "Log summarizing agent for web access data using data mining techniques," in *Proceedings Joint 9th IFSA World Congress and 20th NAFIPS International Conference*, 2001, pp. 2642–2647 vol.5.

[178] I. Siddhartha, H. Zhan, and V. S. Sheng, "Abstractive text summarization via stacked lstm," in *2021 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2021, pp. 437–442.

[179] P. Mudgal and R. Wouhaybi, "An assessment of chatgpt on log data," in *AI-generated Content*. Singapore: Springer Nature Singapore, 2024, pp. 148–169.

[180] X. Li and L. Lei, "A bibliometric analysis of topic modelling studies (2000–2017)," *Journal of Information Science*, vol. 47, no. 2, pp. 161–175, 2021.

[181] C. C. Silva, M. Galster, and F. Gilson, "Topic modeling in software engineering research," *Empirical Software Engineering*, vol. 26, no. 6, p. 120, 2021.

[182] P. W. McBurney *et al.*, "Improving topic model source code summarization," in *Proceedings of the 22nd International Conference on Program Comprehension*, ser. ICPC 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 291–294. [Online]. Available: https://doi.org/10.1145/2597008.2597793

[183] J. Hu *et al.*, "Modeling the evolution of development topics using dynamic topic models," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2015, pp. 3–12.

[184] B. Dit *et al.*, "Feature location in source code: a taxonomy and survey," *Journal of software: Evolution and Process*, vol. 25, no. 1, pp. 53–95, 2013.

[185] H. Hemmati *et al.*, "Prioritizing manual test cases in rapid release environments," *Software Testing, Verification and Reliability*, vol. 27, no. 6, p. e1609, 2017.

[186] A. Hindle *et al.*, "Relating requirements to implementation via topic analysis: Do topics extracted from requirements make sense to managers and developers?" in *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, 2012, pp. 243–252.

[187] J. Garcia *et al.*, "Enhancing architectural recovery using concerns," in *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, 2011, pp. 552–555.

[188] Z. S. Li, "Leveraging user feedback for requirements through trend and narrative analysis," in *2023 IEEE 31st International Requirements Engineering Conference (RE)*, 2023, pp. 386–390.

[189] A. Naghshzan and S. Ratte, "Enhancing api documentation through bertopic modeling and summarization," 2023. [Online]. Available: https://arxiv.org/abs/2308.09070

[190] T. Diamantopoulos, D.-N. Nastos, and A. Symeonidis, "Semantically-enriched jira issue tracking data," in *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*, 2023, pp. 218–222.

[191] L. Tao *et al.*, "Code librarian: A software package recommendation system," in *Proceedings of the 45th International Conference on Software Engineering: Software Engineering in Practice*, ser. ICSE-SEIP '23. IEEE Press, 2023, p. 196–198. [Online]. Available: https://doi.org/10.1109/ICSE-SEIP58684.2023.00023

[192] G. Chu *et al.*, "Lognotion: Highlighting massive logs to assist human reading and decision making," *IEEE Transactions on Services Computing*, vol. 18, no. 2, pp. 940–953, 2025.

[193] Y. Lu, *LogSage: Log Summarization Assistant with Guided Enhancement*. New York, NY, USA: Association for Computing Machinery, 2025, p. 1979–1981. [Online]. Available: https://doi.org/10.1145/3672608.3707990

[194] P. Balasubramanian, J. Seby, and P. Kostakos, "Cygent: A cybersecurity conversational agent with log summarization powered by gpt-3," in *2024 3rd International Conference on Artificial Intelligence For Internet of Things (AIIoT)*, 2024, pp. 1–6.

[195] R. Katukam, "Ai-driven log summarization for security operations centers: A web-based approach using gemini api," *International Journal of Emerging Research in Engineering and Technology*, vol. 6, no. 3, p. 136–145, Sep. 2025. [Online]. Available: https://ijeret.org/index.php/ijeret/article/view/305

[196] P. Mudgal, "Reflex: Reference-free evaluation of log summarization via large language model judgment," 2025. [Online]. Available: https://arxiv.org/abs/2511.07458

[197] T. Cui *et al.*, "Logeval: A comprehensive benchmark suite for llms in log analysis," *Empirical Software Engineering*, vol. 30, no. 6, p. 173, 2025.

[198] V. Bertalan and D. Aloise, "Clustering textual features for log summarization in large software systems," *Authorea Preprints*, 2025. [Online]. Available: https://doi.org/10.22541/au.173865299.90947187/v1

[199] A. Rajaraman and J. D. Ullman, *Mining of massive datasets*. Autoedicion, 2011.

[200] A. K. Jain and R. C. Dubes, *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.

[201] L. Kaufman, "Partitioning around medoids (program pam)," *Finding groups in data*, vol. 344, pp. 68–125, 1990.

[202] D. Steinley, "K-medoids and other criteria for crisp clustering," *Handbook of Cluster Analysis*, pp. 55–66, 2015.

[203] L. Kaufman and P. J. Rousseeuw, *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, 2009.

[204] R. J. G. B. Campello, D. Moulavi, and J. Sander, "Density-based clustering based on hierarchical density estimates," in *Advances in Knowledge Discovery and Data Mining*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 160–172.

[205] H. P. Luhn, "The automatic creation of literature abstracts," *IBM Journal of research and development*, vol. 2, no. 2, pp. 159–165, 1958.

[206] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 74–81. [Online]. Available: https://aclanthology.org/W04-1013

[207] G. Erkan and D. R. Radev, "Lexrank: Graph-based lexical centrality as salience in text summarization," *Journal of artificial intelligence research*, vol. 22, pp. 457–479, 2004.

[208] R. Mihalcea and P. Tarau, "TextRank: Bringing order into text," in *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*. Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 404–411. [Online]. Available: https://aclanthology.org/W04-3252/

[209] J. Steinberger, K. Jezek *et al.*, "Using latent semantic analysis in text summarization and summary evaluation," *Proc. ISIM*, vol. 4, no. 93-100, p. 8, 2004.

[210] R. Randel *et al.*, "On the k-medoids model for semi-supervised clustering," in *Variable Neighborhood Search*, A. Sifaleras, S. Salhi, and J. Brimberg, Eds. Cham: Springer International Publishing, 2019, pp. 13–27.

[211] P. Chadbourne and N. U. Eisty, "Applications of causality and causal inference in software engineering," in *2023 IEEE/ACIS 21st International Conference on Software Engineering Research, Management and Applications (SERA)*, 2023, pp. 47–52.

[212] Q. Wang *et al.*, "Detecting causal structure on cloud application microservices using granger causality models," in *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, 2021, pp. 558–565.

[213] P. Zheng *et al.*, "Granger causality-aware prediction and diagnosis of software degradation," in *2014 IEEE International Conference on Services Computing*, 2014, pp. 528–535.

[214] S. Kobayashi *et al.*, "Mining causality of network events in log data," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 53–67, 2018.

[215] R. Jarry, S. Kobayashi, and K. Fukuda, "A quantitative causal analysis for network log data," in *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*, 2021, pp. 1437–1442.

[216] F. Neves *et al.*, "Horus: Non-intrusive causal analysis of distributed systems logs," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2021, pp. 212–223.

[217] Z. Ren *et al.*, "Root cause localization for unreproducible builds via causality analysis over system call tracing," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2019, pp. 527–538.

[218] C. Xosanavongsa, E. Totel, and O. Bettan, "Discovering correlations: A formal definition of causal dependency among heterogeneous events," in *2019 IEEE European Symposium on Security and Privacy (EuroSP)*, 2019, pp. 340–355.

[219] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," in *Concurrency: the Works of Leslie Lamport*, 2019, pp. 179–196.

[220] B. D'ausbourg, "Implementing secure dependencies over a network by designing a distributed security subsystem," in *European Symposium on Research in Computer Security*. Springer, 1994, pp. 247–266.

[221] M. Markakis *et al.*, "Sawmill: From logs to causal diagnosis of large systems," in *Companion of the 2024 International Conference on Management of Data (SIGMOD '24)*. New York, NY, USA: Association for Computing Machinery, 2024, pp. 444–447.

[222] A. G. Clark *et al.*, "Testing causality in scientific modelling software," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 1, 2023.