# Automatic Construction and Updating of Knowledge Base From Log data

*Ruiqi Zhu*


Doctor of Philosophy

Artificial Intelligence Applications Institute

School of Informatics

The University of Edinburgh

2024

# Lay summary

Large software systems can be really complicated, especially when using a design called Microservice Architecture. This design means that there are many small parts interacting with each other, which adds to the complexity. Traditional methods to detect issues in these systems can work, but they often need a lot of computing power and aren't easy to understand. On the other hand, knowledge-based methods are effective, easy to understand, and explainable, but they rely on having a good knowledge base, which is hard to create and maintain. I developed a framework called TREAT, which automatically creates and updates a knowledge base from system logs, which record what happens during system runtime. This knowledge base reflects the current state of the software system and helps with many operation and maintanence tasks. In the thesis, I showed a case study of how TREAT can be used in the task of locating faulty components. TREAT is the first framework designed to continually update a knowledge base from logs to show the system's internal changes accurately. To assess the quality of the extracted knowledge, I developed a new method called LP-Measure, which directly assesses the quality of a knowledge base by looking at how robust and redundant the knowledge graph is, without extra human examination. Another application of the knowledge base is called knowledge graph embedding. I conducted an in-depth study of probability calibration of knowledge graph embeddings, which confirmed that these probabilities are often uncalibrated, explored how to select the best method to calibrated the probabilities.

In summary, the TREAT framework automates the creation and updating of a knowledge base from software logs, making it easier to understand and maintain complex software systems. My research also developed new methods to evaluate and improve the quality of these knowledge bases, ensuring they are reliable and accurate for tasks like finding faults.

# Abstract

Large software systems can be very complex, and they get more and more complex in the recently popular Microservice Architecture due to increasing interactions among more components in bigger systems. Plain model-based and data-driven diagnosis approaches can be used for fault detection, but they are usually opaque and demand massive computing power. On the other hand, knowledge-based methods have shown to be not only effective but explainable and human-friendly for various tasks such as Fault Analysis [35, 25], but are dependent on having a knowledge base. The construction and maintenance of knowledge bases are not a trivial problem, which is referred to as the knowledge bottleneck.

Software system logs are the primary and most available, sometimes the only available data that record system runtime information, which are critical for software system Operation and Maintenance (O&M). I proposed the TREAT framework, which can automate the construction and update a knowledge base from a continual stream of logs, which aims to, as faithfully as possible, reflect the latest states of the assisted software system, and facilitate downstream tasks, typically fault localisation (FL). To the best of our knowledge, this is the first effort to construct a fully automated ever-updating knowledge base from logs that aims at reflecting the internal changing states of a software system. To evaluate the TREAT framework, I devised a knowledge-based solution involving logic programming and inductive logic programming that makes use of a TREAT-powered knowledge base to fault localisation and conducted empirical experiments of this solution on a real-life 5G network test bed system.

Since evaluating the TREAT framework by fault localisation is indirect and involves many confounding factors, e.g., the specific solution to fault localisation, I explored and came up with a novel method called LP-Measure that can directly assess the quality of a given knowledge base, in particular the robustness and redundancy of a knowledge graph.

Besides, it was observed that although the extracted knowledge is of high quality in general, there are also errors in the knowledge extraction process. I surveyed the way to quantify the uncertainty during the knowledge extraction process and assign probabilities of correct extraction to every piece of knowledge, which led to a deep investigation into probability calibration and knowledge graph embeddings, specifically testing and confirming the phenomenon of uncalibrated probabilities in knowledge graph embeddings and how to choose specific calibration models from the existing toolbox.

# Acknowledgements

Pursuing a PhD is painstaking, especially when it comes across a global pandemic, but the city and the University of Edinburgh made my journey such a nice experience that I could never have imagined. First and foremost, I shall express huge appreciation to my primary supervisor, Alan Bundy, as well as my co-supervisors, Kwabena Nuamah and Jeff Pan. The numerous hours we spent in the past years taught me how to do real research, from exploring state-of-the-art and brainstorming ideas, to formalising precise hypothesis and iteratively evaluating and adapting the hypothesis. Moreover, they offered me freedom and encouragement to explore my ideas and strong backing to pass through frustration. It is my privilege to be their student.

Apart from my supervisors, I received a variety of help from the university cohort. I shall thank my close collaborators, Xue Li and Sylvia Wang, who discussed a lot with me and provided me with valuable feedback for my drafts. I need to say thank you to Vaishke Belle and N. Siddharth for reviewing my annual reports and giving suggestions. I am also very thankful to Antonio Vergari, who set up the reading group on neuro-symbolic reasoning, and Pasquale Minervini for allowing me to serve as a teaching assistant in his class and co-instructed with Jeff. Another thank you to the AIAI cohort. I got various ideas from the seminars, while the leisure events relaxed my nerves from intensive thinking and anxiety. What's more, I shall thank my examiners Dave Robertson and Lucas Dixon, who have read my thesis carefully and provided me with constructive feedback.

Very much appreciate Lei Xu and Stefano Mauceri from Huawei Ireland Research Centre, who led the project that provided funding for my PhD (under grant CIENG4721/LSC). Besides, they often joined our regular meetings and gave useful feedback from their perspective.

A massive thank you to my internship collaborators from Huawei Edinburgh Research Centre, Jeff Pan, Pavlos Vougiouklis, Yuanyi Ji, Wendi Zhou, Victor Basulto, Tianyang Liu, Teddy Liu, Sabastian Montella, and Damien Graux. During my work with them, I acquired industrial research and development experience, which will definitely benefit my career.

Should I continue, the list will become endless, as I did receive various support from all kinds of people.

Finally and most of all, I owe a great debt of gratitude to my family, particularly my dad Huafeng Zhu, my mom Liyun Kuang, and my grandma Yueyi Tan. Without their firm support, my story would have never begun.

*To my beloved family*

# Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Ruiqi Zhu)*

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

In this thesis, I focused on the problem of knowledge base (KBs) construction and updating it from log data for Operation & Maintenance (O&M) of modern software systems. O&M of modern software systems is challenging as the scale and the interactions among components within the software system are increasing rapidly.

As a consequence, it gets harder and harder for unaided human beings to operate and maintain large software systems. AI-aided O&M (AIOps) or even Automated O&M is intended to address this problem [47, 19]. There have been some commonly used techniques in automated O&M [47]. When a system failure occurs, system logs, as well as KPI (Key Performance Indicator) metrics, alarms together with other data sources are useful to help locate and resolve the failure. In some cases, system logs are the most accessible or even the only available data that record software runtime information [82] because logs record events of system changes and reveal the latest system states. Whenever an anomaly is detected and intervention is required, logs are one of the most valuable sources of information we can analyse to locate the problems. Therefore, I was considering how to make the best use of log data for typical O&M tasks. A motivating scenario and potential use case is the vision of *Autonomous Network* [134, 71, 64], which refers to a telecommunication network system that can accurately do self-operation and self-maintenance (monitoring, healing, etc) with minimal or no human intervention, and thus greatly reduce the Operation and Maintenance (O&M) cost.

One way toward this vision is having a sub-system that can be plugged into the network system to be in charge of self-monitoring and self-healing. Communication

Service Providers (CSPs) which run the network system, like Vodafone[1] and Orange[2], can accumulate data and continually improve their quality of service. However, vendors which build the network systems for CSPs, like Huawei[3] and Ericson[4], are no longer allowed to access the network system. Thus, vendors lack enough high-quality data to build data-driven AI models and push forward the Autonomous Network vision. This dilemma will not change until the business mode gets changed. Hence, it drove me to explore another branch of AI, knowledge-based methods [35]. Moreover, in the field of O&M and tasks like fault handling, it is demanded that the adopted methods provide explainability to convince human experts. This again puts knowledge-based methods over data-driven methods.

Thus, I decided to dive into the field of knowledge-based AI methods and explored solutions that can make the most use of system logs to help O&M tasks for large software systems. Specifically, I considered the feasibility and approach of extracting knowledge from a stream of system logs to construct and maintain an evolving knowledge base, which can facilitate a variety of downstream applications in the field of O&M. Accordingly, the main research hypothesis underlying this thesis is:

#### Main Hypothesis

> *It is feasible to extract knowledge from a stream of system logs, and keep track of the system changes by continually updating a knowledge base, thus supporting various downstream applications.*

I defined my main research question as how to extract knowledge from a stream of system logs, and keep track of the system changes by continually updating a knowledge base. The solution I proposed is called TREAT, a framework designed as faithfully as possible to reflect the latest state of the network system, and can support the downstream applications, typically anomaly detection (AD), fault localisation (FL), and root cause analysis (RCA) [177]. To show the effectiveness of TREAT, I conducted experiments that used the TREAT-powered knowledge base to facilitate fault localisation in a test bed system.

The main research question of knowledge base construction and updating by log data also derived 2 other questions: how to evaluate the TREAT system directly and how one can assign probabilities to triples of a Knowledge Graph (see 3.2 for more

[1] https://www.vodafone.com/
[2] https://www.orange.com/
[3] https://www.huawei.com/
[4] https://www.ericsson.com/en

details of KGs). Evaluating TREAT is a challenging task. Apart from indirectly evaluating TREAT via a downstream O&M application, meanwhile, I thought about a more generic approach, less coupled with the O&M domain. As motivated, I developed a novel approach called LP-Measure which can assess the quality of a newly obtained Knowledge Graph. Secondly, considering that uncertainties may arise during knowledge base construction and updating, I explored approaches to assign probabilities to the knowledge within the knowledge base, which led to my work on probability calibration in Knowledge Graph Embedding, i.e., calibrating the embedding scores of triples to have a reasonable argument for them to be used as probabilities. I attempted to apply these outcomes to improve the TREAT framework, but eventually, I decided not to make the integration because there was a mismatch with the requirements of TREAT. More details can be found in Chapter 7.

**Thesis Structure** In this thesis, I will first review some prior efforts related to the problem I aimed to attack in Chapter 2 and some key background information about the TREAT project in Chapter 3. Then I will present the TREAT framework, which implements our hypothesis to extract knowledge from a log stream and continually update a knowledge base in Chapter 4, as well as a case study of applying TREAT in a downstream task of fault localisation, also serving an evaluation of its effectiveness in Chapter 5. In Chapter 6, I will present a novel method called LP-Measure which evaluates the quality of a Knowledge Graph, and in Chapter 7, I will present an investigation of probability calibration at Knowledge Graph Embedding. Finally, the thesis will end with some concluding remarks in Chapter 8.

# Chapter 2

# Literature Survey

I positioned my research within the area of knowledge representation, and specifically knowledge base construction & updating. This chapter surveys some related works to show the big picture and state-of-the-art of this area. I also identified some works similar to my research but differ in some subtle aspects.

In §2.1 and §2.2, I gathered some important works in the field of knowledge representation and knowledge updating respectively. In §2.3, I mentioned some works of knowledge extraction. In §2.4, I showcased some similar works on building (and additionally updating) a knowledge base, and compared them with my research project. Though there are tons of theories and methods in representing knowledge and extracting knowledge from ordinary text data, such as books, newspapers, and dialogues, there is a huge research gap in extracting symbolic knowledge from system logs, as well as methods to organise the knowledge in a dynamically updated knowledge base.

## 2.1 Knowledge Representation

To build an intelligence engine, an important sub-task is to represent knowledge, as well as acquire, store, and make use of the knowledge, which is the core scope of the TREAT project. Knowledge representation is a key topic in artificial intelligence. This section introduces some basic concepts and methods of knowledge representation. Roughly speaking, all kinds of knowledge representation methods can be categorised into two groups: symbolic representations and sub-symbolic representations.

### 2.1.1 Symbolic Representations

**Logical Calculus** The idea of representing knowledge by symbolic logic dates back to a paper by McCarthy and Hayes [119] or even earlier. In that paper, the authors tried to use first-order logic to represent commonsense and other forms of knowledge. In particular, they proposed the Situation Calculus, which is a formalism to represent time, objects, states, actions, and effects of actions upon object states. Since then, researchers proposed various logics to better represent knowledge of different problems, and the idea of using logic to represent knowledge has become a hot topic in artificial intelligence, forming a sub-field called Knowledge Representation and Reasoning.

A problem of Situation Calculus is The Frame Problem [119], which is the problem of representing the effects of actions without explicitly representing the non-effects. For example, such an action `turn_right(ricky)`. changes the direction of the object `ricky`, but not the speed, the type, the mood, and other states of the object. McCarthy later proposed a solution called circumscription [120]. Among all the other solutions, the Event Calculus [98, 129, 188] has a long-lasting impact. It is also a logical formalism to represent knowledge about events and their effects. Notably, the Event Calculus adopted the Commonsense Law of Inertia, which states that if an object has a property at a time and no event occurs that would change the property, then the object has the property at all later times [129]. The Event Calculus also offers simpler formalism, making it easy to use.

**Description Logic** Description Logic is a family of formal knowledge representation languages. Compared with Situation Calculus and Event Calculus, Description Logic focuses on representing ontological knowledge about concepts and their relationships. It is a subset of first-order logic with a restricted but simpler form. There are several different variants of Description Logic with different levels of decidability and expressiveness [15]. The most popular one is the Web Ontology Language (OWL) [121], which is recommended by the World Wide Web Consortium [1]. However, Description Logic does not provide a formalism and mechanism to represent dynamic knowledge like events and actions.

**Logic Programming** Situation Calculus, Event Calculus, and Description Logic are mathematical or logical theories, while Logic Programming is a field of technological studies about how to write programs using logic to solve real computational problems.

[1] `https://www.w3.org/OWL/`

The most popular logic programming language is Prolog [39], a declarative language that only requires the programmer to specify what the program should do, but not how to do it. A Prolog program (as well as other logic programs) is a collection of facts and rules written in the Horn Clause format [84, 39]. The reasoning engine behind Prolog will try to find a solution to the problem by searching and reasoning whether an assertion can be deductively entailed by the given facts and rules.

### 2.1.2 Sub-symbolic Representations

**Embedding** Apart from representing knowledge by symbolic logic, researchers also considered the representations by numerical values, i.e., vectors, matrices, or more generally speaking, tensors. The simplest and most popular sub-symbolic representation is the vector representation, also known as embeddings. For example, the idea of word embeddings is to represent words by vectors, so that the semantic similarity between words can be measured by the distance between their vectors, and thus in a vector space, the words with similar meanings will assemble. Among this line of research, Word2Vec [123] and FastText [20] are the most outstanding methods for representing words. Once the word embeddings are learned, they can be used to perform various vector calculations. For example, the embedding of the word "Paris" can be found as the closest one to $vec(Berlin) - vec(Germany) + vec(France)$. Words can be represented by vectors, and so do sentences and documents [102, 13].

As the nature of linguistics, the same word may have very different meanings in different contexts. For instance, the word "bank" may refer to a kind of financial organisation in economic articles or the land alongside a river in geology literature. Instead of learning the general meanings, researchers also considered learning the concrete meanings in a specific passage context, which is called contextualised word embeddings. The most well-known contextualised embedding method is BERT [57] for words and its variants for sentences and documents [162, 4].

The idea of embedding can not only be applied to words, sentences, and documents, but also to entities and relationships in Knowledge Graphs (KGs), which is a set of $\langle subject, predicate, object \rangle$ triples. The TransE [22] algorithm is a seminal work on graph embedding to learn the embeddings of entities and relations in a knowledge graph, such that the links between a subject entity and object entity $\langle s, p, o \rangle$ can numerically establish the relation $\vec{s} + \vec{p} \approx \vec{o}$. Similar ideas also applied to ontologies, such as the Owl2Vec method [34]. The idea of graph embedding has been widely used

in knowledge graph completion and knowledge graph reasoning [201].

**Neural Network Weight Parameters** Recently researchers found that not only do the embedding vectors encode the meaning of words or entities, but the weight parameters of neural networks also encode the knowledge of a domain and thus may serve as a knowledge base. The earliest finding is that the BERT model [57] which was a probabilistic neural language model trained on a book corpus learnt some factual knowledge [149]. If we feed the BERT model with the input "`The theory of relativity was developed by ____`" and ask BERT to fill in the blank, it will provide the answer "`Einstein`" with a high likelihood. As the number of parameters in a neural network model increases, the amount of knowledge that can be encoded in the model also increases. The BERT model has only millions of weight parameters, while the GPT-3 model [23] has 175 billion and was trained on a filtered corpus of available texts from the Web, which can be used to perform various reasoning tasks, including answering factual questions. The latest model released in March of 2023, GPT-4, had passed SAT and GRE tests as well as other knowledge-intensive tasks with exceptionally high scores [143], it is still being iteratively improved. Many researchers have found that when training on a huge amount of text data, the neural network model remembered the knowledge and modelled the process of human language generation, and jointly encoded them in the model parameters. Therefore, the language model can understand human instructions and questions and give responses in natural language [166, 92, 7, 122].

**Text and Other Modality** For human beings, it is more natural to represent knowledge in texts and audio & video clips. For thousands of years, we've passed knowledge by playing games, real demonstrations, word of mouth, and more recently, books and multimedia recordings. These representations seem old-fashioned and have been ignored by the AI community for a long time because we did not discover an effective computational method to perform reasoning over these representations. However, with the development of large language models (LLMs) and later foundation models, we now find that we can throw the texts and data of other modalities into the model and ask them to do various tasks by natural language instructions. The retrieval-augmented models proposed to store the knowledge in a collection of texts and retrieve the related passages every time being instructed to do something, e.g., question answering [104, 89]. In this way, we can not only update the knowledge easily by simply

modifying the text, but also relieve the models from memorising a lot of knowledge. The same idea can apply to data of other non-text modalities, such as images [77]. At this moment, the model weights of LLMs store world knowledge and determine the reasoning power. Researchers now wish that the model weights ONLY determine the reasoning power while storing as little as possible knowledge, so that the task of updating knowledge and upgrading reasoning power can be decoupled. There is also ongoing research on distilling the reasoning capabilities from large models to smaller models so that models can get rid of the knowledge and focus on performing reasoning [173].

## 2.2 Knowledge Updating

The previous section §2.1.2 focuses on how to represent data in the computer so that we can easily use it for various tasks. However, the knowledge in the real world is not static, it is constantly changing. Updating the existing knowledge is an important task related to knowledge representation, but also a relatively independent and popular research topic. In this section, we will introduce some research on knowledge updating.

**Belief revision** In the field of symbolic representation, the subject dedicated to updating existing knowledge is called belief revision, or belief change [74], arising in the 1980s. For agents with a belief set represented by formal logical representation, belief revision studies how to revise this logical belief set $K$, by introducing or removing logical sentences $a$, so that the belief can adapt to new information [74]. For example, adding a piece of new knowledge into the knowledge base $K + a$ may cause inconsistency, and then the agent must decide whether to delete some old knowledge, or other actions. If it chooses to delete some old knowledge, then the key research question is which pieces of old knowledge should be deleted. At the core of this subject is the AGM model[8] setting postulates to which rational revision operations should follow, which is the dominant theory of belief revision.

One limitation of belief revision is that none of the work in this subject considers syntactical change[105], e.g., adding new predicate symbols or changing the arity of predicate symbols. Ontology evolution, like the GALILEO project[30], uses “repair patterns” represented in higher-order logic to tackle this limitation. It is the timely adaptation of an ontology to the arisen changes and the consistent propagation of these

changes to dependent artefacts[180]. Chan and his colleagues conducted a series of research automating ontology evolution in physical knowledge[24, 30, 103] Li took a further step[105]. She combined abduction, belief revision, and conceptual change to repair faulty theories. Given a subset of preferred sentences, the proposed ABC system can detect insufficiency and incompatibility in a logical theory, and raise repair plans. It is a pity that both Chan and Li evaluated their work using a relatively small dataset. We don't know if their methods can work in large-scale KBs.

**Continual Learning** As for numeric representations, the majority of efforts fall in substantially training an existing model with new data. The term Continual Learning(CL) first appeared in the PhD thesis published by Mark Ring in 1994[165]. Ring's continual learning agent focused on solving reinforcement learning tasks, and transferred the learned skills and knowledge to subsequent, and perhaps even more sophisticated tasks. As time went by the notion of CL got generalised: given an infinite stream of data, a CL system should learn from a sequence of partial experiences where all data is not available at once.

Similar to Continual Learning, Lifelong Machine Learning(LML) also emerged in the field of robotic research. The origin of LML dates back to 1995. Thrun and Mitchell published a paper on how a robot can learn the invariant knowledge about the environment and the robot itself[185]. They described the notion of LML as learning knowledge in $N$ tasks, and leveraged the learnt knowledge to help the $N+1$ task. Mitchell and his colleagues later in 2015 proposed a new learning paradigm called Never Ending Learning(NEL)[127], which is usually regarded as a special Lifelong Machine Learning paradigm in the field of unsupervised or semi-supervised learning. The concept was derived from the famous system, Never Ending Language Learner(NELL)[29]. There is plenty of continual learning research on computer vision models as well. Learning without Forgetting(LwF)[108], Elastic Weight Consolidation(EWC)[95], Incremental classifier and Representation Learning(iCaRL)[161], were designed originally for image classification. There had been established benchmark datasets for Continual Learning in image classification, such as Permuted MNIST[214, 17].

**Knowledge Editing for LLMs** As LLMs and foundation models show their potential towards artificial general intelligence, updating knowledge encoded in them attracts more and more interest. A key difference between LLMs and other neural networks

is that the former ones have orders of magnitude more weight parameters. Re-training the whole model whenever new data comes would be a theoretically feasible option but practically not affordable. A potentially more economical option is to update the model parameters directly. As the knowledge in neural models is jointly encoded in weight parameters, how to make desired modifications to a specific piece of knowledge with minimal influence on other knowledge is the main challenge. KNOWLEDGEEDITOR [50] is such a method that uses a hyper-network to update the parameters of the language model such that it prefers an alternative prediction for a specific input $x$ without affecting the prediction of any other input $X \backslash x$. As for GPT-like models, which are the dominant architecture of current LLMs, some researchers raised a hypothesis that factual knowledge is stored in the MLP modules at specific middle layers, and developed empirically effective methods to edit this factual knowledge [122]. Similar to the TREAT's setting, Han et al. [79] developed the RASE framework to continually update an LLM from a stream of document data. More recent surveys on advances in knowledge editing and updating for LLMs can be found in [212, 216, 202].

## 2.3 Similar Works: Knowledge Extraction

Knowledge Extraction has long been a key research topic of artificial intelligence, particularly extracting knowledge from natural language text [43, 31, 75, 198]. Though the definitions vary within the literature, the main objective of information extraction is to extract some structured information, typically entities and their relations as well as attributes, from unstructured documents.

Stanford CoreNLP[2] [117] includes an information extraction module [10] as a component of its main suites. When deep learning research emerged, deep neural networks and end-to-end methods were used to extract triples from texts, for example, the Universal Information Extraction framework [115]. Today, as recently (LLMs) [57, 157, 158, 23] achieved unprecedented success in almost all kinds of natural language processing tasks, researchers found that LLMs also demonstrated the potential to be a general information extractor [195]. Agrawa et al. [6] have shown that using few-shot learning techniques, LLMs can serve as information extractors from clinical documents. Dagdelen et al also [46] explored the method of fine-tuning LLMs to extract structured information from scientific literature.

One of the main approaches to constructing a knowledge base is to extract triples

[2] https://stanfordnlp.github.io/CoreNLP/

from a collection of text documents. For example, DBpedia is initially constructed by extracting triples from Wikipedia pages [14]. In this way, the quality of an extracted KG largely depends on the performance of the underlying information extraction system. To evaluate the performance of an information extraction system, a lot of benchmark datasets have been created, like OIE2016 [178] and LSOIE [176], which comprises pairs of text documents and corresponding gold standard triples. Table 2.1 shows some examples of the open-domain texts and gold standard triples.

| **Document** | **Triples** |
|---|---|
| AMD, which is based in U.S., is a technology company. | (AMD; is; technology company)<br>(AMD; located at; U.S.) |
| Born in Glasgow, Fisher is a graduate of the London Opera Centre. | (Fisher; born in; Glasgow)<br>(Fisher; graduated from; London Opera Centre) |

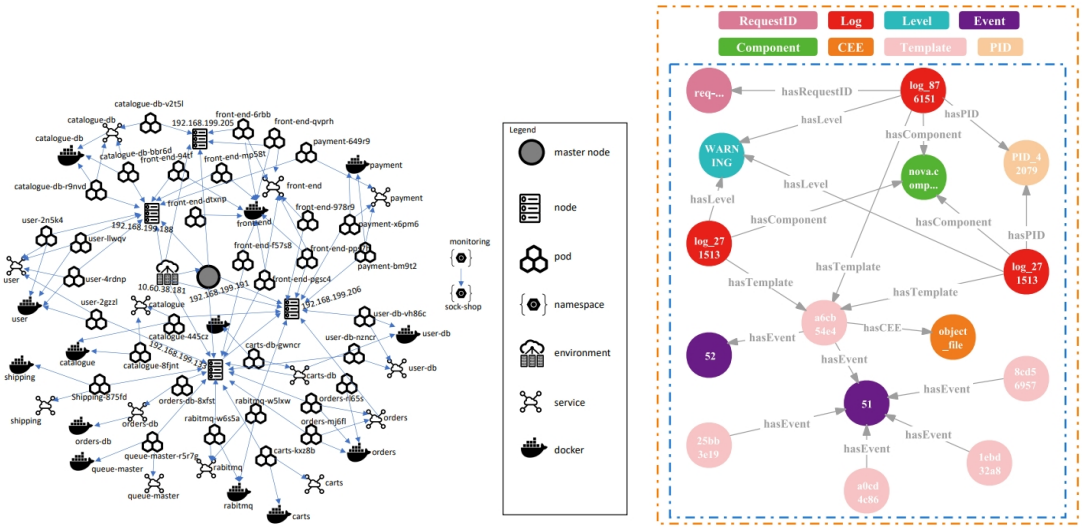
Table 2.1: Example pairs of texts and human-crafted gold standard triples in open information extraction benchmark datasets [178].

The above works are about extracting knowledge from natural language texts. Yet in the TREAT project, we need to extract knowledge from logs. We identified some papers on extracting a knowledge graph from system logs, especially in the field of AIOps (Artificial Intelligence aided Operations) [47].

For example, a paper recently published in 2021 [109] proposed to automatically build an Alarm Knowledge Graph by knowledge extraction from historical alarm and system manual alarm descriptions to perform root cause analysis for an optical network. Then, a Graph Neural Network was developed upon this alarm KG to identify which is the root cause, i.e., which alarm indicates the root cause for a detected anomaly. This research considers two information sources to build an alarm KG. One is the semi-structured alarm description in system manuals, and the other is the structured historical log data stored in relational databases. In terms of extracting knowledge from the manual, they use some template-based methods to extract knowledge from sentences like “alarm A is possibly caused by fault Z”. Moreover, extracting knowledge from tabular log records is simply an Extract-Transform-Load pipeline work. This is a significant difference from our TREAT project; the alarms they handled are already structured in a relational data format, while we aimed to handle much more general

forms of system logging, which is one of the great technical challenges.

LogKG [181] is a work closer to ours. It targets constructing a knowledge graph from raw logs and using this knowledge graph for root cause analysis. different from the one [109] before, LogKG handles "real" raw logs instead of structured, relational alarm records. Similar to LEKG [197], which is our intermediate work (§3.3), LogKG extracts a KG from logs by a pipeline of log parsing, entity recognition, and entity alignment. The LogKG project includes more efforts in obtaining vector representations for every system failure case and training models for root cause analysis, which we don't review here.




(a) Sample KG from LogKG [109] (b) Sample KG from LogKGKG [181]

Figure 2.1: Sample sub-KGs of Knowledge Graphs extracted from logs

There are also works on extracting/learning causal graph structure from event logs. A causal graph, generally speaking, is a directed acyclic graph structure whose vertices are events and whose edges are causal relations. If there is an edge pointing from vertex $v_i$ to another vertex $v_j$, then we say that $v_i$ is the cause of $v_j$ with some probability. More importantly, the absence of an edge between 2 vertices means there are no causal relations between those 2 events. HPCI [215] was designed specifically for extracting causal graphs for network events. It views the event log generation process as a multi-dimension Hawkes process [101] and infers the relations between events. Their later work THP [27] combined the topological information of the underlying network to improve the accuracy of causal graph extraction.

## 2.4 Similar Works: Evolving Knowledge Based Systems

In this section, we review some system implementations that make the knowledge base evolve over time.

**Never Ending Language Learner** Our idea of building an ever-evolving knowledge base was influenced by the NELL project [29, 127]. NELL continually extracts knowledge from Web pages to enrich its KB. The main difference from other Knowledge Base (KBs) Population projects is that NELL does not limit its knowledge source to a certain corpus but assumes endless input articles. However, The ultimate goal of NELL is not to build an evolving KB, but to continually improve its knowledge extractors (language learners) so that as knowledge is accumulated, its knowledge extractors can exploit the knowledge to perform more accurate knowledge extraction. Moreover, NELL only adds new knowledge to its KB, but never deletes or revises its knowledge.

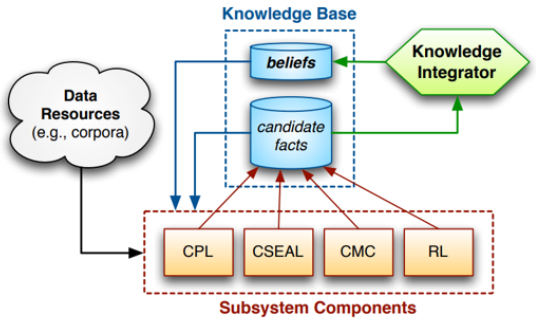



Figure 2.2: An overview of the NELL system [29]

**KG-MRC & GATA** Instead of simply expanding the knowledge base, KG-MRC (Knowledge Graph - Machine Reading Comprehension) [48] and the later work GATA (Graph-Aided Transformer Agent) [5] can learn a knowledge graph from texts and substantially update the states of entities and structure of the graph. KG-MRC leverages neural machine reading comprehension [112] to extract knowledge from procedural texts e.g., an article explaining the procedure of how plants turn water and carbon dioxide into starch. Though they claimed to learn and update knowledge graphs, what they actually operated on were bipartite graphs whose nodes are either entities or locations of entities, and what they updated were the edges between entities and locations. Thus, this is a simplified version of the graph learning and updating task. The followed-up work GATA [5] studied how to learn and update a "belief graph" for an agent playing a

text-based game[42, 116], which is a partially observable environment that only gives texts as observations and receives text commands as actions. The agent's belief is a graph represented as a real-valued tensor $\mathcal{G} \in [-1,1]^{\mathcal{R}\times\mathcal{N}\times\mathcal{N}}$, and the agent can update its belief by reading the textual observations each time it issues an action. Though the belief graph has pre-defined maximum entities and relations, GATA still allows the agent to dynamically learn the semantics of every dimension (entities and relations) and the structure of the graph, but we can't make sense of what the agents learned in their belief graph, typically, what the entities and relations mean.

**Semantic LOG ExtRaction Templating** The previous works are about updating a KB with the new knowledge extracted from **natural language text**, yet our TREAT project aims to make use of **system logs**. The most recent and similar work that extracts knowledge from logs may be the SLOGERT [61] framework and the follow-up KRYSTAL [100] framework that aims to construct a knowledge graph from log data. The SLOGERT framework starts by mining log templates from the log data [219], then extracts information from logs and meta-information about logs, and finally organises the extracted data into RDF format [118], resulting in the final knowledge graph. Thus, the resulting KG is an event knowledge graph whose events represent logs, relating system entities identified from the log messages, such as users, files, servers, etc. Indeed, the major motivation behind SLOGERT is "identifying and linking entities across log sources and enriching them" [61]. Compared with SLOGERT, TREAT takes a further step: unlike SLOGERT which mainly focuses on transforming logs into RDF data and relating log events and system entities, TREAT aims to directly and dynamically model the relationships among system entities without the proxy of events, and track their changes during system evolution.

**ALKB-QA** Another research project that shares the same spirit of TREAT is the ALKB-QA project [113], which studied "Dynamic Updating of the Knowledge Base for a Large-Scale Question Answering System". This project is designed for updating a KB in the KB-QA domain. It takes chatbot records (QA pairs) as an information source and filters out low-quality records, after which these records will be parsed and appended into the KB. Nevertheless, it was tightly coupled with QA application. Even the KB model is defined as a collection of QA pairs $\mathcal{K} = (\mathcal{Q}, \mathcal{A})$ [113], so the techniques developed for the ALKB-QA project are not so generalisable. Similar to NELL, ALKB-QA only adds knowledge into its KB, but never does modification or deletion.

In TREAT, we not only add new knowledge but also revise the existing knowledge when necessary, e.g., when old knowledge becomes outdated.

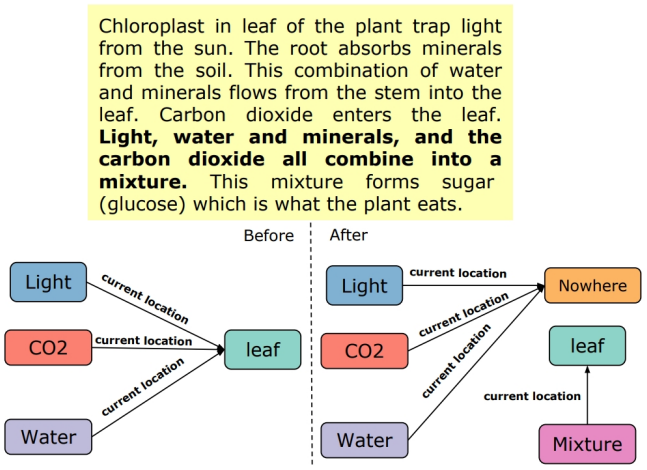



Figure 2.3: Snapshot of the dynamic knowledge graph created by KG-MRC [48]

## 2.5 Summary and Research Gap

There have been tons of works in the popular research field of knowledge representation and knowledge base construction & updating. Knowledge can be represented by both symbols and numerics. Symbolic representations are usually in the form of logic, while sub-symbolic representations are usually in the form of vectors or tensors (neural networks). Both representations have their advantages. Sub-symbolic representations are better at scalability and capturing implicit semantics, and become widely adopted because of the boost of deep learning and neural networks. In terms of transparency and ease of updating, which is desired in domains including telecommunications, symbolic representations show their supremacy, and thus in the TREAT project, we chose this representation. Constructing such a knowledge base in symbolic representation automatically is usually done by extracting formally structured information from external sources. We've seen that lots of efforts have been dedicated to extraction from natural language texts, including methods, systems, and benchmark datasets. However, extracting knowledge from system logs, especially logical theories from raw logs, is still under-explored, let alone establishing a continually updating mechanism. One of the motivations of the TREAT project is to address this gap.

# Chapter 3

# Background

This chapter provides some background information about the TREAT project and preliminary knowledge about the employed approaches. It is organised as follows. Section 3.1 introduces the Autonomous Network Vision, which motivates and positions the TREAT project. Section 3.2 lays out some fundamentals of Knowledge Graph (KG), which is chosen to be the main knowledge representation scheme in this project. Section 3.3 outlines some prior research in the TREAT project mainly conducted by my collaborators, including an approach to extract knowledge from logs and a method to perform root cause analysis. Section 3.4 introduces some general methods of log analysis that will be used in our methods, such as log template mining.

## 3.1 The Vision of Autonomous Network

The Autonomous Network [71, 63] envisions a telecommunication network system that can run and adapt to changes with minimal or even no human intervention. Such a level of autonomy is expected to be achieved by the accompaniment of an intelligence engine that can process various data and make vital decisions. The TREAT project is a research effort to build such an intelligence engine or a sub-system of this intelligence engine.

### 3.1.1 Telecommunication Networks

A telecommunications network, especially a modern one, is an arrangement of computing devices(e.g., switches) and transmission links(e.g., fibre optics cable) that enables information exchange between distant senders and receivers [11].

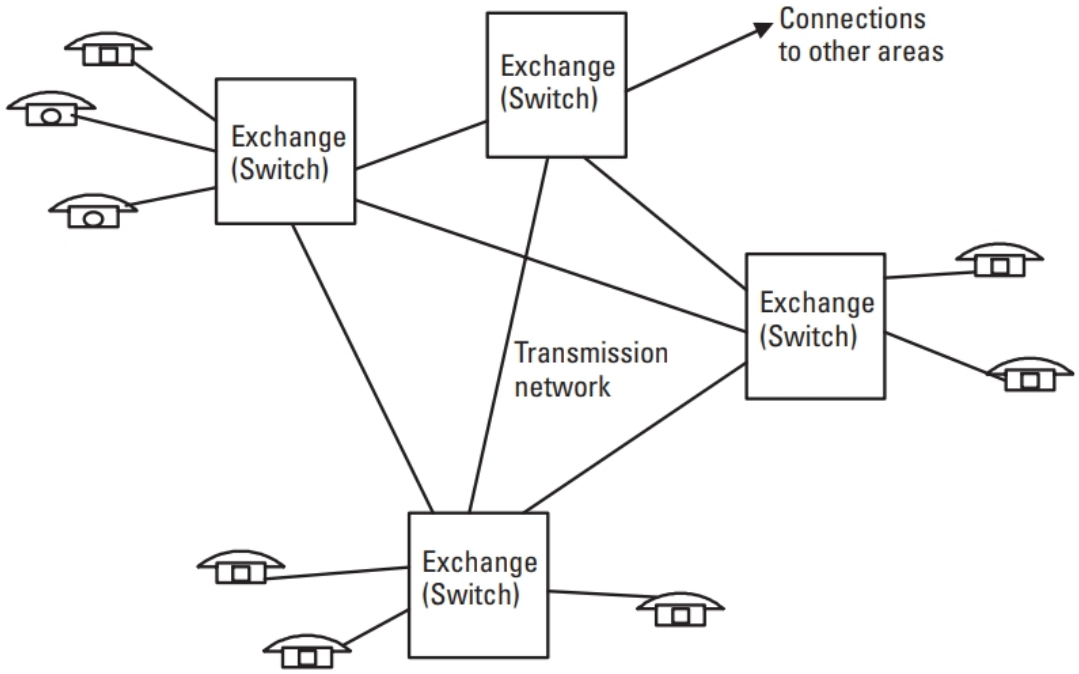



Figure 3.1: An example of a basic telecommunication network [11]

Figure 3.1 shows an example of a basic telecommunication network. The network consists of a number of switches, which are connected by transmission links. Instead of connecting everyone to every other one resulting in a fully connected network, end users connect to switches, which are responsible for redirecting signals to the correct destination via other switches and the transmission links. With the help of switches as intermediaries, the number of transmission links can be reduced significantly compared with a fully connected network.

Telecommunication networks have a long history and have evolved into numerous variants. In the very beginning, telecommunication networks were merely post office networks. Letters and parcels were transmitted on roads and streets by human posters. After the Second Industrial Revolution, textual telegraphs could be transmitted via electricity and electromagnetic waves, and telegraph networks were built. Then there were telephone networks that carried voice messages, and later computer networks that transport all kings of data signals. Nowadays, telecommunication networks are mostly computer networks that carry all kinds of data, including voice, video, text, and so on. With the wide adoption of smartphones, essentially personal mobile computers, mobile telecommunication networks are becoming increasingly important, and are perhaps the most widely used network.

Mobile telecommunication networks have evolved through generations. Starting in the 1980s, the first generation (1G) mobile telecommunication network supported only voice calls. In the 4G era, we can watch online movies and have video conferences on our mobile phones. We are now in the 5G era when a lot of new sophisticated applications are expected to be realised, including the Internet of Things, remote surgery, virtual reality, and so forth [137]. As 5G systems are expected to support much more complicated applications, they are designed to achieve several key requirements: enhanced Mobile Broadband, ultra-Reliable Low Latency Communications, and massive Machine-Type Communications [170]. In order to meet these requirements, new technologies are introduced. One of the key innovations lies in the system architecture that the whole network is further virtualised and decomposed into multiple Network Functions (NFs), which are software entities. In this way, the network system can be defined and configured by simply composing and configuring software components based on common hardware infrastructure. Compared with the 2g/3g/4g systems in which management software is coupled with dedicated hardware, the software-defined 5G network system is more flexible and scalable. Nevertheless, flexibility and scalability come with the challenge of complexity, and the increasing difficulty in O&M.

### 3.1.2 Autonomous Network

Modern telecommunication network systems are very powerful in terms of transmitting data at high speed, low latency, and high reliability, as well as large capacity to offer communication services for a huge number of terminal devices simultaneously. However, the complexity of the network systems has also increased dramatically, due to the scale, the number of devices, the number of **categories** of devices, various policies, all kinds of administrative requirements, etc. In fact, telecommunication network systems are considered to have made up the most complicated equipment in the world [11]. This complexity not only makes the network systems difficult to operate and maintain but also potentially introduces more errors and failures. Hence, it is necessary to substantially simplify the network systems and automate as many operations as possible. The idea of Autonomous Networks was proposed to meet this demand.

The term “Autonomous Network” was first coined in a TM Forum [1] white paper published in May 2019 [71]. The white paper summarised that in the future, as the number of connected devices increases, presumably 70 to 100 billion termi-

[1] https://www.tmforum.org

nals, the complexity of infrastructure will increase exponentially, which incurs huge maintenance costs and prevents substantial growth. Hence, transformations are necessary, such as simplifying the network architecture and introducing artificial intelligence technologies to facilitate operations.

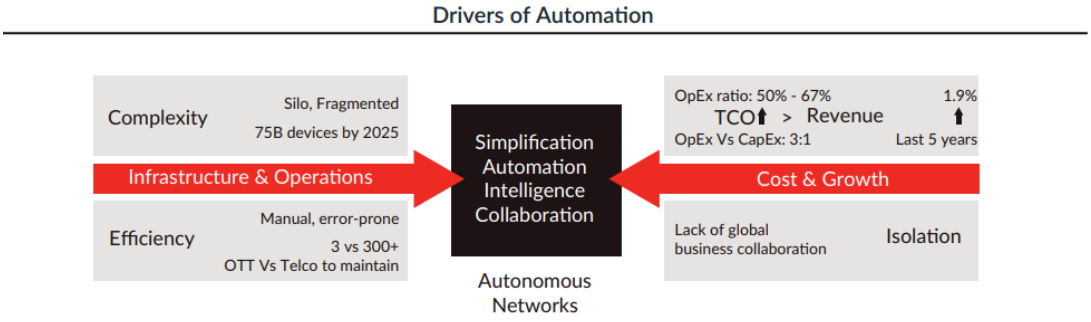



Figure 3.2: Driving factors for Autonomous Networks [71]

Yet it was reported that by 2018, about 56% of the CSPs (Communication Service Providers) around the world will have little or no automation in their network operations [71]. Consequently, TM Forum initiated a vision of Autonomous Networks, which is to appeal to the companies, institutes, and standardisation organisations of the industry to work together and promote automation in the networks. Moving from no automation to nearly full automation requires progressive efforts. Inspired by the Autonomous Vehicle dividing automation into 6 (0-5) levels [168], the Autonomous Network Vision also defines 6 levels of automation, as illustrated in Figure 3.3. The higher the level, the more and the harder jobs the network can automate. For example, L0-level networks can only execute operations that are explicitly instructed by human operators, while L5-level networks can not only run executions and be aware of real-time environmental changes but also make decisions based on predictive analysis and handle user intents in order to optimise user experience.

Once the Autonomous Network Vision was proposed, it was immediately advocated by other institutes and companies in the industry [66]. Just to name a few, ITU-T[2] (International Telecommunication Union - Telecommunication Standardization Sector) set up a Focus Group on Autonomous Networks (Study Group 13) and published technical specifications on the architecture framework [87] and evaluation of trustworthiness [88]. 3GPP[3] (The 3rd Generation Partnership Project) issued standards that introduce levels of automation into 5G networks [1], architecture enhancements

[2] https://www.itu.int/en/ITU-T
[3] https://www.3gpp.org

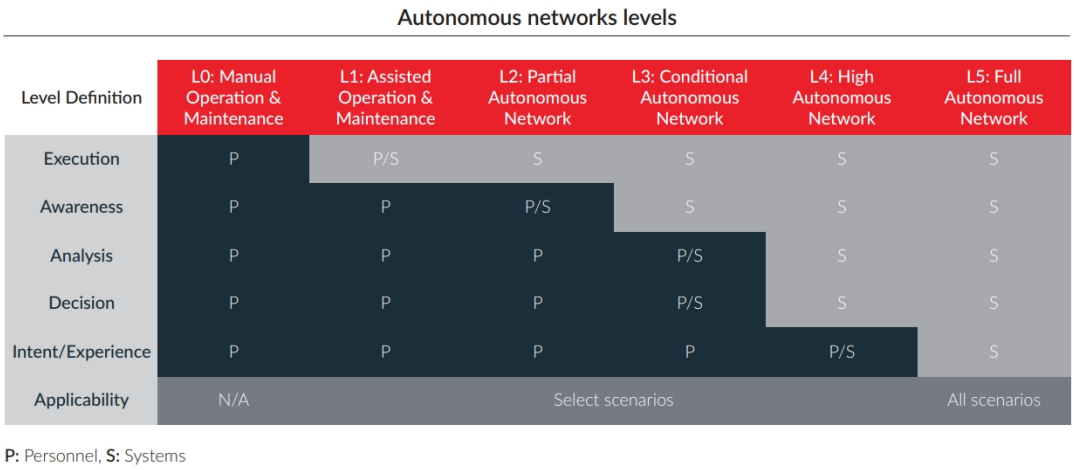


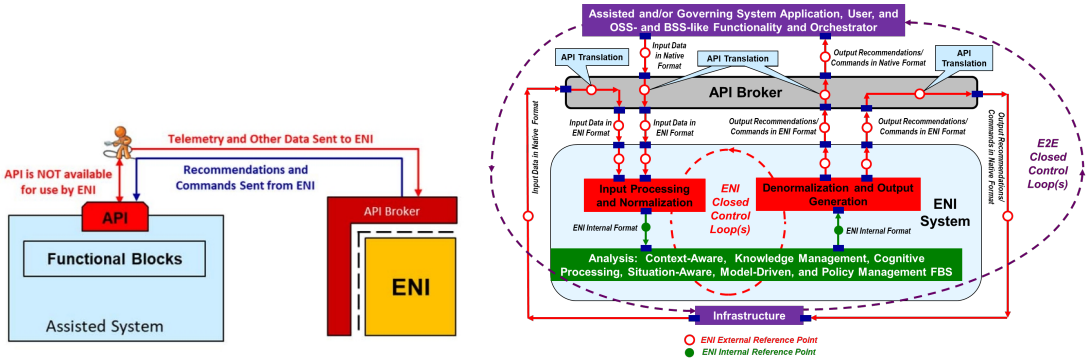

Figure 3.3: Levels of Autonomy in Autonomous Networks [71]

for 5G systems to support network data analytics services [2], etc. ETSI[4] (European Telecommunications Standards Institute) drafted white papers on the vision and kept updating it [63, 66]. What's more, ETSI formed several Technical Committees and Study Groups that deliver solutions to Autonomous Network related topics, such as ENI (Experiential Networked Intelligence) [65], ZSM (Zero-touch network and Service Management) [67], GANA (Generic Autonomic Networking Architecture) [62], and so on.

### 3.1.3 Autonomous Network and the TREAT project

The TREAT project aims to develop a methodology that can constitute part of the intelligence engine of an autonomous network. Such an intelligence engine could be plugged into the telecommunication network so as to gather data from the network, analyse the data, and recommend actions to the network. Take the ETSI ENI (Experiential Networked Intelligence) for example [65]. ENI is a cognitive network management system that aims to "quickly recognise and incorporate new and changed knowledge", and hence support decision-making, such as which services are appropriate to be offered and which services are in danger and need to be adjusted. Figure 3.4a shows the position and 3.4b shows the high-level architecture of ENI. In Figure 3.4a, the "Assisted System" is the telecommunication network that ENI is supporting. ENI pulls telemetry data and other forms of data, e.g., logs, alarms, KPI metrics, etc. from the Assisted System, and recommends actions (executable commands) back.

[4] https://www.etsi.org




(a) ENI Position [65] (b) ENI Architecture [65]

Figure 3.4: High-level overview of the ENI system

The TREAT system can be viewed as an instance of the ENI, or a subsystem of an ENI, though it did not aim to fully conform with the ENI architecture. For example, it does not have the so-called ”recommendation mode” and ”management mode” in ENI. The core scope of the TREAT project is to develop a method that extracts knowledge from logs to build a knowledge base and continually update its knowledge when new knowledge is acquired.

One of the differences from the ENI which takes various forms and sources of data as input, is that the TREAT project focuses on extracting knowledge from system logs, due to data availability and computing resources. A telecommunication network system usually consists of multiple modules, including the access networks that take charge of different types of device access; the edge networks that offer real-time and localised computation required by connectivity, administration, and analytics; the core network where major data processing, forwarding, and storage happens. The core network is equipped with plenty of computing resources, so we can monitor and collect all kinds of data frequently from the core network, and perform heavy computing tasks, like deep learning. However, the access networks and edge networks are usually distributed and the computing resources are limited. For these modules to realise autonomy, lightweight data collection and processing methods are necessary. We consider that the log data are the primary most available data source. Hence, TREAT focused on developing a method that can use the system logs.

## 3.2 Knowledge Representation with Knowledge Graph

To build an intelligence engine, an important sub-task is to represent knowledge, as well as acquire, store, and make use of the knowledge, which is also the core scope of the TREAT project. This section introduces some basic concepts of knowledge representation with a Knowledge Graph, which is our chosen representation.

Built on top of description logic [15], Ontologies and the later Knowledge Graphs [146] technology is one of the mainstream knowledge representation approaches. Simply speaking, a knowledge graph is a specification for a domain of interests that describes the concepts, entities, and the relationships between them. Usually, a knowledge graph can be represented as a set of subject-predicate-object $\langle s, p, o \rangle$ triples, where the subject $s$ and object $o$ represent some individual entities or conceptual classes, while the predicate $p$ asserts the relationship between $s$ and $o$. As its name indicates, the whole body of knowledge can be represented as a graph, where the concepts and entities are represented by nodes, and the relationships are represented by edges. Figure 3.5 shows an example knowledge graph.

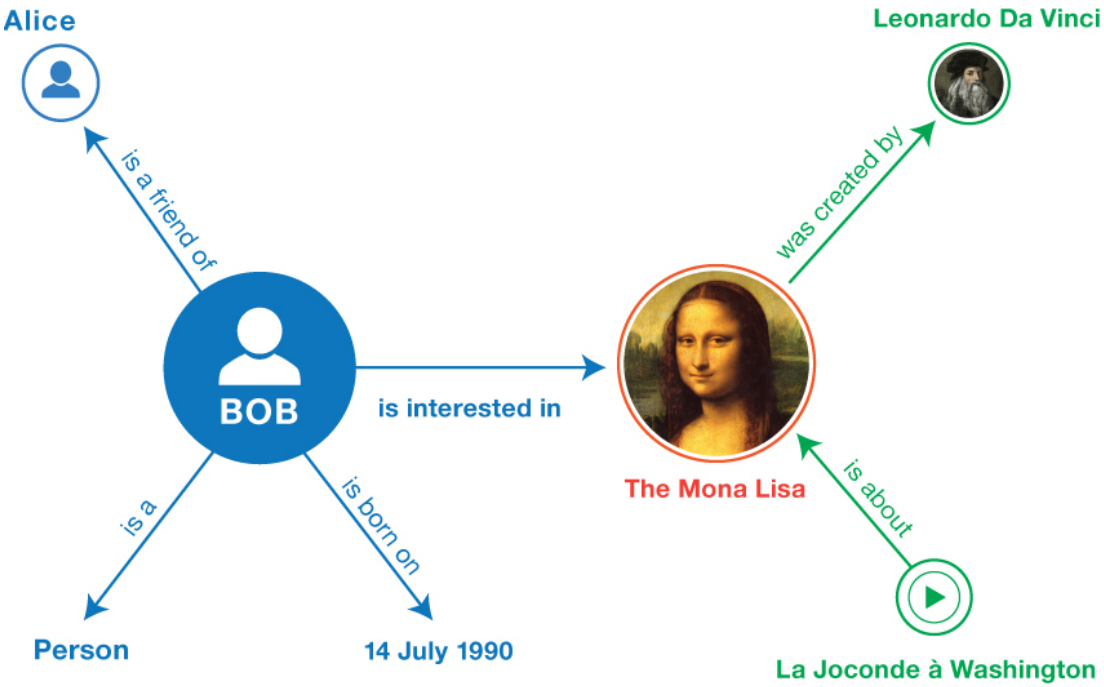



Figure 3.5: An example knowledge graph [118]

More formally, **Knowledge Graphs** [147] are represented in a standard format for graph-structured data. A *knowledge graph* $\mathcal{G}$ is a tuple $(\mathcal{E}, \mathcal{R}, \mathcal{T})$, where $\mathcal{E}$ is a set of

entities, $\mathcal{R}$ is a set of relation types, and $\mathcal{T}$ is a set of relational triple $\{\langle s,p,o\rangle | s,o \in \mathcal{E}, p \in \mathcal{R}\}$ [144].

Structured knowledge representation by graph or network is not a new idea of ontologies and knowledge graphs. Earlier in the 1960s Quillian proposed the Semantic Network [154, 155], where concepts are indicated by nodes and linked together by arcs. The meaning of a node is defined as its connection with other nodes[5]. Later in the 1980s, Minsky proposed the Frame model [126, 184] to represent knowledge of a narrow subject, which can include not only attribute-value pairs but also simple procedures. Figure 3.6 shows 2 simple examples of Semantic Networks and Frames. Though a Semantic Network or Frame can express knowledge in a structured way, researchers made criticisms that they lack clear semantics [208]. For example, there is ambiguity in the Semantic Network between set membership and subset, i.e., the "IS-A" link in the Semantic Network cannot clearly tell whether a concept is equivalent to another concept, a subclass of another concept, or a member of another concept.

Compared with Semantic Networks and Frames, Ontologies and Knowledge Graphs were developed based on description logic that has a solid foundation in syntax and semantics. In practice, Ontology and Knowledge Graph get support from the W3C standards, which includes the Resource Description Framework (RDF) [6], the Web Ontology Language (OWL) [7], and the SPARQL Protocol and RDF Query Language (SPARQL) [8]. Hence, engineers and developers can follow the same guidelines to build and use ontologies and knowledge graphs, which increase the reliability, interoperability, and reusability of relevant applications. For these reasons, we chose to represent knowledge in the form of Knowledge Graphs in the TREAT project.

## 3.3 Prior efforts of The TREAT Project

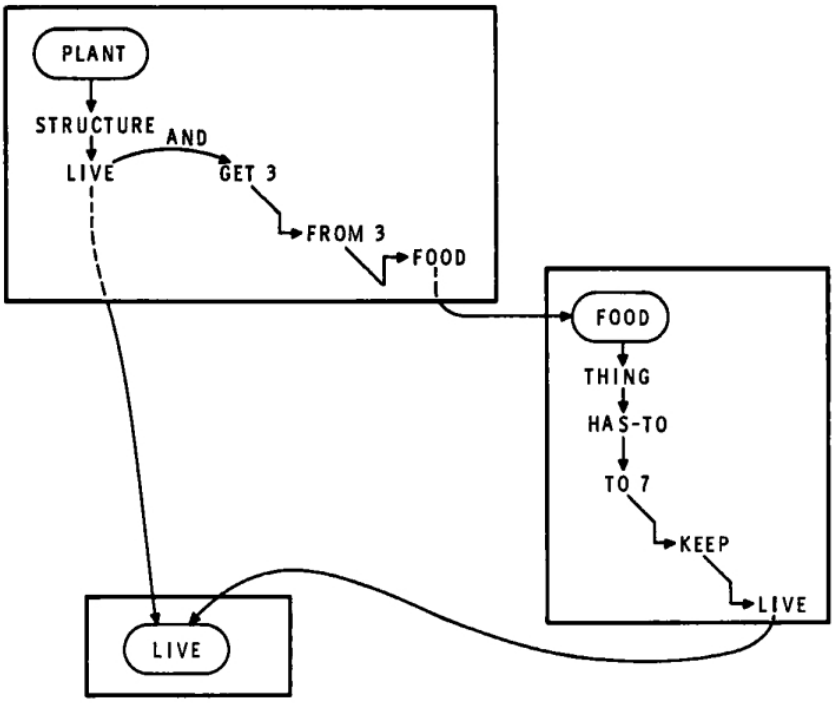
The TREAT project is a collaborative project. The research output in this thesis was more or less inspired by other research outcomes of the project done by my collaborators. This section briefly introduces some prior efforts and the lessons of the project, including an alternative approach to extract knowledge from logs and a method to perform root cause analysis, with the critique of these approaches that led to my other

[5] "*To summarize, a word's full concept is defined in the model memory to be all the nodes that can be reached by an exhaustive tracing process, originating at its initial, patriarchal type node, together with the total sum of relationships among these nodes specified by within-plane, token-to-token links.*" [155]

[6] `https://www.w3.org/TR/rdf11-primer/`

[7] `https://www.w3.org/TR/owl2-overview/`

[8] `https://www.w3.org/TR/sparql11-overview/`




(a) Example semantic network describing two paths from “Plant” to “Live” [155]

```
(MACDONALDS
(A-KIND-OF              (VALUE (RESTAURANT)))
(LOCATION               (VALUE (ANN-ARBOR))))
(SERVING-STYLE          (VALUE (FAST-FOOD)))
(FOOD-STYLE             (VALUE (AMERICAN)))
(COST                   (VALUE (CHEAP)))
(DECOR                  (VALUE (MINIMAL)))
(RATING                 (VALUE (FAIR)))     )
```

(b) Example Frame describing the subject “MacDonald’s” [184]

Figure 3.6: Simple examples of Semantic Network and Frame

approach. Readers can refer to the cited papers for more information.

### 3.3.1 Extracting a Knowledge Graph from Logs

In the beginning of the TREAT project, we decided to follow a conventional approach to extract a knowledge graph from textual data, leading to the LEKG (Long Extraction Knowledge Graph) method [197], which is our first attempt to achieve the project goals. The main idea of LEKG is to extract triples from logs, apply reasoning techniques to infer more knowledge, and then use filtering to get rid of invalid triples with

the help of the so-called Background Knowledge Graph (BKG). In a word, LEKG is not only an extraction method but an "extract-enrich-filter" process. Figure 3.7 shows the general procedure of LEKG.

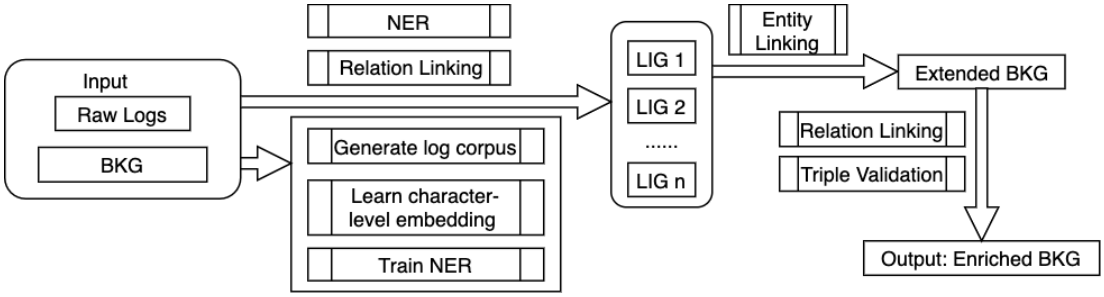



Figure 3.7: The general procedure of LEKG [197]

Briefly speaking, this LEKG method regards the manual and logs to be both natural language texts, and conducts standard information extraction procedure, i.e., Named Entity Recognition [128] to identify possible entities from texts, Entity Linking [171] to map the identified text span to an entity ID in an ontology, Relation Extraction [136] to extract the relation between entities, and so on. Hence, the LEKG extraction method involves the following steps:

1. **Corpus Preparation** This step selects a subset of texts from the documentation and logs and performs preprocessing, such as formatting and human annotation, enabling model training for subsequent NLP models.

2. **Named Entity Recognition** This step, as its name indicates, recognises Named Entities from a piece of documentation text or log text. A notable remark is that in this program, the NER module is implemented using SpaCy[9] NER tools, and trained on the annotated corpus.

3. **Entity Linking** The NER step simply outputs text spans that indicate entities. This module links the text spans to the entity/relation of the Background Knowledge Graph. This is implemented as fuzzy string matching, e.g., computing the string overlap or other forms of similarity between a text span and the labels, and descriptions of entities in the ontology. If no match is found, a new entity is created in the Background Knowledge Graph.

4. **Relation Inference** This step infers relations for given pairs of entities via predefined rules. At the end of this step, new triples are generated. As the logs

[9]https://spacy.io/api/entityrecognizer

are clustered and processed in different groups, the output triples constitute the so-called Local Instance Graphs (LIGs).

5. **Validation with Constraints**. After extraction and enrichment, LEKG conducts an additional post-processing step. This step validates the new triples by the predefined constraints (could be given by human experts), and filters out invalid ones. At the end of this step, the valid LIGs will be fused into the BKG.

This procedure can be run iteratively. As new logs are generated by the assisted network and pushed to the intelligence engine, this procedure captures new knowledge from the logs and updates the knowledge base (the Background Knowledge Graph).

**Limitation** Note that the LEKG method demands quite a lot of human labour. First, the corpus preparation step requires manual annotation. Second, the initial Background Knowledge Graph is constructed by hand. Third, the relation inference step and the validation step require inference rules that were predefined by human experts.

Another more serious limitation is the knowledge model. In the LEKG method, the new triples are continually added to the Background Knowledge Graph, but there is no mechanism to revise or remove triples from the existing knowledge, which is an indispensable capability for the intelligence engine to adapt to changes of the assisted network. Furthermore, a knowledge base with ever-growing size not only gradually eats out the storage but also makes it difficult to perform reasoning and inference.

The most severe drawback of this approach lies in its core idea of treating the log messages as natural language sentences and applying NLP techniques, which is an inappropriate assumption. The logs are generated by some engineer-crafted templates, and thus they are closer to formal languages than natural languages. Consequently, though the LEKG is a valuable attempt, we turned to other approaches that exploit the syntax of logs.

### 3.3.2 Root Cause Analysis

The ultimate goal of the TREAT project is to build an intelligence engine that can support various downstream applications of telecommunication networks O & M. One of the most important applications is Root Cause Analysis (RCA) [177]. Hence, after we developed the LEKG method, we started to explore a possible means to use the knowledge base for RCA tasks [106]. This can not only serve as proof of the usefulness of the constructed knowledge base, but also serve as an indirect way to evaluate how

well we accurately extract the knowledge and update the knowledge base. To this end, we developed a method that can discover missing information from the logs that are instrumental to fault recovery [106].

We took the perspective of theorem proving, a pure logical method (against the so-called hybrid or neuro-symbolic method [83]), to represent the problem. The assisted network is represented by the knowledge base, and the faults, are represented by logical formulas. We assumed that when faults occur, it should be possible to deduce the faults from the knowledge base by establishing proofs. In case the knowledge base is incomplete and thus the faults cannot be deduced, we use the ABC system, i.e., Abduction, Belief revision, and Conceptual change methods [105] to discover the missing information and repair the knowledge base. After the missing information is found and added to the knowledge base, *the shortest proof towards the faults is considered as the root cause of these faults*. We then try to block all the proofs of the faults, by adding or removing formulas from the knowledge base, which mimic the real operations of fault-healing. The blocking solution with minimal changes to the knowledge base is considered to be the best one, and corresponding real operations to the assisted network are considered to be the best fault recovery solutions.

In short, I outlined a novel method for RCA that makes use of the knowledge base constructed by LEKG. However, the perspective and the way we framed the RCA problem are different from the common understanding in the literature, which usually defines a root cause as an ultimate event, error, alarm, or component of the system [177], instead of a proof towards faults. Yet inspired by this work and the idea of indirect evaluation, I targeted an easier problem, which is Fault Localisation, and developed a simpler but more general method.

## 3.4 Log Parsing

Log parsing is a class of techniques that extract structured information from unstructured log messages [82]. It is a fundamental step and a prerequisite for many log analysis tasks, such as log summarisation, anomaly detection, and root cause analysis. As the TREAT project needs to handle logs generated by the assisted network, log parsing is a valuable tool. This section introduces some basic notions and ideas of log analysis.

In terms of logging, engineers can log information in whatever manner they like, e.g., use the `print` function to draw an ASCII diagram. Figure 3.8 shows various

```
01/24 10:36:07.280: [app] INFO: Configuration: '/open5gs/install/etc/open5gs/amf.yaml'
(../lib/app/ogs-init.c:126)
01/24 10:36:07.280: [app] INFO: File Logging: '/open5gs/install/var/log/open5gs/amf.
log' (../lib/app/ogs-init.c:129)
01/24 10:36:07.286: [metrics] INFO: metrics_server() [http://172.20.0.210]:9091 (../
lib/metrics/prometheus/context.c:510)
01/24 10:36:07.286: [sbi] INFO: NF Service [namf-comm] (../lib/sbi/context.c:1334)
01/24 10:36:07.286: [sbi] INFO: nghttp2_server() [http://172.20.0.210]:7777 (../lib/
sbi/nghttp2-server.c:237)
01/24 10:36:07.286: [amf] INFO: ngap_server() [172.20.0.210]:38412 (../src/amf/
ngap-sctp.c:61)
```

(a) Standard log entries generated by logging instructions

```
Write out database with 1 new entries
Data Base Updated
Generating RSA private key, 1024 bit long modulus (2 primes)
.........+++++
....+++++
e is 65537 (0x010001)
Using configuration from /usr/lib/ssl/openssl.cnf
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 2 (0x2)
        Validity
            Not Before: Jan 24 10:36:05 2023 GMT
            Not After : Jan 21 10:36:05 2033 GMT
        Subject:
            countryName               = KO
            stateOrProvinceName       = Seoul
```

(b) Ad-hoc logs generated by printing statements

Figure 3.8: Various logs produced by software systems. The example logs shown are generated by a deployed Open5GS system.

examples of logs. Because of that, software engineers can produce logs in any format.

Nevertheless, there is a general practice to do logging. To generate logs, engineers need to decide what information they care about, and write a log template with some placeholders in the source code of the system. On running, the system will produce the logs by the log template and fill the placeholders with runtime information.

As we can see in Figure 3.9, a log entry is generated by a logging instruction and captures a particular system event. The log entry consists of a header and a message. The header, which includes elements like timestamps, verbosity levels (such as ERROR, INFO, DEBUG), and system components, is set by the logging framework and is in the same format for all log entries. While the body of the log entry, which is the message, is set by the programmer and is in a free-text format. The message is composed of a template (constant strings) and a set of parameters (variable strings that carry dynamic runtime information of that log event). Due to the nature of the log entries, engineers and researchers believed that it would be a good idea to distinguish the

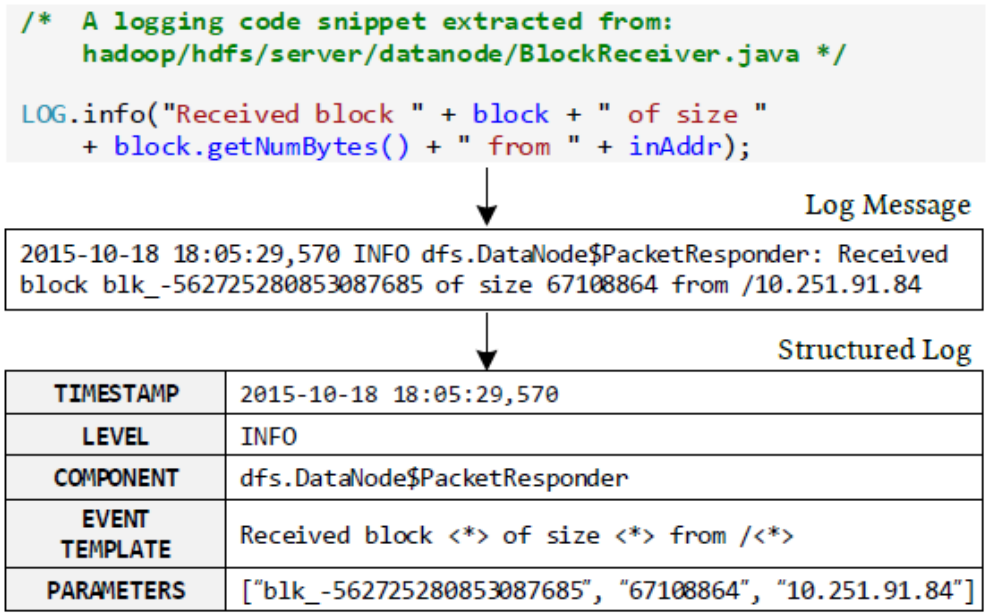



Figure 3.9: Log generation and parsing by template [219]

template and the parameters, and the output of log parsing should be a structured log message containing the log template and the key parameters. Once we turn the seemingly unstructured log messages into structured log events, all the tools in our toolbox developed for analysing structured data, such as logical rules, statistical learning, etc., can be applied to the log analysis tasks.

There have been loads of research [82] and development [219] efforts on log parsing. The simplest way to perform log parsing is by scanning the source code of the software system to find out all the logging statements, and then design templates, e.g., regular expressions, to match the logging statements and extract the parameters. An obvious drawback of this approach is that it costs a lot of human labour to manually design templates for each logging statement, especially when modern software updates frequently. Each time a new version is released, it will need to check again and the templates need to be updated accordingly. Moreover, it demands access to the source code, which is not always available.

To address these problems, researchers have proposed automated log parsing techniques. For example, the SLCT (Simple Log Clustering Tool) method [189] counts the occurrence of all tokens in a corpus of logs and calculates the frequency of each token. Then it filters the tokens whose frequency exceeds a threshold to form the log templates. Another approach is to leverage the textual clustering algorithms to cluster the log messages into groups [72, 190], and then extract the templates from the groups

by removing the parameters by developer-crafted rules.

However, the aforementioned approaches all run in an "offline" mode, meaning that they need to collect a large number of log messages and then process them in a batch manner. When software systems are updated, the log parsing process needs to be repeated with a new corpus of logs produced by the updated system. This is not only troublesome but also not applicable in some domains. For example, in the 5G era, the telecommunication network is built using the Microservice architecture, which allows users to easily introduce new services into the system in runtime. New services come with new logs, and the log parsing algorithms need to be able to parse the new logs in runtime, hence "online" log parsing algorithms are needed. Example methods include Spell [60] and Drain [81], which maintain a tree-like structure to hold log groups and allow online log group merging to create new log templates.

## 3.5 Summary

In this chapter, I provided some background information and context for the research presented in this thesis. I introduce the vision of Autonomous Networks, which envision future telecommunication networks that can operate with minimal human intervention, enabled by an intelligent system that gathers data, processes data, and makes decisions to enable self-operation and self-maintenance. The TREAT project aims to develop an intelligent system contributing to this vision, with a focus specifically on extracting knowledge from system logs.

I discussed some prior work in the TREAT project as well, including the LEKG method for extracting knowledge graphs from logs using natural language processing techniques, and a method on root cause analysis using logical reasoning over the extracted knowledge. Limitations of these approaches are identified, motivating the exploration of new methods that better exploit the structured nature of log data. The chapter also provides an overview of log parsing techniques, which aim to extract structured information from unstructured log messages and enable further log analysis. This sets the foundation for the log parsing and knowledge extraction methods that will be presented in subsequent chapters.

# Chapter 4

# The TREAT Framework

## 4.1 Overview

Recall that the aim of this research is a systematic methodology that can automate the construction and updating of a knowledge base from a continual stream of logs. In this chapter, I present our design of such a systematic framework. Though the TREAT project contains other outputs than just a final system, for simplicity, I will name this system *TREAT* in the rest of this thesis. As mentioned in the §1, the expected input of the TREAT system is a stream of logs and the expected output is a continually updated knowledge base (KB), as illustrated in Figure 4.1. Now we formally describe the input and output of the TREAT System.

### 4.1.1 Input: Log Stream

We assume that the TREAT system can access a continual stream of logs produced by the assisted software system, e.g., the autonomous network, and process the logs to update its maintained KB, and the logs correctly reflect what happened within the system. A *log* is a semi-structured textual record, comprising several fixed fields and a piece of message generated by a log template. A *log stream*, as the name indicates, is a possibly unbounded sequence of logs. Formally, it may be defined as:

$$S = (\tau_1, d_1), \cdots, (\tau_i, d_i), \cdots$$

where $\tau_i$ is the $i$th integer time step and $\tau_i \leq \tau_{i+1}$, while $d_i$ represents the log generated at time $\tau_i$. More formally, we denote $T \subset \mathbb{N}$ to be a set of integer time steps, e.g., Unix timestamp[1], such that $\tau \in T$, and denote $D$ to be a set of all possible logs generated by

[1] `https://unixtime.org/`

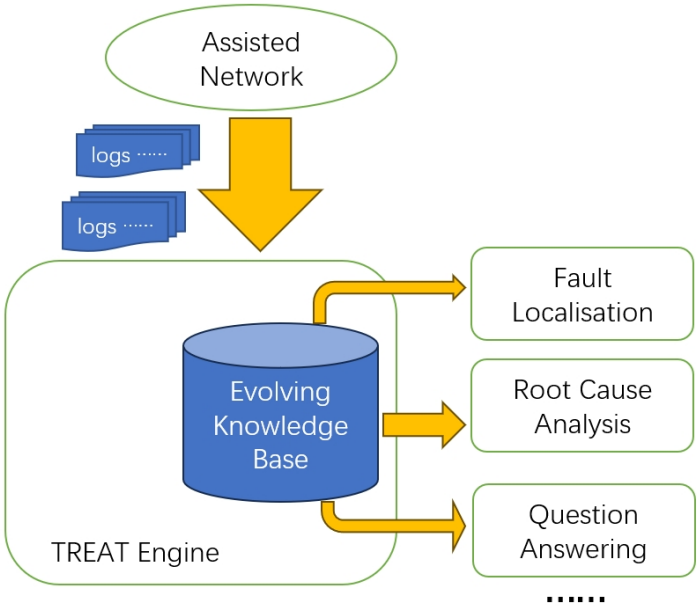



Figure 4.1: The abstract overview of the TREAT system

the assisted software system such that $d \in D$. Therefore, we could define a log stream to be a function

$$S : T \to 2^D$$

which returns a set of logs for each time step.

Furthermore, a log entry can be decomposed into 3 parts:

1. header attributes, like the timestamp, level, and component of the log. These header attributes (the key, not the value) are usually the same regardless of the log type
2. the template of the log message, and we regard the logs generated by the same template belong to the same log type
3. the *log parameters* (filling in the placeholders/slots of the template). The log parameters are dependent on the log types.

Without losing generality, we may formalise a log as a logical term

$$d = e_i(a_1, a_2, ..., p_1^{[i]}, p_2^{[i]}, ...)$$

where the predicate $e_i$ denotes the log type/template, $a_1, a_2...$ represent the header attributes that are shared by all log entries, like the timestamp, severity level, etc, and $p_1^{[i]}, p_2^{[i]}, ...$ represent the parameters that carry specific information of that log type.

Take the log in Figure 3.9 for example.

```
2015-10-18 18:05:29,570 INFO dfs.DataNode$PacketResponder:
   Received block blk_-562725280853087685 of size 67108864
   from /10.251.91.84
```

`2015-10-18 18:05:29,570` is the time, which can be converted to a Unix timestamp "1445187929570"[2]; `INFO` is the severity level; `dfs.DataNode$PacketResponder` is the component that generated this log. These are the header attributes (and their values in this specific log entry). The template is identified as:

```
Received block <*> of size <*> from /<*>
```

along with its important parameters (such as ["blk_-562725280853087685", "67108864", "10.251.91.84"]). In this context, "`<*>`" indicates the placeholder for each parameter. If we assign `log_type_1` as the id of that log template, then the log can be converted into a structured logical term:

```
log_type_1(
    "2015-10-18 18:05:29,570",
    "INFO",
    "dfs.DataNode$PacketResponder",
    "blk_-562725280853087685",
    "67108864",
    "10.251.91.84"
)
```

or expressed in JSON format if we can assign meaningful names to the attributes and parameters:

```
{
    "log_id": "log-xxxxxxxxxxxxxxxxxxxxxxxxx",
    "log_type": "log_type_1",
    "time": "2015-10-18 18:05:29,570",
    "component": "dfs.DataNode$PacketResponder",
    "level": "INFO",
    "param1": "blk_-562725280853087685",
    "param2": "67108864",
```

[2]Meaning the 1445187929570-th milliseconds seconds since 00:00:00 UTC on Thursday, 1 January 1970

```
        "param3": "10.251.91.84"
    }
```

where the "log_id" is a randomly generated unique ID for each log entry, and "param1", "param2", and "param3" are dummy names of the parameters. The JSON format provides more information than the logical terms, as the logical terms only show the predicate and values, but the JSON format also reveals the names of values as well. However, converting the extracted log information in JSON format requires that we assign meaningful names to log parameters, which is not included in raw logs, but we can get these meaningful names by human labour, or even via Large Language Models.

### 4.1.2 Output: Self-evolving Knowledge Base

The TREAT system is associated with a KB. Each time the TREAT system receives a set of logs, it performs necessary updates on its associated KB. Inspired by the field of Knowledge Graphs [193], we represent a knowledge base in the following form:

$$\mathcal{K} = \{\mathcal{E}, \mathcal{P}, \mathcal{T}\}$$

where $\mathcal{E}$ is the set of entities, $\mathcal{P}$ is the set of predicates (relations), and $\mathcal{T} = \{\langle s, p, o\rangle | s, o \in \mathcal{E}, p \in \mathcal{P}\}$ is the set of triple statements.

Partly inspired by Event Calculus [98, 125, 188], I define 3 operations upon this knowledge model:

- $add(\mathcal{K}, \langle s, p, o\rangle)$ add the given triple statement and the involved entities and relations into the knowledge base: $\mathcal{E} := \mathcal{E} \cup \{s, o\}$, $\mathcal{P} := \mathcal{P} \cup \{p\}$, and $\mathcal{T} := \mathcal{T} \cup \{\langle s, p, o\rangle\}$
- $delete(\mathcal{K}, \langle s, p, o\rangle)$ delete the given triple statement in the knowledge base $\mathcal{T} := \mathcal{T} \backslash \{\langle s, p, o\rangle\}$
- $remove\_entity(\mathcal{K}, e)$ delete the given entity and all the triple statements that involve this entity: $\mathcal{E} := \mathcal{E} \backslash e$, $\mathcal{T} := \mathcal{T} \backslash \{\langle s, p, o\rangle | s = e \vee o = e\}$

We employ a Closed World Semantics[163, 59]: for a given triple statement $t$, it is considered to be true if and only if it is included in the knowledge base $t \in K$. Moreover, if the knowledge base $K$ is accompanied by a set of rules $R$, then a triple statement $t$ is considered to be true if it could be entailed by the knowledge base and the rules $K \cup R \models t$

The design choice of representing knowledge by triples was made at the beginning of the TREAT project. At that time we wanted to use a flexible representation, and immediately we thought of Knowledge Graphs. A lot of previous knowledge base projects [58, 209, 127] also adopted triples to represent knowledge. Thus, we kept this tradition. Though we considered general first-order logic, JSON, and so on, I still prefer triple representation because of its simplicity and its good enough expressive power. What's more, representing knowledge by triples enables the post-processing of the knowledge by the existing Knowledge Graph Embedding (KGE) techniques that map the entities and relations into a continuous vector space, and thus benefits the downstream tasks which require a vector representation of the knowledge.

The rationale of the 3 defined operations is a bit complicated. At first glance, readers familiar with Event Calculus (EC) will know that *add* and *delete* are terms borrowed from EC. The first operation, *add*, is used to add a new fact.

The second operation is used to represent that a fact becomes false, e.g., a connection between 2 components is lost. The implementation of this operation is simply deleting the triple from the KB, because we employed a Closed World Semantics. We ever considered adopting an Open World semantic [59], but we would have to assign a truth value to each added triple, and the *delete* operation would need to mark the truth value from 'true' to 'false', leading to more tedious designs.

The *remove_entity* operation, which abstracts multiple subtle *delete* operations, except for offering great convenience, is necessary to remove old entities from the entity set of the Knowledge base. Some of the application scenarios, such as telecom networks, often see objects (e.g., devices, users, etc) added into and removed from the system frequently. This is unlike a world knowledge base (e.g., Wikidata) and commonsense knowledge base (e.g., ConceptNet) which only add new entities but (theoretically) never delete old entities. Introducing the *remove_entity* operation simplifies deleting triples when entities get removed.

We chose to adopt the Closed World Semantics over the Open World Semantics. In the Knowledge Graph community, the tradition is to adopt the Open World Semantics, because, in the beginning, the community focused on modelling world knowledge, where incompleteness is inevitable and not finding a triple in the KG is very likely due to missing rather than it being wrong. Thus Open World Semantics is more sensible. In the TREAT project, we aim to extract knowledge from the logs. Considering the logs only reveal limited information about the assisted system (some information might not be logged), in the beginning, we also adopted the Open World Semantics. But then

we realised that this may not be a good choice. Difference from world knowledge and commonsense knowledge:

- the knowledge in the telecom network scenarios is highly dynamic and variable. A piece of knowledge can be true but soon becomes false, and then turns true again. Suppose that such a triple `<User_1, connected, AMF>` means the knowledge that "the user `<User_1` is now connected to the network". As a user can disconnect and then connect to the network quickly over and over again, due to device problems or connection errors, the truth of the triple `<User_1, connected, AMF>` would correspondingly fluctuate between True and False.
- unbounded number of entities (e.g., devices, users, etc) can be created and then deleted. Under Open World Semantics, falsifying a triple can only be done by carefully crafting an ontology/schema so that the triple cannot be true in any interpretations. Yet this requires high-level human expertise. Otherwise, we need to associate a truth value to each triple, and when a triple becomes false, we explicitly mark it as 'false'. But this will (theoretically) require all triples to be stored, resulting in a monotonically increasing KB size.

Because of these issues brought about by the Open World Semantics, we decided to turn to the Closed World Semantics.

### 4.1.3 From Input to Output: The updating mechanism

Having defined the expected inputs and outputs, the TREAT system as a whole, can be viewed as a state transition function:

$$\mathcal{K} := \texttt{TREAT}_\Theta(\mathcal{K}, d)$$

*Continually updating the KB* is achieved by continually fetching logs from the assisted system and applying the $\texttt{TREAT}_\Theta$ function to the new logs $d$ and the KB $\mathcal{K}$.

If we unroll the formulae and express the time information:

$$\mathcal{K}^{(\tau+1)} = \texttt{TREAT}_\Theta(\mathcal{K}^{(\tau)}, S(\tau))$$

where $\mathcal{K}^{(\tau)}$ is the KB at time step $\tau$, $d$ and $S(\tau)$ is the set of logs arrived at time step $t$. The TREAT system, or the state transition function `TREAT` extracts new knowledge from $S(\tau)$, and transitions the states $\mathcal{K}^{(\tau)}$ to $\mathcal{K}^{(\tau+1)}$. Note that the time notations $\tau$ or $\tau+1$ merely indicate the time steps (versions) of the knowledge base.

The function TREAT is bonded with some extra information $\Theta$ needed to execute transitions, which may include inference rules, model parameters, and so forth. The main stuff included in $\Theta$ will be the log templates used to parse log messages (§4.2.1) and updating rules used to update the knowledge base (§4.3).

### 4.1.4 Key Procedures

In the previous section, I described the inputs and outputs of the TREAT system, and introduced the TREAT function. In this section, I describe the key procedures that are to be used in TREAT, which mainly involves 2 lower-level functions: Extract and Update.

The Extract function parses/converts the logs into structured log events. We define the function Extract as followed:

$$le = \texttt{Extract}_{\Omega}(d)$$

where $d$ is a set of logs, and $le$ is a set of extracted log events, as defined in §4.1.1.

These extracted log events contain important information about changes that occurred in the assisted system. The Update function will identify these changes and perform relevant updates upon the knowledge base.We define the Update function as:

$$\mathcal{K} := \texttt{Update}_{\Omega}(\mathcal{K}, le)$$

Therefore, the function TREAT can be represented by the above three lower-level functions:

$$\texttt{TREAT}_{\Omega}(\mathcal{K}, d) = \texttt{Update}_{\Omega}(\mathcal{K}, \texttt{Extract}_{\Omega}(d))$$

In the following sections, I will describe the essence of the TREAT function mentioned in §4.1.3, which is the mechanism to extract knowledge from the log stream and to perform updates on the knowledge base by the latest knowledge continually extracted from the subsequent logs.

The system architecture of the $\texttt{TREAT}_{\Theta}$ engine is shown in Figure 4.2. The TREAT engine extracts knowledge by converting the semi-structured log messages into structured log events with a library of log templates. The knowledge updating operations are issued by updating rules on new log events extracted. The knowledge is stored and updated in the Evolving Knowledge Base, which has been specified in §4.1.2, while the procedures of extracting knowledge and performing updates will be explained in §4.2.1 and §4.3 respectively.

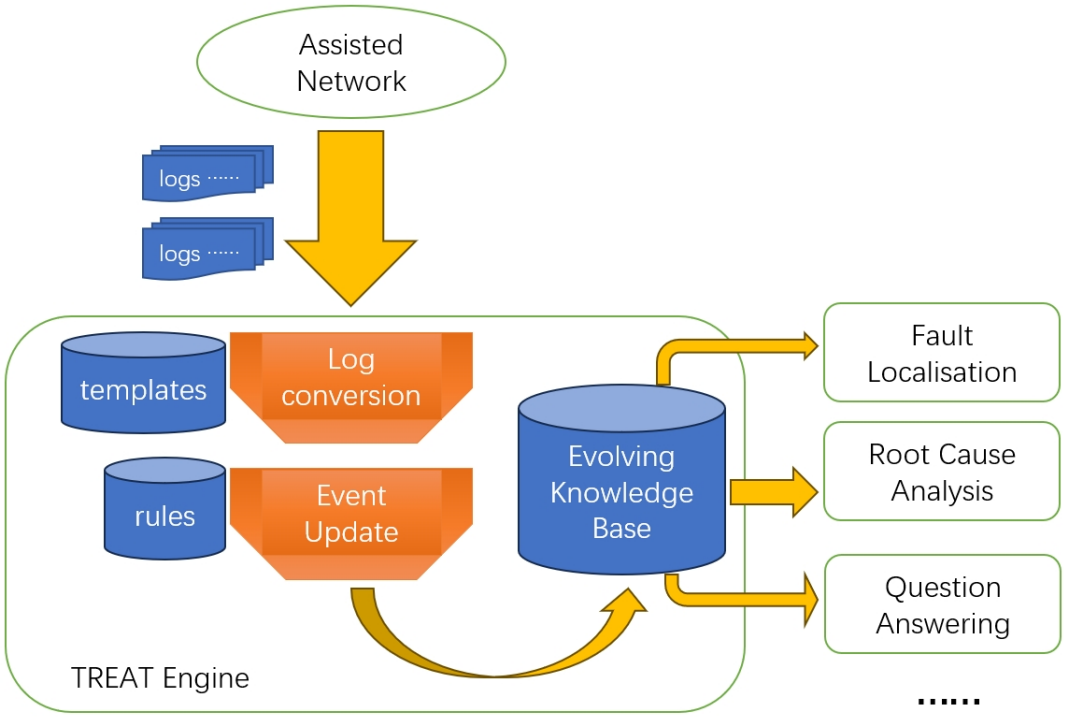



Figure 4.2: The architectural overview of the TREAT system

## 4.2 Extracting Knowledge

### 4.2.1 Template-Based Event Extraction

In §4.1.1, we defined the scope of logs that the TREAT system can process, which are the log entries generated by templates. Hence, our idea of extracting knowledge from the log stream is to first collect the templates, and then extract knowledge from the log stream by parsing the log entries into structured log events. This can be done easily by scanning the source code and finding out all the logging instructions. On the other hand, if we have no access to the source code, it is possible to gather a large set of system logs, and abduce the templates on the fly. There are several log template mining tools out there, for example, the Logparser toolkit [3].

The aim of log parsing is to identify the template that generated each log entry, and extract its important attributes and parameters. We chose the Drain[4] [81, 80] algorithm as the backbone of our knowledge extraction component, because of its popularity and effectiveness.

[3] https://github.com/logpai/logparser
[4] https://github.com/logpai/Drain3

#### 4.2.1.1 The Drain algorithm

The Drain algorithm is an online log parsing algorithm, meaning that it can do log matching and parsing as well as log template mining at the same time. For each new log entry, the algorithm will run an efficient search process with some heuristic rules, and try to match it with the existing templates. If no template is matched, the algorithm will create a new template, and continually tune the template as more new log entries come. Nevertheless, we can still use the Drain algorithm in a 2-phase manner: firstly, in the *preparation* phase we run the Drain algorithm on a corpus of collected logs to mine the templates. Then, in the *deployment* phase, we use the mined templates to parse the logs. This 2-phase manner is adopted in the TREAT system, as we need to make extra annotations for the templates before deployment, e.g., assigning names to template slots (§4.2.1.2) and generating updating rules for knowledge updating (§4.3).

I will very briefly introduce the Drain algorithm here, and for more details, please refer to the original papers [81, 80].

Finding the best match of a log message with a set of templates can be simply done by computing the similarity score between the log message and each template, and then choosing the template with the highest similarity score. However, this is not efficient as the number of templates increases. The Drain algorithm uses a heuristic search process to prune the search space, and only computes the similarity score for a small number of templates.

In general, the logs will be preprocessed and then roughly classified into several groups by some heuristic rules. Then, the logs will only need to be matched with the templates associated with that log group, instead of all the templates. As shown in Figure 4.3, the Drain algorithm is a 5-stage search procedure operated on a 5-layer directed acyclic graph (DAG) data structure. The DAG is like a search tree, whose branches of each layer heuristically prune the search space, and only in the last layer some results can be merged.

1. At the first stage, the raw logs are preprocessed by pre-defined regular expressions. Commonly known and widely used data items, such as IP addresses, email addresses, telephone numbers, and so on, will be extracted in this stage. This stage just performs simple operations but can greatly lower the difficulty of the extraction task.

2. At the second stage, the logs are categorised according to the log length, i.e., the number of tokens in the log message. The logs with the same length will be

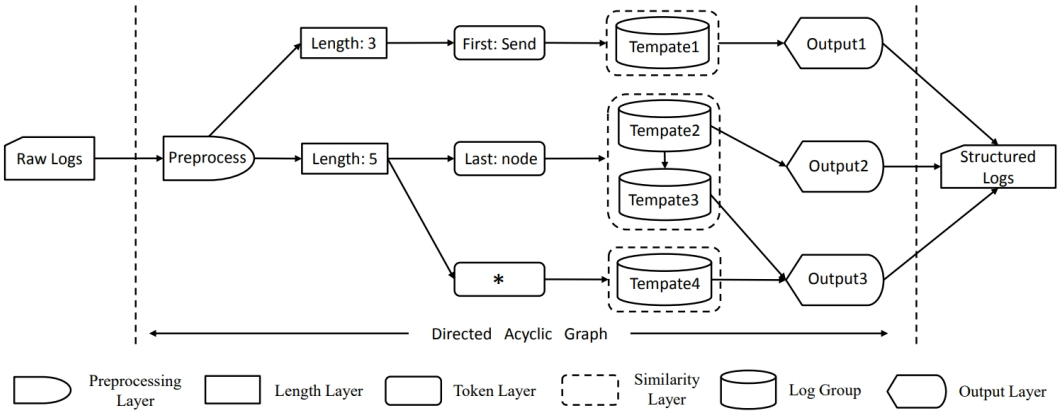



Figure 4.3: The procedural overview of the Drain workflow [80]

grouped together. This is due to the heuristics that the logs with very different lengths are likely to be generated by different templates.

3. At the third stage, the logs will be further categorised by their *split token*, which is the token that we choose as the signal of different log types. Usually the first or the last token would be selected. This is due to the heuristics that the logs with the same first/last tokens are probably generated by the same templates. However, if the first/last token contains digits or special characters, like `#^$'*+,/<=>@ )|~` then the second/second last token will be selected because digits and special characters are likely to be parameters instantiating a variable.

4. After 2 rounds of pruning, the logs are sorted into different groups, each of which is associated with some log templates. At the fourth stage, the logs are matched with the templates in the same group. The matching process is done by computing the similarity score between the log and each template. The similarity score is defined as the number of tokens that are the same between the log and the template, divided by the total number of non-variable tokens (i.e., non `<:*:>` tokens) in the log template:

$$simSeq = \frac{\sum_{i=1}^{n} \mathbb{I}(log(i), tmpl(i))}{n_c}$$

where $log(i)$ and $tmpl(i)$ represent the $i$th token of the log and the template, $n_c$ is the number of non-variable tokens, and the function $\mathbb{I}(x,y)$ is the indicator function

$$\mathbb{I}(x,y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases}$$

The similarity score is a number between 0 and 1, where 0 means the log and the template are entirely different, and 1 means the log and the template are regarded as the same. The template with the highest similarity score will be chosen as the best-matched template for the log. Yet this similarity score is expected to exceed a threshold $simThrd$. If the similarity score is lower than the threshold, the log will be regarded as one generated by a template that has not been discovered yet, and the Drain algorithm will create a new template for this log.

Inspecting the workflow diagram in Figure 4.3, the Drain algorithm runs depending on the DAG data structure as well as a bunch of internal states, which are the split tokens, the mined templates, and the thresholds of the similarity calculation. The Drain algorithm will dynamically update the DAG structure, and add and tune the internal states as more new logs come.

**Add and Tune a Log Template** When a new log message comes and it does not match any existing log template, the Drain algorithm will add a new log template to the system. The new log template will be initialised with that log message and no variables/slots. when the next log message traverses the DAG and matches this new log template, the Drain algorithm will try to tune the log template by adding new variables/slots to it. Specifically, Drain scans the message and template token by token, and replaces the token of the template with the wildcard `<:*:>` symbol in the position where the tokens of the template and the message are different.

**Initialise and Tune the Threshold** Each log template is attached with a similarity threshold, so that the log message could be regarded as generated by the template only if the similarity score exceeds the threshold. Hence, this threshold is an important parameter. Instead of setting a fixed threshold, the Drain algorithm will dynamically tune the threshold as more new logs come. By default, the initial threshold is set by an empirical formula:

$$simThrd_{init} = 0.5 * \frac{seqLen - digLen}{seqLen}$$

where $seqLen$ represents the number of all tokens in the initial log template, and $digLen$ represents the number of tokens containing digits in the initial log template. This is a heuristic estimation of the lower-bound ratio of the number of constant tokens to the total number of tokens in log templates. This is quite a low threshold, encouraging matches at the beginning and tuning the template. Each time the template gets tuned

(a <:*:> is introduced), the threshold is increased a bit. Specifically, the adjusted threshold is computed by the following formula:

$$simThrd = min(1, simThrd_{init} + 0.5 * log_{base}(\eta + 1))$$

where $base = max(2, digLen + 1)$ and $\eta$ is the number of tokens that have been replaced by the wildcard. The intuition behind this formula is that the more tokens have been replaced by the wildcard, the more likely the log template is to be a generalised template, so the increment of the threshold each time should be smaller.

#### 4.2.1.2 Log parsing in TREAT

Having explained the Drain algorithm, we now describe how we use the Drain algorithm to extract knowledge from the log stream. Applying Drain is straightforward, yet there are some extra jobs to do.

The first job is to convert the time information to a standard format and annotate the extracted log events with additional information (log ID, log type ID, the source of the log, etc). This job can be easily done by simply parsing the time information to a standard Unix timestamp, and then assigning the log type ID and giving a unique random log ID to the log event. Most modern programming language provides utilities for converting time and generating random unique identifiers in their standard libraries.

The second and the more tricky job is to assign the slots of the log template with consistent names, meaning that the same entity across different log templates should be assigned with the same name. During the preparation phase, we mine the log templates from a corpus of collected logs by running the Drain algorithm. Note that we only see wildcards <:*:> at the log templates mined by Drain, but we don't know what the wildcards mean. To make use of the extracted information, We need to annotate the wildcards with meaningful and consistent names. For example, in the below log template

```
[<:*:>] NF registered
```

This template indicates an event that a Network Function instance (the building block of a 5G network system) was registered. The wildcard <:*:> means the ID of a Network Function instance, so we can annotate it with the name `nf_instance_id`. In fact, we can assign arbitrary names to the slot, such as `param_101`, but we need to make sure that the names are consistent across all the templates. For example, here is another log template

```
[<:*:>] NF No heartbeat
```

This template indicates an event that a Network Function instance (the building block of a 5G network system) was lost. The wildcard `<:*:>` means the same kind of entity as the previous template, so we need to assign the same name to it, i.e., then the name of this slot should be `param_101` as well, instead of other names like `param_102`.

From the NLP perspective, this could be viewed as a task of Named Entity Recognition and Entity Typing. But we can simply ask human experts to annotate the wildcards with meaningful names. This is a bit *ad-hoc*, but it is a one-time job.

## 4.3 Event-Based Updating Mechanism

Once the log entries are parsed into structured events, they tell us what was happening at a certain time point inside the system, and we can use these events to update the knowledge base.

To update the knowledge base with the log events, we need to know which log may cause what changes. To make things clearer, we can compare our logs with database logs. Traditional RDBMS systems [174] record SQL queries and the corresponding results in logs. SQL queries can be roughly classified into SELECT, INSERT, UPDATE, DELETE, and a few others. SELECT queries are equivalent to observing the system and don't modify system states, while INSERT, UPDATE, and DELETE queries are equivalent to changing events and changing actions to the system states.

The relations between SQL logs and database updates are straightforward, yet the relations between our logs and software system updates are not that simple. After all, there are no clear standardised keywords in the log indicating what changes occurred in the system. Hence, we introduce *Updating rules* to explicitly connect log events and updating operations to be performed on the KB. For simplicity, an updating rule can be formalised as

$$le_i \rightarrow ue_1, ..., ue_k$$

where $le_i$ is a log event and $ue_1, ..., ue_k$ are update events[5]. The meaning of this rule is: when receiving a log event, and the type of this log event is $le_i$, then update operations $ue_1, ...ue_k$ should be performed upon the knowledge base. All the allowed update operations are those defined in the KB model stated in §4.1.2, which are *add* (adding

[5]Please note that updating rules are different things from inferences rules used in logical reasoning, such as Logic Programming

a fact a the KB), *delete* (deleting a fact from the KB), and *remove_entity* (remove an entity and all the facts involving this entity from the KB). For example, the following updating rule

```
le1($timestamp, $component, $level, $nf_instance_id, $nf_type)
    ->
    add(($nf_instance_id, type, $nf_type)),
    add(($nf_instance_id, type, network_function))
```

matches the log event type `le1`, and issues two *add* operations. The symbols starting with the "$" sign are variables to be instantiated at runtime. Other symbols that do not start with the "$" sign predicates are concepts of the knowledge base. Note that all the variables used in the updating events should appear in the log event.

The above updating rule is the simplest form, which only allows one log event as the precondition. This significantly limits the ability to recognise patterns of change. More potential features can be added to the rule, for example, the time interval between two log events, unordered sequence of log events, etc. There is a whole body of research called Complex Event Processing (CEP) [45, 188] that deals with this kind of problem, though, we will not go into the details of CEP here. In this research project, we will keep using the simplest form of updating rules, which is sufficient for a proof-of-concept system. What's more, enabling more features of updating rules demands a more sophisticated event processing engine, which is an engineering difficulty. How to trade these off should be left to software engineers and product managers for consideration.

How to obtain these updating rules is a key challenge. At present, we produce these rules by human experts, but we are also looking for an automated solution for future work.

Below is a simple example demonstrating how to update the KB according to the log events and updating rules.

### An example

Here is an example log entry from the Open5GS system[6]:

```
02/24 08:51:02.485: [sbi] INFO: [nf_5dd8edb8_b420_41ed] NF
    registered
```

[6] https://open5gs.org

It matched the following log template:

```
[<:nf_instance_id:>] NF registered
```

By parsing the log entry using the methods in §4.2.1, we can obtain the following log event:

```
{
  "log_type_id": "1",
  "log_id": "evt_b77d5cfd-7aa26-4905-bad4-53ec973a4adb",
  "timestamp": 1677228662485,
  "component": "sbi",
  "severity": "INFO",
  "nf_instance_id": "nf_5dd8edb8_b420_41ed",
  "log_source": "amf",
}
```

where the `log_type_id` is the ID of the log template, `log_id` is the randomly created unique ID of the log event, and `log_source` is the source part of the log entry (did not show in the log itself). This log event will be matched with such an updating rule:

```
le1($timestamp, $component, $level, $nf_instance_id, $nf_type)
    ->
   add(($nf_instance_id, type, $nf_type)),
   add(($nf_instance_id, type, network_function))
```

meaning that the log event will issue two update operations upon the KB

```
add((nf_5dd8edb8_b420_41ed, type, network_function)),
add((nf_5dd8edb8_b420_41ed, type, amf)),
```

The two *add* operations add two facts to the KB: the NF instance `nf_5dd8edb8_b420_41ed` is an entity belonging to the type `network_function` (the fundamental building block of the 5G system), as well as the type `amf`.

Here is another example log entry :

```
02/28 08:00:25.724: [sbi] INFO: [nf_5dd8edb8_b420_41ed] NF No
   heartbeat
```

It matches the following log template:

```
[<:nf_instance_id:>] NF No heartbeat
```

By parsing the log entry we can obtain the following log event:

```
{
    "log_type_id": "35",
    "log_id": "evt_a5fd5cfd-8ef6-4905-bad4-53ec973a8adb",
    "timestamp": 1677571225724,
    "component": "sbi",
    "severity": "INFO",
    "nf_instance_id": "nf_5dd8edb8_b420_41ed",
    "log_source": "amf",
}
```

which will be matched with the updating rule:

```
le35($timestamp, $component, $level, $nf_instance_id,
    $log_source) ->
    delete(($log_source, available, true)),
    remove_entity(($nf_instance_id)) .
```

meaning that the log event will issue two update operations upon the KB

```
    delete((amf, available, true)),
    remove_entity((nf_5dd8edb8_b420_41ed)) .
```

The second operation falsifies the fact that the AMF function is available, and the last operation removes the NF instance `nf_5dd8edb8_b420_41ed` from the KB, together with all the facts involving the entity, including the two facts added to the KB previously.

### 4.3.1 Limitation

As mentioned above, it is a challenging problem to obtain the rules that perform update operations on the entity KB from the log events. At present, we hand-crafted these rules. However, it demands human expertise to write these rules to map the log event types to updating operations, though the number of log event types should be limited. In our experiment §5.1 conducted on the Open5GS system[7], we collected over a hundred thousand log entries, but only mined 102 log event types.

Secondly, the knowledge that can be expressed in the system logs, and therefore the knowledge base is restricted by the available log types/templates. We can think that

[7] `https://github.com/open5gs/open5gs`

these log types are like the schema of a database. Each log type defines a relation/table, and all the possible data we can record by that database.

**How general/limited is this framework?** The framework should be applicable to all kinds of software systems which continually produce a stream of logs as defined in §4.1.1. Nevertheless, each time deploying the framework to a new assisted software system will require human labour to collect logs, prepare regular expressions for pre-processing, inspect and revise templates, and write event-based updating rules. Among these human labour, writing event-based updating rules requires high-level expertise. Within a software system, the log templates are finite, meaning that there is a limited number of log types. If the number of log templates in a software system is small, like the Open5GS test bed, then it will only need a trivial amount of expert labour to write the updating rules, but the knowledge base can merely express a limited amount of knowledge. But if the number of log templates in a software system is huge, the knowledge base can express a lot of various knowledge but it will also take a lot of expert labour to identify useful log events and devise the updating rules.

### 4.3.2 Summary

This chapter covered the various aspects of a systematic framework called TREAT, which takes as input a stream of logs continually produced by a large software system, extracts the knowledge from the logs, captures the changes occurring within the system and outputs an evolving knowledge base to reflect as faithfully as possible the internal states of that software system. I formally described the expected inputs and outputs. I also depicted the key procedures, including knowledge extraction via log template mining and parsing and event-based updating via updating rules. In Chapter 5, I will present an application of the TREAT framework to fault localisation, a typical downstream O&M task in software systems. This downstream application will also serve as an indirect evaluation of the whole TREAT framework.

# Chapter 5

# Applying TREAT to Fault Localisation

We have been discussing how to construct a KB from log, yet the ultimate target is to facilitate various O&M tasks. That means whether our extracted KB is good or bad depends on how well it can support these tasks, typically fault localisation. In this chapter, we are going to show how to apply the symbolic knowledge extracted by the TREAT system in addressing the fault localisation task. We will formulate the problem of fault localisation within the framework of Logic Programming (LP) [97, 114] and Inductive Logic Programming (ILP) [130, 44], through which we can make use of the symbolic/logical knowledge in the TREAT KB. Finally, I will present an experiment conducted on a test bed 5G system, where I injected system faults into the components and demonstrate that the TREAT-powered KB can facilitate accurate prediction of the injected faults.

### 5.0.1 Some Background

LP is a programming paradigm based on deductive formal logic. Problems are expressed as sets of facts and inference rules, and the solution is derived by means of logical inference. ILP is the somewhat "inverse" version of LP. It learns a set of inference rules from provided examples. ILP is regarded as a machine learning technology, but it combines with logic tightly, making itself demand less training data and computation than other machine learning methods, as well as being more explainable, and thus fits in the scenario of autonomous networks. Both LP and ILP have a long history, and there are many books and papers elaborating on them. Here we only give briefly some essential concepts of these techniques.

#### 5.0.1.1 Logic Programming using Prolog

The most significant benefit offered by LP is perhaps its declarativeness, whose aim is that the programmer only needs to specify what the problem is, and the problem solver, i.e. the inference engine of logical programming in this context, will figure out how to solve it. To specify a computational problem in LP is to specify a set of facts and inference rules that capture the knowledge of the problem domain, and queries that should be proved or satisfied by the inference engine.

Facts are the basic units of knowledge, and inference rules are the logical relations between facts. In most LP languages, typically the most famous Prolog [39, 204], facts and inference rules are written in the form of Horn clauses [84]. Simply speaking, in Prolog, a **inference rule** is written as $h \leftarrow b_1, ..., b_n$ where $h$ and $b_1, ..., b_n$ are logical atoms $p(t_1, ..., t_n)$ consisting of an $n$-ary predicate $p$ and $n$ arguments $t_i$. $h$ is called the *head* of the inference rule, and $b_1, ..., b_n$ are called the *body* of the inference rule. A **fact** is an inference rule $h$ without a body, meaning that $h$ is always true (thus a fact). On the contrary, a head $h$ with a body $b_1, ..., b_n$ is true if the body is true.

The semantics of an inference rule in Prolog reveals the difference between logic programming and mathematical logic. For example, in first-order logic, an implication $p \leftarrow q$ means that $p$ is true if $q$ is true, but it does not mean that $p$ is false if $q$ is false. However, in Prolog, a logical term $h$ is considered true if and only if it is a fact or if it can be proved by the inference engine given the existing facts and inference rules. This principle is called *negation as failure* [38], a kind of Closed-World Assumption [163, 59]. Hence, an inference rule $p \leftarrow q$ means that $p$ is true if $q$ is true, and $p$ is false if $q$ is false, when there is no other rule entailing $p$.

The following code listing is a simple and widely used example LP program that specifies the kinship in a family.

Listing 5.1: An example of a Prolog program

```
% facts
parent(john, mary).
parent(john, tom).
parent(mary, ann).
parent(mary, tom).
parent(tom, bob).

% inference rules
```

```
ancestor(X, Y) :- parent(X, Y).
% the notation ":-" symbolise the left arrow
ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).
```

The first part of the program specifies the factual parent-child relations in the family. Note that by convention the predicates and constants start with lowercase letters. The second part is the inference rules, which specify the ancestor-descendant relations in the family. Note that by convention the variables start with uppercase letters. The first inference rule says that if $X$ is the parent of $Y$, then $X$ is the ancestor of $Y$. The second inference rule recursively defines that if $X$ is the parent of $Z$, and $Z$ is the ancestor of $Y$, then $X$ is the ancestor of $Y$. The syntax of Prolog is much richer than this, but we do not need to go into details here. Readers can refer to the books [39, 204] for more information.

**Satisfying a Goal** is the main approach to making use of the specified knowledge in an LP program. If we view a Prolog program as a database, then a goal is a query to that database. Simply speaking, a goal is a logical term with some variables, and *satisfying* a goal is finding the constants that can instantiate the variables and make the goal true. For example, `ancestor(X, bob).` is a goal, and satisfying this goal is to find out all the constants that can instantiate the variable `X` and make the goal true. In this case, the constants are `tom`, `john`, and `mary`. Thus, in the Prolog interpreter we can see the following results:

```
?- ancestor(X, bob). % the notation "?-" is the prompt
   of the Prolog interpreter
X = tom ;
X = john ;
X = mary ;
false.
```

The last line `false` means that there are no more solutions to the goal. A goal can have 0 variables, such as `ancestor(john, bob).` Satisfying such a goal is equivalent to proving whether the goal is true or false, In this case, the Prolog interpreter will try to prove the goal by searching the facts in the program or establishing proof with the given facts and inference rules. Examples are:

```
?- parent(mary, ann).
true.
```

```
?- parent(mary, anne).
false.

?- ancestor(john, bob).
true.

?- ancestor(bob, john).
false.
```

#### 5.0.1.2 Inductive Logic Programming

ILP is a machine learning technology that imposes strong inductive bias preferring the patterns that can be represented by a specified formal language. If we say that LP mimics the deductive thinking of human beings which derives specific cases (existing facts and inference rules) from general cases (new facts), then ILP mimics inductive thinking which comes up with general cases (new inference rules) from specific cases (existing facts and inference rules). ILP also reuses the concepts of LP. The basic elements are facts and inference rules as well, and they are all written in the form of Horn clauses, as introduced in §5.0.1.1.

Formally speaking, a **learning problem** in ILP is a tuple $(B, E^+, E^-)$, where $B$ is the background knowledge, $E^+$ is the set of positive examples, and $E^-$ is the set of negative examples. The objective of ILP is to learn a set of inference rules $H$ that together with the background knowledge $B$, can cover the positive examples and not the negative examples. More concretely, the background knowledge $B$ is, like an LP program, a set of facts and inference rules.

There are different **learning settings** [51], depending on which the representation of positive and negative examples can vary. The most common learning setting is **learning from entailments**, where the positive and negative examples are all single logical atoms without variables (called *grounded atoms*). In this case, the positive examples are the logical atoms that can be entailed by the background knowledge, and the negative examples are the logical atoms that cannot be entailed by the background knowledge.

$$\forall e \in E^+ \; R \cup B \vdash e$$

$$\forall e \in E^- \; R \cup B \nvdash e$$

$$\forall e \in Fault^{+}\ R \cup KB \vdash e$$

$$\forall e \in Fault^{-}\ R \cup KB \nvdash e$$

Below is a simple demonstrative ILP problem. Suppose we have the following background knowledge:

```
% background
parent(john, mary).
parent(john, tom).
parent(mary, ann).
parent(mary, tom).
parent(tom, bob).
```

and here are the positive and negative examples:

```
% positives                  % negatives
ancestor(john, mary).        ancestor(mary, john).
ancestor(john, tom).         ancestor(tom, john).
ancestor(mary, ann).         ancestor(ann, mary).
ancestor(mary, tom).
```

Feeding this background knowledge and learning examples to an ILP engine, the following inference rule might be induced:

```
ancestor(X, Y) :- parent(X, Y).
```

There are other learning settings, like **learning from satisfiability** [52], **learning from partial interpretations** [70], **learning from interpretation transitions** [86], and so forth, but we do not need to go into details here.

**ILP by ProbFOIL** There are many ILP systems, such as FOIL [156], Progol [131], and so forth. Here we use ProbFOIL [53]. It is based on FOIL, which is an induction system that learns from entailments as well. The difference is that FOIL mechanism is deterministic, while ProbFOIL uses a probabilistic approach to learn inference rules, adding noise toleration capabilities to the positive and negative examples.

### 5.0.2 Formulating Fault Localisation as LP

Fault localisation is a typical task in Operation and Maintenance [207]. Usually, it refers to identifying the component/subsystem/part that is responsible for a fault, so

that we can fix the system by replacing or simply restarting that faulty component.

Suppose a software system can be decomposed into a set of components $Y = \{y_1, ..., y_n\}$ (which is a reasonable assumption because modularisation/componentisation is a chief principle of software engineering). We can write a fault localisation program as a function $y = fl(x)$ that takes some input and returns the faulty component $y \in Y$ that is responsible for the fault. Depending on the underlying algorithm, the input $x$ can vary. In our scenario, the input is the snapshot of the TREAT-powered KB at the time $\tau$ when the fault happened, i.e., $y = fl(K^{[\tau]})$.

As we shall use LP techniques to address the fault localisation problem, we need to formulate the problem to be a logic program. That is, we need to transform the input $K^{[\tau]}$ to be a set of facts and inference rules in Horn Clause format, and then feed the program to a Prolog engine for querying the faulty component. This is what the function $fl$ does. Recall in §4.1 that the knowledge in the TREAT KB is encoded as a set of triples. Thus, converting the triples into Horn logic is straightforward: a triple $(s, p, o)$ can be directly converted to a logical term $p(s, o)$. Then, loading these facts, as well as the inference rules into a Prolog engine, we can query the faulty component by means of Prolog's built-in inference engine using a special predicate `faulty`. For instance,

```
?- faulty(Y).
Y = c1.
```

Readers may wonder where this seemingly magical predicate `faulty` comes from and how to obtain the inference rules to enable logical reasoning. We obtain the inference rules that infer `faulty` components by the ILP techniques. That is, we can first collect a set of fault cases as examples and induce the inference rules from these fault cases. In our scenario, the positive examples are the faulty KB snapshots together with their corresponding correct faulty components, while the negative examples are the faulty KB snapshots with the incorrectly identified faulty components., which can be generated by substituting the correct ones with new components.

### 5.0.3 Automatically Obtaining Inference Rules by ILP

Taking the snapshot of the knowledge base at the time points when the faults occur, we can create a dataset consisting of $(KB_i, fault_i)$ pairs, where $i$ denotes the time point of the fault occurred.

**KB Snapshot** Using the log extraction method described in §4.2.1 and event-based updating method mentioned in §4.3, we can obtain an evolving knowledge base $\mathcal{K}$. Taking the snapshot of the knowledge base at the time points when we injected the faults, we can create a dataset consisting of $(KB_i, fault_i)$ pairs, where $i$ denotes the time point of the fault injected.

We then formulate the fault localisation problem as an ILP problem. That is, we need to convert the $(KB_i, fault_i)$ pairs into an ILP problem $(B, E^+, E^-)$.

1. We find out the facts that remain unchanged in all the snapshot KBs $KB_i$ and put them into $B$. This is the most general background knowledge, like the type declarations.

2. For the rest of the facts in every snapshot KB $KB_i$, we rectify them, i.e., insert a time argument to highlight the time point when the fact is true. For example, a fact `p(s1, o1)` in $KB_1$ will be rectified to be `p(case1, s1, o1)`. Afterwards, we put all these rectified facts into $B$.

3. Finally, for each fault $fault_i$, we put it into the positive examples $E^+$ `faulty (snapshot_i, fault_i)`. As it has been shown that a mixture of positive and negative examples lead to better learning results than positive examples alone [132], We also generate some negative examples $E^-$, inspired by the paradigm of contrastive learning [37], by randomly selecting some other faults and put them into $E^-$.

## An example of ILP formulation

Suppose we have two snapshots of the knowledge base, $KB_1$ and $KB_2$, and the faults that occurred are $fault_1$ and $fault_2$. The following code listing shows the general background knowledge and the specific background knowledge for the two cases.

```
% general background knowledge, specifying
% the types of the components that should be available
nf(amf).
nf(smf).
nf(udm).
nf(pcf).
nf(udr).
nf(ausf).
```

```
nf(nssf).
nf(scp).

inuse_nf(Case, NF) :- nf(NF), typeOf(Case, Ent, NF).

% specific background knowledge for case 1
typeOf(case1, nf_d4a8bef4_0a1b_41ee, net_func) .
typeOf(case1, nf_c6edf27a_9d8d_41ed, net_func) .
typeOf(case1, nf_d48324f2_0a19_41ee, net_func) .
typeOf(case1, nf_3c68a5c8_0a1b_41ee, net_func) .
typeOf(case1, nf_aae405d8_0a1b_41ee, ausf) .
typeOf(case1, nf_e5b4272e_0a1b_41ee, pcf) .
typeOf(case1, nf_c6edf27a_9d8d_41ed, scp) .
typeOf(case1, nf_6b25cdd6_3550_41ee, net_func) .
typeOf(case1, nf_e5b4272e_0a1b_41ee, net_func) .
typeOf(case1, nf_d48324f2_0a19_41ee, udm) .
typeOf(case1, nf_3c68a5c8_0a1b_41ee, smf) .
typeOf(case1, nf_aae405d8_0a1b_41ee, net_func) .
typeOf(case1, nf_6b25cdd6_3550_41ee, amf) .
typeOf(case1, nf_cdd54ba6_9d8d_41ed, net_func) .
typeOf(case1, nf_d4a8bef4_0a1b_41ee, udr) .

% specific background knowledge for case 2
typeOf(case2, nf_d4a8bef4_0a1b_41ee, net_func) .
typeOf(case2, nf_c6edf27a_9d8d_41ed, net_func) .
typeOf(case2, nf_d48324f2_0a19_41ee, net_func) .
typeOf(case2, nf_3c68a5c8_0a1b_41ee, net_func) .
typeOf(case2, nf_aae405d8_0a1b_41ee, ausf) .
typeOf(case2, nf_e5b4272e_0a1b_41ee, pcf) .
typeOf(case2, nf_c6edf27a_9d8d_41ed, scp) .
typeOf(case2, nf_c5e117bc_3550_41ee, nssf) .
typeOf(case2, nf_e5b4272e_0a1b_41ee, net_func) .
typeOf(case2, nf_c5e117bc_3550_41ee, net_func) .
typeOf(case2, nf_d48324f2_0a19_41ee, udm) .
typeOf(case2, nf_3c68a5c8_0a1b_41ee, smf) .
```

```
typeOf(case2, nf_aae405d8_0a1b_41ee, net_func) .
typeOf(case2, nf_cdd54ba6_9d8d_41ed, net_func) .
typeOf(case2, nf_d4a8bef4_0a1b_41ee, udr) .
```

The following code listing shows the examples for the two cases.

```
% positive and negative learning examples for case 1
positive(faulty(case1, nssf)). % the true fault of case 1 is
   nf3
negative(faulty(case1, smf)). % the generated negative fault of
    case 1
negative(faulty(case1, pcf)).
negative(faulty(case1, ausf)).

% positive and negative learning examples for case 2
positive(faulty(case2, amf)).
negative(faulty(case2, udr)).
negative(faulty(case2, udm)).
negative(faulty(case2, nssf)).
```

Feeding this background knowledge and learning examples to an ILP engine, the following inference rule will be induced:

```
faulty(A,B) :- nf(B), \+inuse_nf(A,B).
```

meaning that if `B` is a network function and it is not in use in case `A`, then it is the faulty component of case `A`. Once we induce these inference rules, we can use them to infer the faulty component of new cases by means of Logic Programming. Figure 5.1 is an example of how to use the induced inference rules to infer the faulty component of a new case (I assigned it id "999" so that it would not clash with previous training and test cases). In the file `temp.pl`, I included all the background knowledge, case-specific facts, and the induced inference rules, then ran the Prolog interpreter to infer the faulty component of this new case.

## 5.1 Evaluation

In this section, we provide experimental evidence to support the effectiveness of our approach. The experiment is to evaluate the effectiveness of the TREAT system in

```
(logkg-rca2) D:\TREAT\logkg-fl\data\exp1\fl\test>cat temp.pl
typeOf(case999,nf_c6edf27a_9d8d_41ed_96db_73930f919161, nrf) .
typeOf(case999,nf_864b66ac_3569_41ee_8fac_7f2339d9a9b0, nrf) .
typeOf(case999,nf_c6edf27a_9d8d_41ed_96db_73930f919161, scp) .
typeOf(case999,nf_cdd54ba6_9d8d_41ed_a30a_45676ecf6589, nrf) .
typeOf(case999,nf_7357ad38_3566_41ee_8b3b_55200db0a294, udr) .
typeOf(case999,nf_7357ad38_3566_41ee_8b3b_55200db0a294, nrf) .
typeOf(case999,nf_89ce4084_3568_41ee_a169_0598f8d6b34f, nrf) .
typeOf(case999,nf_a0e74ecc_3569_41ee_a561_21c4794bc1ef, nrf) .
typeOf(case999,nf_b9ddadc8_3568_41ee_b59c_2fee4387a34e, nrf) .
typeOf(case999,nf_a0e74ecc_3569_41ee_a561_21c4794bc1ef, ausf) .
typeOf(case999,nf_3d40bc78_3569_41ee_9994_530712d94060, smf) .
typeOf(case999,nf_3d40bc78_3569_41ee_9994_530712d94060, nrf) .
typeOf(case999,nf_89ce4084_3568_41ee_a169_0598f8d6b34f, amf) .
typeOf(case999,nf_864b66ac_3569_41ee_8fac_7f2339d9a9b0, udm) .
typeOf(case999,nf_b9ddadc8_3568_41ee_b59c_2fee4387a34e, pcf) .

nf(amf).
nf(smf).
nf(udm).
nf(pcf).
nf(udr).
nf(ausf).
nf(nssf).
nf(scp).

% if there is a statement asserting an the NF type of an entity, then in this case
% this NF is in use
inuse_nf(Case, NF) :- typeOf(Case, _, NF), nf(NF).
faulty(A,B)  :-  \+inuse_nf(A,B), nf(B).

faulty(A,B)  :- nf(B),  \+inuse_nf(A,B).

(logkg-rca2) D:\TREAT\logkg-fl\data\exp1\fl\test>swipl temp.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 9.1.18)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

1 ?- faulty(case999, X).
X = nssf .
```

Figure 5.1: An example of using the induced inference rule to infer the faulty component in a new case by Logic Programming.

facilitating the fault localisation task. We first describe the experimental setup, including the deployed test bed and the experiment steps, and then present the experimental results. In brief, the TREAT-powered fault localisation methods achieved 90% prediction accuracy in our experiment, showing that the TREAT system can continually extract high-quality knowledge from the logs and capture the changes of the assisted system.

### 5.1.1 Open5GS Test Bed System

To conduct the experiment, we deployed Open5GS, a simple test bed system, as the assisted system. We used Open5GS[1], an open-source implementation of a 5G System conforming to the 5G standard Release-17[2]. A 5G System (5GS) can be divided into three main parts as shown in Figure 5.2[3]

1. User Equipment(UE)
2. 5G Radio Access Network(5G-RAN)
3. 5G Core Network (5GC)


Figure 5.2: The 5G System big picture

UEs are the 5G-enabled devices, typically a 5G smartphone. 5G-RAN is a type of network infrastructure used commonly for mobile networks that consists of radio base stations with large antennas. An RAN wirelessly connects user equipment to a core network. 5G Core network facilitates main data forwarding and other various Network Functions, such as session management, authentication, policy control, data storage etc.

We deployed our own Open5GS, i.e., the virtual UE, 5G-RAN, and 5GC by means of docker in a Linux/Ubuntu server. For reference purpose, here is the list of related containers running in the test bed. Apart from the Open5GS microservices, there are some auxiliary microservices, such as the database (MongoDB), etc.

```
$ # Display the docker containers information of the Open5GS test bed
```

[1] https://github.com/open5gs/open5gs
[2] https://www.3gpp.org/specifications-technologies/releases/release-17
[3] https://www.3gpp.org/wiki/index.php?id=182

```
$ docker ps --format "table {{.ID}}\t{{.Names}}\t{{.Image}}"
CONTAINER ID   NAMES          IMAGE
28b2025a05dd   nr_ue          docker_ueransim
c6164c44d8ca   nr_gnb         docker_ueransim
4e5896d10082   amf            docker_open5gs
9fcb45ba2458   upf            docker_open5gs
1548c8b5ef5f   bsf            docker_open5gs
cac9fd79b8cd   webui          docker_open5gs
0b18d8386cfd   udr            docker_open5gs
f4c19b067ba0   nssf           docker_open5gs
f426d8fc859e   udm            docker_open5gs
cccd561a338b   ausf           docker_open5gs
658469b647ad   pcf            docker_open5gs
5a4bfb697552   smf            docker_open5gs
591d3d080498   mongo          docker_mongo
8c953da17d42   scp            docker_open5gs
fe9f8ffc1208   nrf            docker_open5gs
```

Since the whole 5G system is run by docker, the simplest way to collect logs is to use the logging command supported by docker. Below are the logs produced by AMF in the early stage:

```
$ docker logs amf
Deploying component: 'amf-1'
Open5GS daemon v2.5.5-48-gfa5b2fe

01/24 10:36:07.280: [app] INFO: Configuration: '/open5gs/
   install/etc/open5gs/amf.yaml' (../lib/app/ogs-init.c:126)
01/24 10:36:07.280: [app] INFO: File Logging: '/open5gs/install
   /var/log/open5gs/amf.log' (../lib/app/ogs-init.c:129)
01/24 10:36:07.286: [metrics] INFO: metrics_server() [http
   ://172.20.0.210]:9091 (../lib/metrics/prometheus/context.c
   :510)
01/24 10:36:07.286: [sbi] INFO: NF Service [namf-comm] (../lib/
   sbi/context.c:1334)
01/24 10:36:07.286: [sbi] INFO: nghttp2_server() [http
```

```
   ://172.20.0.210]:7777 (../lib/sbi/nghttp2-server.c:237)
01/24 10:36:07.286: [amf] INFO: ngap_server()
   [172.20.0.210]:38412 (../src/amf/ngap-sctp.c:61)
01/24 10:36:07.286: [sctp] INFO: AMF initialize...done (../src/
   amf/app.c:33)
01/24 10:36:07.288: [sbi] INFO: [e8ff155e-9bd2-41ed-9a11
   -0158331531c3] NF registered [Heartbeat:10s] (../lib/sbi/nf
   -sm.c:214)
01/24 10:36:16.322: [sbi] INFO: [ee622a4a-9bd2-41ed-989a-
   bd451676dded] (NRF-notify) NF registered (../lib/sbi/nnrf-
   handler.c:632)
01/24 10:36:16.322: [sbi] INFO: [ee622a4a-9bd2-41ed-989a-
   bd451676dded] (NRF-notify) NF Profile updated (../lib/sbi/
   nnrf-handler.c:642)
```

The most significant advantage of deploying a self-owned test bed is that we can freely perform various operations on the system and observe the behaviours, and thus collect the logs.

The common operation of normal users with 5G is accessing a website through the 5G network. Below is a simple example. I used the UE (sit in container 28b2025a05dd) to ping a website on the Internet.

```
$ docker exec -it nr_ue ping -I uesimtun0 www.huawei.com
PING www.huawei.com.web3.hwgslb.com (10.3.42.32) from
   192.168.100.2 uesimtun0: 56(84) bytes of data.
64 bytes from 10.3.42.32 (10.3.42.32): icmp_seq=1 ttl=243 time
   =176 ms
64 bytes from 10.3.42.32 (10.3.42.32): icmp_seq=2 ttl=243 time
   =176 ms
64 bytes from 10.3.42.32 (10.3.42.32): icmp_seq=3 ttl=243 time
   =176 ms
64 bytes from 10.3.42.32 (10.3.42.32): icmp_seq=4 ttl=243 time
   =176 ms
64 bytes from 10.3.42.32 (10.3.42.32): icmp_seq=5 ttl=243 time
   =176 ms
64 bytes from 10.3.42.32 (10.3.42.32): icmp_seq=6 ttl=243 time
```

```
   =176 ms
64 bytes from 10.3.42.32 (10.3.42.32): icmp_seq=7 ttl=243 time
   =176 ms
^C
--- www.huawei.com.web3.hwgslb.com ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6009ms
rtt min/avg/max/mdev = 175.735/176.012/176.351/0.180 ms
```

Here is what happened: The UE established the connection with the 5G Core with the interface `uesimtun0`. When it pings a website, the packets are routed through the interface `uesimtun0` to the Internet via the Open5GC 5G Core network.

An administrator can do almost whatever it wants to do with the network, including destroying the whole network! Yet the most common operations may be to change some configuration and then apply the new settings. In the microservice system, this is normally done by editing the config file and then restarting the container. Below are some common commands to restart the network functions.

```
$ docker stop ausf # stop the AUSF function
$ docker start ausf # start the AUSF function
$ docker restart ausf # restart the AUSF function
```

As for fault injection, we used Pumba[4], which is a chaos testing command line tool for Docker containers. Pumba disturbs containers by crashing containerized applications, emulating network failures and stress-testing container resources (CPU, memory, file system, I/O, and others). With Pumba, I can inject faults by disturbing the network or the container. There are a bunch of fault injection case studies in this thesis based on Open5GS. For example, I can emulate the "AMF indicates AUSF error" by stopping the AUSF containers and then issuing a UE connection. In reality, this fault could happen when the AUSF is down or when it is already deployed and there is a communication error. Ideally, the following two commands should implement the fault injection

```
$ pumba stop ausf # stop AUSF
$ /UERANSIM/build/nr-ue -c /UERANSIM/config/custom-ue.yaml
   # UE attempt to connect
```

This is an instance of microservice-level fault injection. Apart from AUSF, we can do the same for other network functions like NSSF, PCF, SMF, etc. If we only care

[4] https://github.com/alexei-led/pumba/

about the location of faults (which service is erroneous), then the above fault injection process is enough.

In the simplest version, we consider only one kind of fault: microservice down. We define a fault as a tuple $(t, S)$, where $t$ is the time of the root fault and $S$ is the set of down Microservices (containers). Then, by simply running at some time $\tau_1$

```
$ pumba stop amf smf
```

we killed the service `amf` and `smf`, meaning that we injected the fault $(\tau_1, \{amf, smf\})$.

### 5.1.2 Experiment Steps

With the deployed test bed, we now describe our experiment steps. The experiment is done in 2 phases. In the first phase, we set up the TREAT system to align with the Open5GS test bed system, meaning preparing the template library and the event-based updating rules. In the second phase, we use the TREAT system to address the fault localisation task.

The second phase can be further illustrated in Figure 5.3. We run the test bed system and use a script to continuously simulate user operations, i.e., the UE accessing a website. Occasionally, The script also injects some faults $(\tau_1, S_1), ..., (\tau_m, S_m)$ into the test bed system (the previous fault would be corrected before a new fault is injected). During this process, the logs are collected and fed into the TREAT system. The TREAT system then uses the templates and the updating rules to extract knowledge from the logs and update the knowledge base. I take snapshots of the knowledge base at each time point $\tau_1, ..., \tau_m$ and use these snapshots to perform fault localisation by the method described in §5, i.e., to infer the faulty components via logic programming and ILP. The inferred faulty components are then compared with the gold standard faulty components, which are the ones we deliberately injected, to evaluate the effectiveness of the TREAT system.

In the first phase, I collected 102 log templates and wrote 8 event-based updating rules. The log templates are collected by manually inspecting the logs of the Open5GS test bed system. The event-based updating rules are written by manually inspecting the logs and the source code of the Open5GS test bed system.

In the second phase, I injected 520 faults in total. Taking the snapshots of the knowledge base at each time point $\tau_1, ..., \tau_m$ (in practice, I postponed 5 seconds to wait for the system reaction and generating the logs), I obtained 520 KB snapshots, and hence 520 $(KB_\tau, fault_\tau)$ cases.

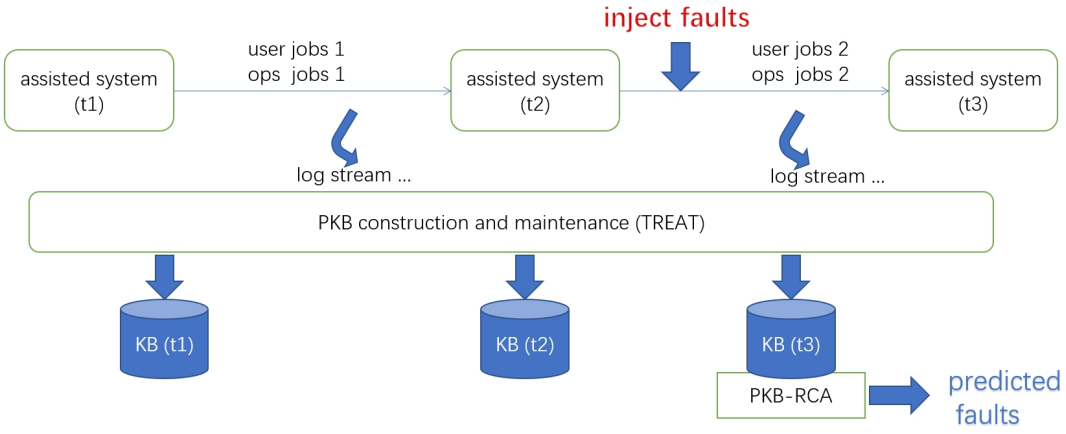



Figure 5.3: The procedural overview of the fault localisation process.

I split all the cases into 220 training cases and 300 testing cases. We used the 220 training cases to induce the inference rules, and then used the inference rules to infer the faulty component of the 300 testing cases. The precision of the inferred faulty components is 90% correct. This is the empirical evidence that the TREAT system can continually extract knowledge from the logs and update the knowledge base, and the evolving knowledge base can be used to facilitate the fault localisation task.

### 5.1.3 Discussions

The 10% failed cases were due to the mismatch of log templates and log messages, and hence not triggering the correct updating operations upon the knowledge base. Thus, the knowledge base cannot correctly reflect the system states, and therefore lead to incorrect fault localisation. For example, the two log events `<:nf_instance_id :> NRF NF registered` and `<:nf_instance_id:> (NRF-notify) registered` are similar but totally different. The former one says that an NRF instance was registered, while the latter one says that an NF was registered (but it may not be an NRF) and this log message was triggered by the NRF instance.

In fact, determining which template generates a log message is an ill-posed problem, and there is no solution guaranteed to give 100% correct results. As an extreme example, a log event `"Service <:serv_name:> created!"` tells which service was created, and the `<:serv_name:>` could be “WebUI” or “Database”, another logger monitors the event of the database service `"Service Database <:serv_evt:>!"` while the `<:serv_evt:>` can be “created” or “deleted”. Then, for a log message `"Service Database created!"`, it is impossible to determine which log template

generates this log. Arguably, this extreme example indicates bad software engineering practices that all projects should avoid, but it could exist in the real world.

## 5.2 Summary

This chapter not only demonstrated how the TREAT framework can be integrated into a localising the faulty components of a software system, but also provided empirical evidence supporting our hypothesis that our proposed TREAT framework can construct and continually update a knowledge base that reflects the state of a software system from its generated log stream. We formulate the fault localisation task as a Logic Programming problem and solve it using the existing toolkit (Problog & ProbFOIL), leveraging the extracted knowledge in the TREAT-powered knowledge base. We still see limitations of the fault localisation solution and the evaluation developed in this chapter. For instance, we still see some extent of human labour required, like writing regular expressions for preprocessing and updating rules for event-based updating, some of which demand high-level expertise. What's more, the test-bed system I deployed in this experiment is quite small, so we didn't see how well the TREAT framework can scale up in large industrial software systems. In spite of that, this experiment satisfactorily evaluated the effectiveness of the TREAT framework in extracting knowledge from logs and capturing changes within its assisted software system.

# Chapter 6

# LP-Measure: A Novel Method for KG Quality Assessment

In the last chapter, I evaluated the TREAT framework via fault localisation. As this TREAT-powered downstream application achieved good performance, I claimed that the knowledge in the TREAT-powered KB is accurate and useful and that the TREAT framework is effective. However, this evaluation has several limitations, and one of the most severe ones is that it relies on many external factors, particularly the Prolog-based fault localisation algorithm and the self-deployed testbed system (§5.1). It would be better to have an evaluation approach that relies as little as possible on external factors. To this end, I define the *LP-Measure* that can automatically assess the quality of a given knowledge graph without either gold standards or human labelling. To be more precise, I mainly assess robustness and redundancy rather than correctness and completeness. The core idea is to randomly remove some triples from the knowledge graph and then use a set of off-the-shelf link prediction models to predict the removed triples (hence called LP-Measure). The more removed triples can be recovered, the more the given knowledge graph is robust and redundant.

I designed experiments to empirically show that our LP-Measure is effective, at least in distinguishing high-quality KGs from low-quality ones. Though the LP-Measure is motivated to evaluate the TREAT-powered KB, unfortunately, it does not satisfy its initial purpose because of some limitations. In this chapter, I present this novel method for knowledge graph quality assessment and discuss its limitations and why I eventually did not use it.

---

This chapter is adapted from a conference paper from NLPIR 2023 [220]

## 6.1 Introduction

Knowledge Graph (KG) Construction is the prerequisite for all other KG research and applications. Researchers and engineers have proposed various approaches to build KGs for their use cases. However, how can we know whether our constructed KG is good or bad? Is it correct and complete? Is it consistent and robust? In this chapter, I propose a method called *LP-Measure* to assess the quality of a KG via link prediction tasks, without gold standard or other human labour. Though theoretically, the LP-Measure can only assess robustness and redundancy, instead of the more desirable correctness and completeness, empirical evidence shows that this measurement method can quantitatively distinguish the good KGs from the bad ones, even in terms of incorrectness and incompleteness. Compared with the most commonly used manual assessment, our LP-Measure is an automated evaluation, which saves time and human labour.

One widely used method is to estimate the correctness by sampling a small set of triples from the constructed KG and manually checking the precision of these sampled triples [142, 73]. For example, the SymbolicKD [205] project constructed a KG with 4.38 million triples by prompting a large language model, after which they manually checked the acceptance rate of a sample of 1000 triples. This method costs a lot of time and human labour and it can only estimate the correctness but not the completeness of a KG, and even this relies on the expertise of the human examiners.

Motivated by the above problem, in this chapter, I consider how to *automatically* assess the quality of a given KG with neither gold standards nor human labelling. I proposed *LP-Measure* to automatically assess the quality of a KG via the auxiliary link-prediction task. Simply speaking, the main idea is to *remove a small part of the KG, and then apply a standard suite of link prediction tools to check how many of the removed triples can be recovered.* I claim that the more triples that can be recovered, the more robust and redundant the original KG is, and the more likely it is that the original KG is of high correctness and completeness.

We can view this measurement from an intuitive perspective of fitting statistical models to datasets: using the same collection of datasets, the statistical methods that produce more accurate predictions are considered to be better methods (e.g., evaluating probabilistic language models on the GLUE benchmark [196]). Conversely, using the same collection of statistical methods, the datasets that produce more accurate models are considered to be better datasets (e.g., higher-quality corpus leads to more powerful

language models). Following the same idea, researchers usually evaluate different link prediction methods on some benchmark KGs, and conversely, we can evaluate the quality of KGs using the benchmark link prediction methods.

### 6.1.1 Quality of Knowledge Graphs

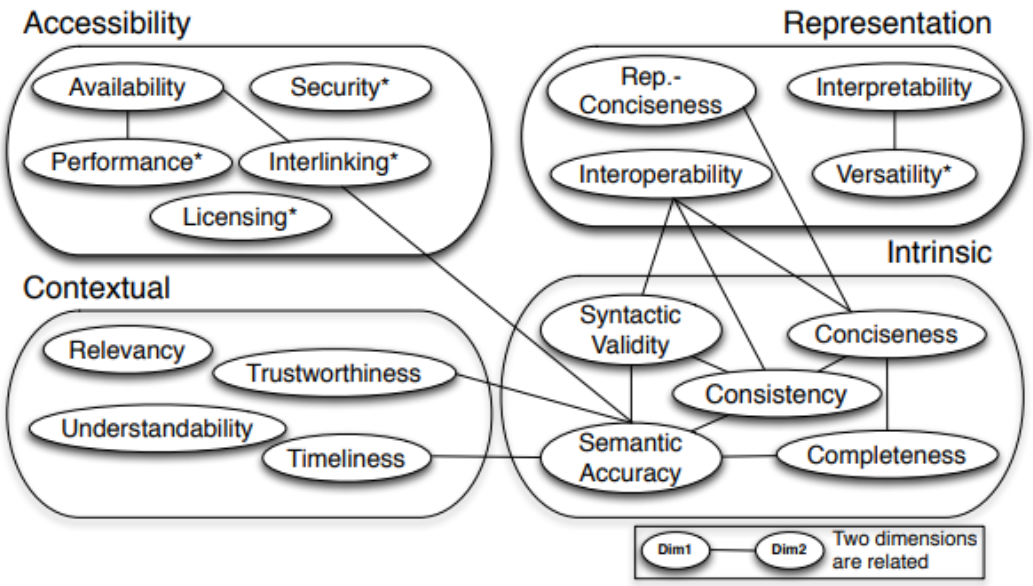



Figure 6.1: Linked Data quality dimensions and the relations between them. The dimensions marked with '*' are specific for Linked Data [213]

"Quality" is a broad and vague term, so needs to be defined in finer-grained dimensions. Zaveri et al. [213] proposed a comprehensive quality assessment framework designed for Linked Data, which can also be viewed as a knowledge graph but attempts to inter-connect all the open knowledge graphs. The framework refines "quality" to be 18 dimensions, grouped into 4 categories, as shown in Figure 6.1. We can see that some of these quality dimensions are specific to Linked Data, such as whether a KG is licenced, and some of the quality dimensions are for humans, such as interpretability. Of the quality dimensions, some of them are related, such as consistency, accuracy (correctness), and completeness.

Later, based on the framework of Zaveri et al. [213], Chen et. al. [33] proposed another framework for measuring the quality of "fit for purpose" KGs, or domain-specific KGs, as shown in Figure 6.2. They exclude the dimensions specific to Linked Data, and add some dimensions for domain-specific applications. For example, the KG

should be related to that application domain, and the KG should be of high authority in the aviation risk field.

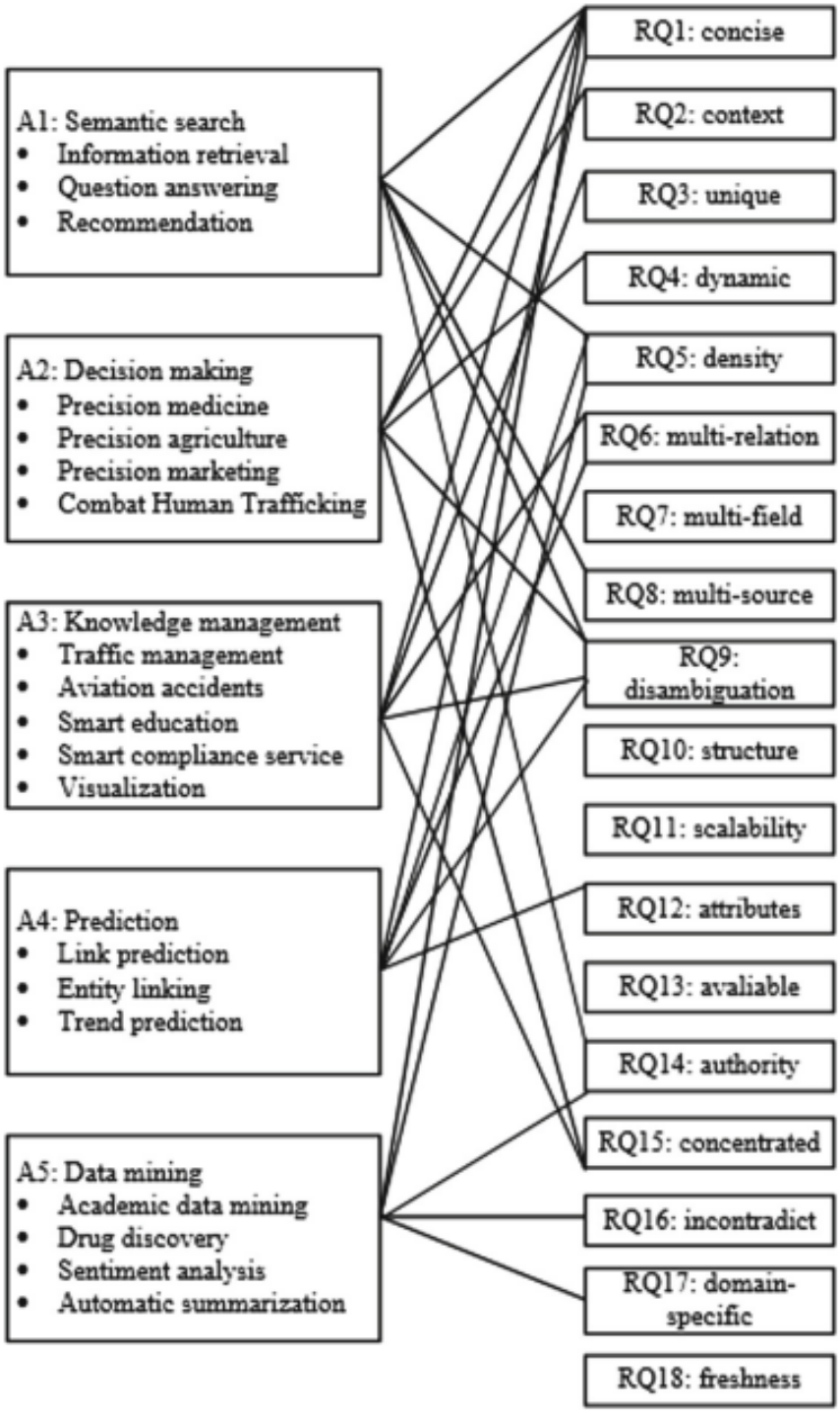



Figure 6.2: Knowledge Graph quality dimension curated from representative applications scenarios [33]

These metrics can be combined together to form a comprehensive quality assess-

ment framework. For example, the Luzzu Framework [54] is a Java implementation that allows users to conveniently define as many as possible of their own metrics and run them on the given KGs to obtain an overall quality score.

In the field of ontology engineering, a formal domain-independent framework called OntoClean [76] can be used to justify the decisions of ontology engineers. This is done by examining the ontology and analysing the meta-properties of ontology classes, such as *identity, unity, rigidity, dependence*, etc. Faria et al [68] introduced a crowd-based approach, called Crowd Quality (CQ) that aims at assessing the quality of data *linking*, which are mappings of entities across different ontology datasets.

Different from other datasets in the field of machine learning and general data analysis, Knowledge Graphs (and Semantic Web) usually evolve through time. Most well-known KGs, e.g., DBpedia and Wikidata are updated every day by users around the world. ConceptNet and YAGO also experienced several iterative releases in the past decade. Hence, how well a KG evolves, such as the degree of changes and lifespan of any entity type, is also an important dimension of quality, and is studied thoroughly [160, 18, 151].

### 6.1.2 Assessing the Intrinsic Quality

Though there are many quality dimensions, the intrinsic dimensions, including correctness, completeness, and consistency, are more important than others. According to the framework of Zaveri et al [213], correctness (semantic accuracy) means the degree to which the triples correctly reflect real-world facts, while completeness means the degree to which the entities and relations of a particular domain are represented in a KG (population completeness & Property completeness). Consistency means a KG is free of contradictions with respect to its representation and inference mechanism. Generally speaking, consistency is a weaker notion of correctness and completeness because a KG that's correct and complete should also be consistent, while a KG that's consistent may be total nonsense.

Most large KGs assess their quality by human evaluation, such as SymbolicKD [205] and Knowledge Vault [58]. As mentioned in §6.1, this is done by sampling a set of triples and manually checking the correctness of the sampled triples. The obvious drawback of human evaluation is that it costs time and money. There are chapters on how to save time and optimise the estimation [73, 142] by designing better sampling strategies, but the drawback still exists.

Some KGs are built to facilitate downstream applications. In this case, the performance of the downstream applications can be used as indirect indicators of the quality of the KGs. For instance, AttacKG [107] was built to aggregate cyber threat intelligence from textual reports. It was evaluated by the Named Entity Recognition (NER) task: with the help of AttacKG, the result of recognising cyber-security terminologies greatly improved. Yet, this merely moves the workload of evaluating a KG to evaluating another task. When no gold standards exist in those downstream tasks, we still need to employ human labour for those tasks. The AttacKG project manually labelled 16 Cyber Threat Intelligence reports to facilitate their evaluation.

A methodology called Competency Questions [164, 55] can be viewed as a general downstream task for ontologies and knowledge graphs. It borrows the idea from test-driven software development: before implementing any function modules, software engineers clarify the engineering requirement and write the test suites. Then, this test suite can be used as a guide during development, and as an assessment tool after development: the more tests are passed, the higher quality of the software. Competency Questions-driven ontology development requires the knowledge engineers first to write down a suite of questions that the expected ontology can answer. For example, an ontology of computer science could be expected to answer questions like "Which program implements algorithm X?" Once we set up a suite of competency questions, we can use these questions to guide ontology development and quality assessment: the more questions that can be answered, the higher the quality of the ontology. Besides this, there are also methods that directly mimic test-driven software engineering to assess and assure the quality of ontology engineering [96]. Yet these methods require the KGs to be equipped with schemas of the domain of interests, and like the competency questions, demand human labour to write the test suites.

There are studies on assessing the quality of triples. For instance, KGTtm [91] proposed a confidence measurement called Triple Trustworthiness to evaluate how much a triple in a KG can be trusted. The Trustworthiness is produced by a neural network trained to capture the triple semantics and global information of the KG. Besides the neural net approach, Inductive Summarisation [16] tried to learn a set of rules that can best summarise the KG, and leverage the induced rules to detect the abnormal triples, as well as infer missing triples. Both these methods and other similar approaches can be considered as variants and further applications of the knowledge graph link prediction task, of which the core idea is to train a model that captures the information of the KG, and use the model to score triples (either new triples or existing ones).

## 6.2 Preliminary: Knowledge Graph Link Prediction

In this section, I briefly explain some important concepts used in LP-Measure. Recall that **knowledge graph** $\mathcal{G}$ is a tuple $(\mathcal{E}, \mathcal{R}, \mathcal{T})$ (§3.2), **Ideal Knowledge Graphs** [16] are the KG $\mathcal{G}^* = (\mathcal{E}^*, \mathcal{R}^*, \mathcal{T}^*)$ that contains all the correct triples of the domain of interest and no incorrect ones. The notion of correctness (precision) and completeness (recall) is defined by comparing the constructed KG $\mathcal{G}$ with this ideal KG $\mathcal{G}^*$:

$$correctness = \frac{|\mathcal{T} \cap \mathcal{T}^*|}{|\mathcal{T}|}$$

$$completeness = \frac{|\mathcal{T} \cap \mathcal{T}^*|}{|\mathcal{T}^*|}$$

Nevertheless, this ideal KG is merely a conceptual aid, and usually does not exist. After all, if we have such an ideal KG at hand, we don't need to bother constructing a KG.

**Link Prediction** [167, 200], or knowledge completion, is a typical task in the field of Knowledge Graph that aims to predict missing links between entities (triples) of a KG based on its existing knowledge. Our proposed measurement, LP-Measure, relies on the knowledge graph link prediction as an auxiliary task.

Commonly used link prediction methods include TransE [22] and ComplEx [187]. Later methods based on deep neural networks include ConvKB [138] and GNN [210]. More advanced methods, based on pretrained language models, include MEM-KGC [36], SimKGC [199], and so forth.

In the link prediction research community, a new prediction model $\mathcal{M}$ is usually evaluated in such a way: take an already known high-quality KG as a benchmark, remove some of its triples, and see how many triples can be recovered (predicted) by the new prediction model $\mathcal{M}$. The more triples that are recovered, the better the performance of the prediction model. For example, when TransE [22] was proposed, it was evaluated on a KG called FB15k, a subset of Freebase [21], and WN11, a subset of WordNet [124].

More concretely, researchers split the triples of a KG into a training set and a test set. The triples in the training set are used to train the prediction model, while the triples in the test set, together with some generated synthetic negative triples are used to evaluate the performance of the trained model. Since almost all KGs only encode positive triples [12], so one may wonder how we can obtain the negative triples to facilitate training and testing. A widely-used method in KG link prediction is **negative sampling** under the **Local Closed World Assumption** [139].

Unlike the Closed World Assumption which asserts all triples not encoded in the KG to be false, and the Open World Assumption which asserts all triples not encoded in the KG to be unknown, the Local Closed World Assumption states that for a given subject-predicate pair $(s, p)$ observed in a KG, we assume that all non-existing $\langle s, p, ?\rangle$ to be false, but for all other $(s, p)$ not observed in the KG, we assume that all non-existing $\langle s, p, ?\rangle$ to be unknown. Under this Local Closed World Assumption, we can generate synthetic negative triples by taking a positive triple from the KG and then randomly replacing its subject and object with other entities to create non-existing triples. For example, taking a triple `<James_Cameron, director_of, Titanic>`, we may generate a negative triple `<James_Cameron, director_of, God_Father>`. Of course, this Local Closed World Assumption is valid for functional predicates like `born_in`, but not valid for multi-valued predicates like `director_of`, yet in practice, it can reliably generate negative triples.

**To evaluate the performance of a Link Prediction model**, I trained and then tested it: for a positive (correct) triple and several negative (incorrect) triples, the prediction model will output scores of these triples, and rank the triples based on their scores. If more positive triples are ranked ahead of more incorrect ones, then this prediction model is considered to be more powerful in learning the features of a KG and predicting new triples. The commonly used metrics are Mean Reciprocal Rank (MRR) and Hit at K (Hit@k), both of which are within the unit interval $[0, 1]$, the larger the better.

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i}$$

$$Hit@k = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \mathbb{I}[rank_i < k]$$

where $Q$ is the test set, and $\mathbb{I}[\cdot]$ is the indicator function of an assertion, returning 1 if the assertion is true while 0 if the assertion is false. The removed triples for testing are called *silver standards*[148, 206] instead of gold standards because the benchmark KG, though high-quality, is not perfect.

The evaluation of link prediction algorithms is a kind of controlled experiment: we use the same benchmark KGs (FB15k or WN18), and check the performance of different link prediction models. Conversely, if we use the same benchmark link prediction models, we can also check the quality of different KGs. This is the main idea of our proposed measurement method, LP-Measure.

## 6.3 LP-Measure

LP-Measure is inspired by the task of link prediction and the idea of a controlled experiment. One key observation/assumption of link prediction is that:

> It is possible to predict the missing triples of a KG based on its existing structure. The higher quality a KG is, the more reliably we can predict the missing triples [169, 200].

This observation is quite straightforward and intuitive. After all, the task of link prediction is to capture the information and underlying patterns of the existing triples, and leverage them to predict the missing triples. If the existing triples contain a lot of noise or miss a lot of information, then the prediction model cannot learn well and will make inaccurate predictions. Therefore, to be able to conduct the task of link prediction, a precondition is that this KG should be already relatively correct and complete. Taking a further step, I **hypothesise** that

> *For a high-quality KG, we can remove a small part of the KG, and reliably recover (predict) the removed part. The higher the quality, the more removed triples we can recover.*

In other words, a high-quality KG is also highly recoverable. Based on the hypothesis above, I propose to **use a standard set of link prediction models** (served as a benchmark) to assess the quality of different KGs.

The idea is simple: first of all, we determine a controlled set of link prediction algorithms as the benchmark, *e.g.*, TransE or ComplEx. Given a KG $\mathcal{G}$ whose quality is unknown, we randomly remove a small part of the KG. We denote the removed triples to be $g$ and the rest to be $G$ (similar to the training/test set split). Then we train the benchmark link prediction model $\mathcal{M}$ on $G$.

Finally, we apply the trained model to recover the removed triples in $g$. If most of the removed triples can be recovered, then we can claim that the given KG $\mathcal{G}$ is of high quality. Algorithm 1 is the pseudo-code of our proposed method. Figure 6.3 illustrates the main process. The returned link prediction result, e.g., *mrr* score, indicates the quality of the KG $\mathcal{G}$, the higher the score *mrr*, the higher the quality of $\mathcal{G}$

In the demonstrative pseudocode, the prediction mode, $\mathcal{M}$ is a combination of multiple chosen models, such as TransE [22] and ComplEx [187]. The output of the function `evaluate_prediction(m, g)` is also the average of the *MRR*s or *Hit@k*s by every single prediction model.

**Algorithm 1:** Measuring quality of KG by Link Prediction

**Data:** The given knowledge graph $\mathcal{G}$, the given link prediction model $\mathcal{M}$

**Result:** The link prediction result $mrr$ or $hit@k$

$G, g \leftarrow split(\mathcal{G})$ ;

$m \leftarrow train(\mathcal{M}, G)$ ;

$mrr, hit@k \leftarrow evaluate\_prediction(m, g)$

One point to note is that LP-Measure is not a single metric, but an assessment method. The metrics to tell the good or bad are those of the link prediction tasks, such as MRR and Hit@k mentioned in §6.2. The higher these metrics of a KG, the better correctness and completeness, or to be more exact, the higher robuestness and redundancy.

## 6.4 Evaluation

In this section, I provide empirical evidence to support the effectiveness of our proposed measurement method, LP-Measure. It is a direct application of the hypothesis

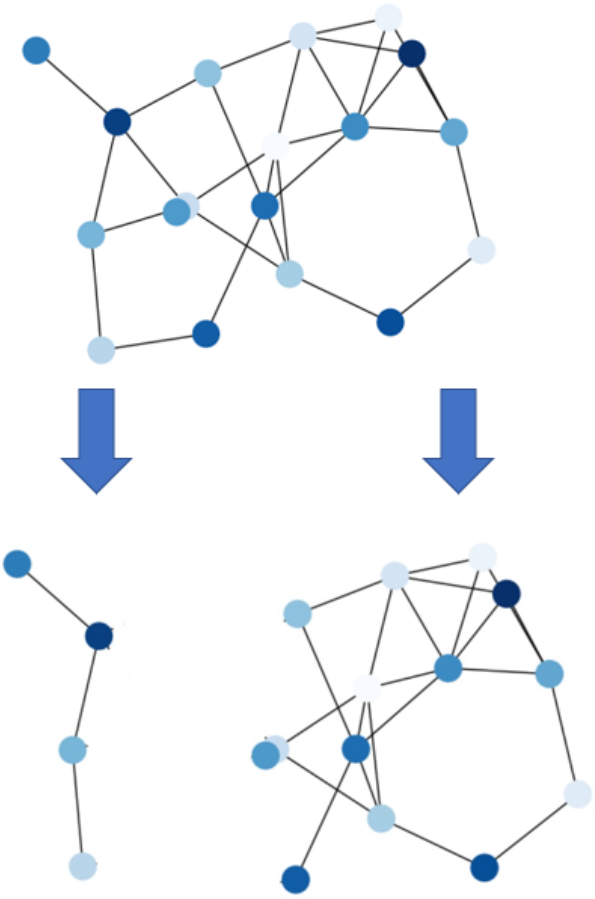
> *For a high-quality KG, we can remove a small part of the KG, and reliably recover (predict) the removed part. The higher the quality, the more removed triples we can recover.*

To evaluate the hypothesis, I conducted an empirical experiment:

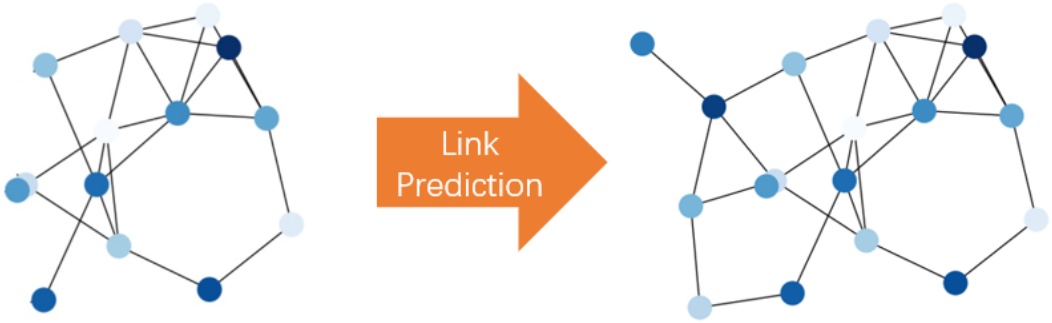
1. Identify a well-known good KG $\mathcal{G}$.
2. Create a worse KG $\mathcal{G}'$ i.e., less correct and less complete ones, based on the good KG $\mathcal{G}$.
3. Apply LP-Measure on both $\mathcal{G}$ and $\mathcal{G}'$, obtaining corresponding link prediction results. In this experiment, I showed the results of MRR, Hit@1, and Hit@3.

If we see the link prediction results of $\mathcal{G}$ significantly greater than that of $\mathcal{G}'$, then here is the evidence supporting our hypothesis, and justifying the efficacy of our LP-Measure.

Within this framework, I chose 4 good KGs: FB15k, FB15k-237, WN18, WN18RR [22, 186, 56]. These 4 KG datasets are widely used in the link prediction research community, and the popularity of these benchmark KGs indicates their well-recognised high quality (correctness, completeness, and consistency). I chose 2 models, TransE [22]



(a) Remove a small proportion of triples from the KG




(b) Train a controlled set of prediction models on the rest of the KG, and try to recover the removed triples

Figure 6.3: The main idea of LP-Measure

and ComplEx [187], implemented by Ampligraph[1], which are also well-known in the

[1] https://github.com/Accenture/AmpliGraph/

link prediction community. There are many more powerful and later models, but TransE and ComplEx are simple and easy to run. The deep neural network models or even the transformer models are too slow and demand too much computing power. After all, the important point is to use fixed link prediction models, not to use state-of-the-art models. When removing triples, I randomly take off 10% of the triples, i.e., I did a 90%/10% split, and LP-Measure will check how many triples of the removed 10% can be recovered from the rest 90%.

I created 2 types of "worse" KGs: incorrect KGs and incomplete KGs.

### 6.4.1 Incorrect KG

I create an incorrect KG by replacing part of the triples with negative ones generated by negative sampling techniques as mentioned in §6.2. Here I chose 25% and 50% triples to corrupt because I consider it to be a fair enough proportion to create a noisy KG. I apply the LP-Measure on both good and worse KGs. Figure 6.4 shows the experimental results.

Recall that both MRR and Hit@k are within the unit interval $[0, 1]$, the larger the better. We can see that the link prediction results of the original (good) KGs significantly outperform the injected (worse) KG, which is empirical evidence supporting that our measurement method is effective in distinguishing the correct KGs from the incorrect ones. Given that the more incorrect triples injected, the lower the MRR and Hit@k, we can say that LP-Measure is a quantitative assessment method, i.e., the higher LP-Measure, the better KG quality.

### 6.4.2 Incomplete KG

I create an incomplete KG by randomly taking out some triples from the original KG. Unlike the previous experiment on incorrectness, I didn't inject any corrupted triples. Again I took out 25% and 50% of the triples, to create incomplete versions compared with the original one. I apply the LP-Measure on both the good and worse KGs. Figure 6.5 shows the experimental results[2].

Similar to the previous experiment on inconsistency, We can see that the link prediction results of the original (good) KGs significantly outperform the incomplete (worse) KG, except in the FB15k dataset, where the KG taken out 50% of triples

[2]I conducted the experiments 3 times and took the average of every datum, so did the later experiments on incompleteness.

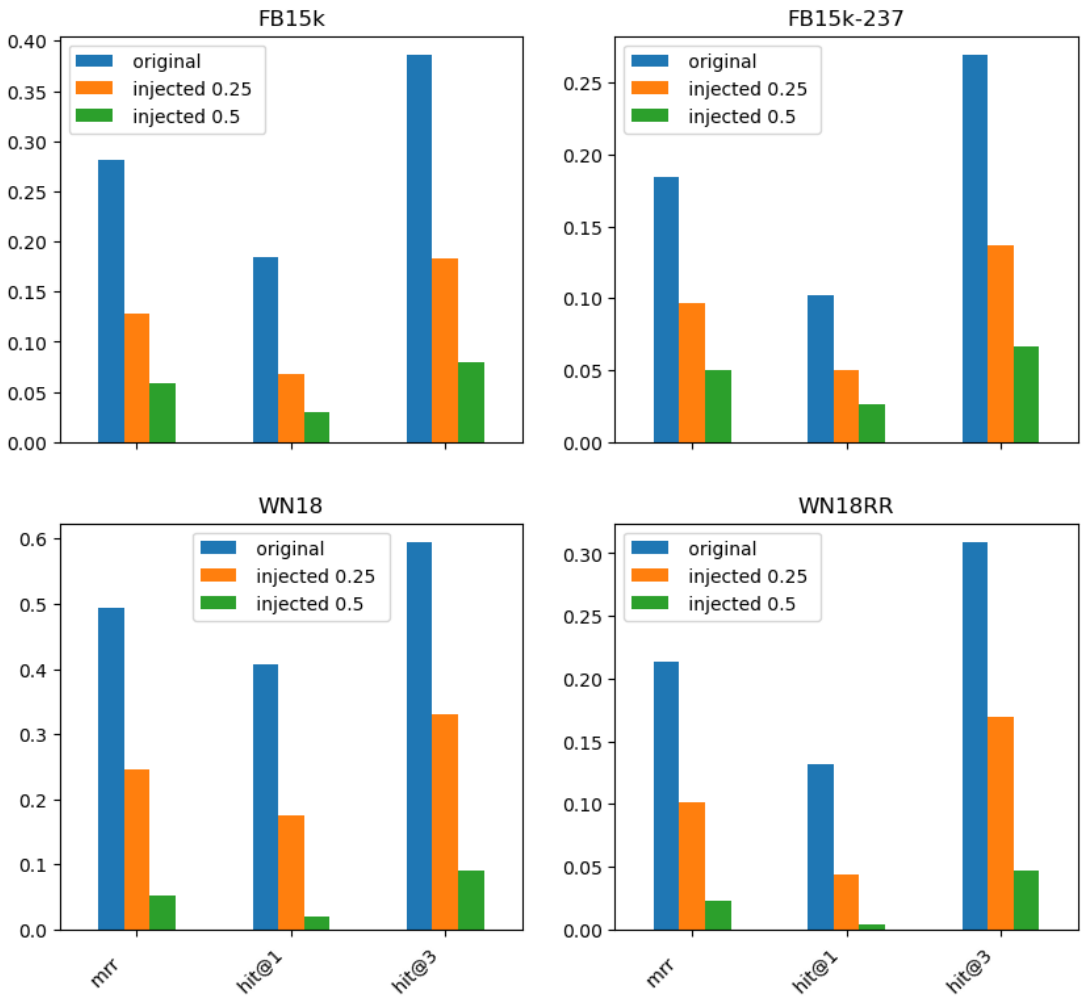



Figure 6.4: LP-Measure results of the original datasets and the worse versions with injected incorrect triples. Metrics are computed by averaging those of TransE and ComplEx.

got even better results than the one taken out only 25%. The experimental results of incompleteness do not look as convincing as those of the experiment on incorrectness, this also indicates that the LP-Measure may not be good at distinguishing the complete one and the incomplete one, which is also evidence supporting our thought that the LP-Measure concerns the self-consistency and redundancy of a KG.

Interestingly, the datasets FB15k vs FB15k-237, and WN18 vs WN18RR also constitute 2 pairs of incompleteness comparison. I show the MRR results of them in Table 6.1. FB15k-237 and WN18RR are subsets of FB15k and WN18 respectively by removing the inverse relations, e.g., `(s, hyponym, o)` and `(o, hypernym, s)` to avoid test leakage [186, 56]. Thus, a link prediction algorithm always gets lower

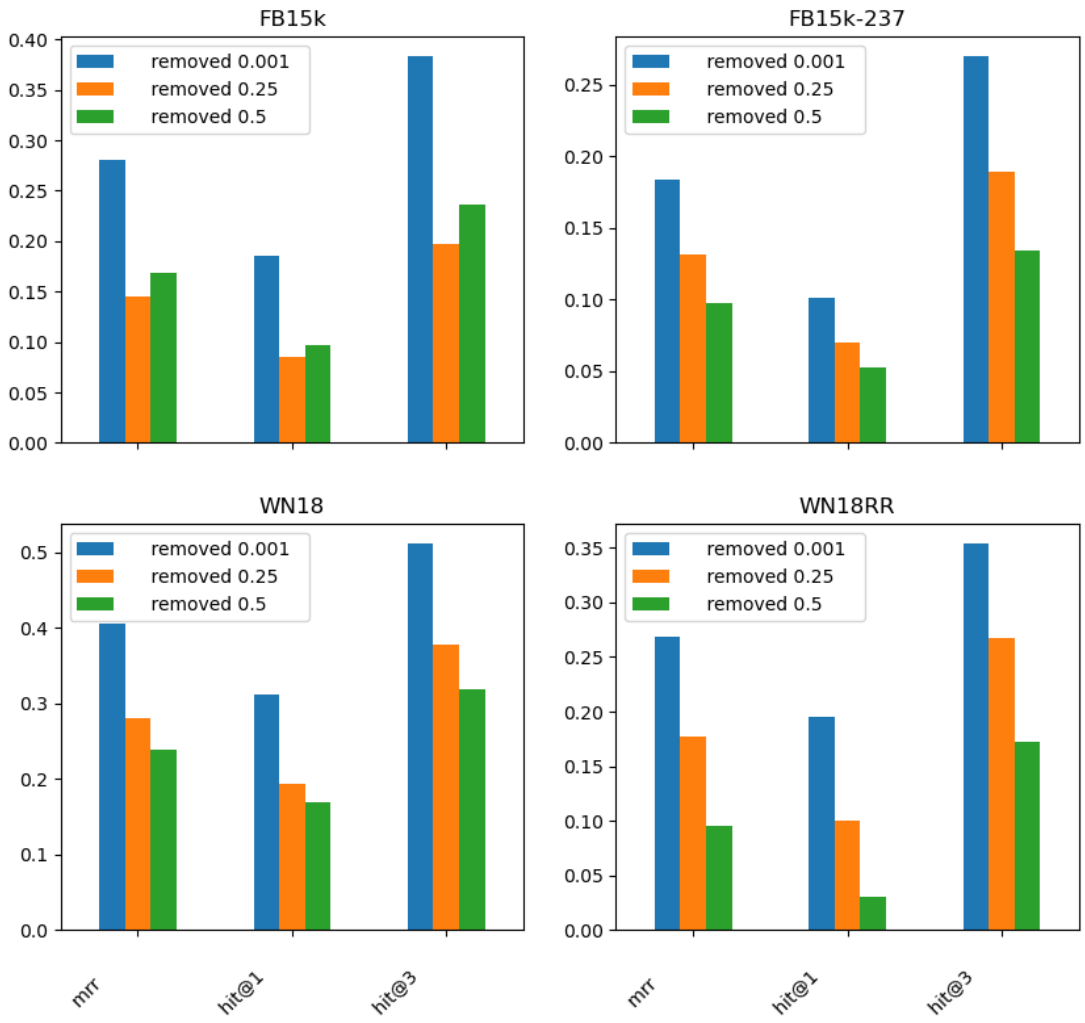



Figure 6.5: LP-Measure results of the original datasets and the worse versions with missing triples. Metrics are computed by averaging those of TransE and ComplEx.

results in FB15k-237 and WN18RR than FB15k and WN18, which is commonsense in the research community. This commonsense is also evidence supporting the claim that the higher LP-Measure a KG would be, the more complete (redundant) it is.

| | FB15k | FB15K-237 | WN18 | WN18RR |
|---|---|---|---|---|
| TransE | 0.145 | 0.127 | 0.180 | 0.143 |
| ComplEx | 0.417 | 0.241 | 0.813 | 0.395 |
| Average (LP-Measure) | 0.281 | 0.184 | 0.495 | 0.269 |

Table 6.1: MRRs on all the original datasets

Another interesting result is that the LP-Meausre of WN18 (WN18RR) is significantly better than that of FB15k (FB15k-237). This also aligns with our intuition, since the domain of FB15k is open-world knowledge, so we expect that the KG would be fairly incomplete and involve many mistakes. In contrast, the domain of WN18 is merely lexical knowledge, which is a much smaller and restricted domain. Therefore, we would expect that such a constructed KG would be more correct and complete than the one of open-world knowledge.

## 6.5 Discussion & Limitations

Though the LP-Measure is simple, effective, and doesn't require gold standards and extra human labelling, there are limitations.

The most significant limitation is that LP-Measure only works well on large KGs. Because we exploit link prediction models to do the measurement, and most link prediction models require large data to learn the statistical patterns and perform link prediction. Thus, our measurement may not well reflect the quality of small KGs. There could be KGs with less than 100 triples describing a very small domain, e.g., the relationship within a family. In this case, most data-driven link prediction models cannot work. This is why I mentioned in the beginning of this chapter that LP-Measure does not satisfy its initial purpose of evaluating the quality of TREAT-powered KB/KG. Currently, the TREAT KB size is relatively small, and the LP-Measure cannot produce meaningful metric scores for the small KBs.

Secondly, by removing part of a KG and measuring how much can be recovered, LP-Measure cannot assess the correctness and completeness of a KG. Rather, it measures the aspect of robustness or redundancy of a KG. This is due to the nature of link prediction methods [167]: they essentially learn the statistical patterns of the graph topology, including the similarity between nodes based on shared attributes, and the patterns of relationships between nodes in the graph, and then use these learned patterns to predict the new links (triples) that are not shown in the KG but are statistically consistent with the existing graph structure. In an extreme case, there might be a KG whose triples are entirely incorrect with respect to the real world but internally consistent. Then LP-Measure could output a high score for this nonsense KG, but its correctness and completeness should be indeed low. For instance, a KG created in 2012 may be a highly correct KG, but standing in 2022, most of the triples can be outdated and thus become incorrect, e.g, `<USA, president, Barack_Obama>` is correct

in 2012 but incorrect in 2022. Suppose there are 2 KGs $\mathcal{G}_1$ and $\mathcal{G}_2$ on the same specific domain of interest. $\mathcal{G}_1$ is created at $t_1$ while $\mathcal{G}_2$ is an evolved version created later at $t_2$. In this case, LP-Measure could not distinguish which one is better, though standing at $t_2$, most of the triples in $\mathcal{G}_1$ may become incorrect.

Last but not least, LP-Measure is also an indirect measurement. What we can obtain are the standard link prediction metrics in the research community, like MRR or Hit@k on that KG, not as straightforward as direct measurements like precision and recall, or the more general ROC-AUC. What's worse, there's no guarantee that it could work in all KGs. LP-measure is built on the basis of Link prediction tasks, and not all KGs are applicable to this task. However, there are few studies on the precondition of KGs applicable to link prediction.

However, in case we don't have gold standards to compute precision and recall, our measurement can serve as a quick and easy-to-use probe. We can always use the LP-Measure first, after which we can decide whether or not to spend some time on human labelling and compute the precision for correctness (and hopefully the recall for completeness).

## 6.6 Summary

In this chapter, I proposed a method called the LP-Measure to assess the quality of a KG by means of link prediction tasks. The biggest advantage of our LP-Measure is that it can be run in a fully automated manner, without needing extra human labour to provide gold standards. Though the applicability is limited, and LP-Measure essentially assesses the consistency of a KG instead of the more desirable correctness and completeness, empirical experiments show that our proposed measurement is effective for large KGs, at least where it can distinguish the good KGs from the bad ones.

# Chapter 7

# A Closer Look at Probability Calibration of KG Embedding

In the TREAT framework, knowledge extraction is done by template parsing. However, as templates can be mismatched, there could be some false positives, and that's the main reason for the 10% failed predictions in the fault localisation evaluation. Thus, I wished to assign probabilities to the extracted triples, so that we could realise the potential risks in downstream reasoning. During research, I found that probability calibration is a widely adopted approach to convert scores without probabilistic semantics into mathematical probabilities as well as refining probability estimations. That led my research interests to probability calibration, and its application in knowledge graph embeddings.

When the estimated probabilities do not match the relative frequencies, we say that these estimated probabilities are *uncalibrated* [191], which may cause incorrect decision-making, and is particularly undesired in high-stakes tasks [217]. Knowledge Graph embedding models are reported to produce uncalibrated probabilities [183], e.g., for all the triples predicted with probability 0.9, the percentage of them being truly correct triples is not 90%. In this chapter, I take a closer look at this problem. First, I confirm the claim that typical KG Embedding models are uncalibrated [183]. Then, I show how off-the-shelf calibration techniques can be used to mitigate this issue, among which binning-based calibration produces more calibrated probabilities. I also investigated the possible reasons for the uncalibrated probabilities and found that the *expit transform* [183], the way typically used to convert embedding scores into probabilities, is ineffective in most cases.

This chapter is adapted from a conference paper from IJCKG 2022 [221]

## 7.1 Introduction

One approach to assigning probabilities to triples of a KG is to train embedding models [26] (see §6.2 for more details), e.g., TransE [22] or ComplEx [187], for KGs, and then use the scoring function of the trained embedding model to score the new triples:

$$score = f_{embed}(\langle \vec{s}, \vec{p}, \vec{o} \rangle)$$

where $f_{embed}$ is the scoring function of the embedding model, and $s,\ p,\ o$ represent subject, predicate and object, respectively. There have been some works suggested that these scores can be converted into probability values via *expit transform* [139, 183], i.e., passing these scores through the Sigmoid function as follows.

$$prob = \sigma(score) = \frac{1}{(1 + exp(-score))}$$

where $\sigma$ represents the Sigmoid function. However, this treatment is problematic as the expit transform does not necessarily convert scores into values of probabilistic sense. Even it if does, later works [183] showed that the probability values obtained in this way are uncalibrated; e.g., for all the triples with probability 0.9, the percentage of them being correct triples w.r.t. the real world is not 90%. Thus, these expit-transformed probability values need to be calibrated

$$prob^* = f_{calib}(prob)$$

where $f_{calib}$ is a calibration model, and $prob^*$ are calibrated probability values that neither over-estimate nor under-estimate the truth of triples. Then, a subsequent question is how to pick an appropriate calibration model from the off-the-shelf toolkit.

In this chapter, I looked deeper at the research of probability calibration for knowledge graph embedding, with the following findings:

1. I stressed [1] that not all expit-transformed scores are appropriate to be interpreted as probabilities. Also, I argue that probability calibration can serve as an accurate technique to convert embedding model scores into probability values.

2. Though expit-transformed scores of some embedding models can be interpreted as probabilities, I found that even these probabilities are uncalibrated, and thus calibration is needed.

[1] We are not the first to show this phenomenon, but unfortunately still many people mess up.

3. I provided empirical evidence for a useful rule of thumb [141] for how to choose calibration techniques: for a large set of held-out data (say, over 10 thousand triples), binning-based calibration techniques perform better, such as Isotonic Regression and Histogram Binning. Otherwise, scaling-based techniques, such as Platt Scaling, are more suitable.

## 7.2 Preliminaries: Probability Calibration

In this section, I briefly explain some important notions of calibration used in this chapter.

**Score vs Probabilitiy** Numeric scores are common tools to express uncertainty and rank items, usually without formal definitions. Scores can range between arbitrary real numbers, for example, an un-thresholded Linear Support Vector Machine $f(x) = w^\top x + b$ output a score within $[-\infty, +\infty]$, and the higher the absolute value of the score, the more likely the corresponding instance is classified. Probabilities are scores with formal mathematical definitions to express uncertainty. Clear distinctions between scores and probabilities include that probabilities lie in $[0, 1]$ while scores can be unbounded; and probabilities follow a set of Probability Axioms [69] while scores don't.

Although there is a firm mathematical foundation for probability, and it is agreed that probability is a measure of uncertainty, the exact interpretation of probability values is still not controversial [85]. Some argue that probabilities measure the objective uncertainty behind this real world, while some argue that there is no such thing as objective uncertainty probabilities are subjective estimates of the degree of belief. And there is a wide spectrum between these 2 extreme interpretations. In this chapter, the term "probabilities" is used to refer to the *estimates of the degree of uncertainty*, whether the uncertainty is believed to be objective or subjective. As they are estimates, there could be *calibrated probabilities* and *uncalibrated probabilities*.

**Epistemological Origin of Calibration** In a word, calibration means that the forecast probabilities should match the relative frequencies: $fr(X|pr(X) = \beta) = \beta$, where $fr(X)$ represents the relative frequency of $X$ and $pr(X)$ represents the predicted probability of $X$. It is also a technique to adjust the uncalibrated probabilities, or directly transform classifier scores with no probability meanings into probabilities that satisfy

probability axioms and have probability semantics. The notion gained attention and became well-studied in the 1960s [133, 191, 49] from a philosophical inquiry of an epistemic virtue for partial beliefs (aka, subjective probabilities or credence), analogous to the virtue of truth for full beliefs.

Suppose that I would like to go to a picnic tomorrow. But here in Edinburgh it often rains. If it won't rain tomorrow then I can begin to buy some food and make phone calls to friends. If it will rain tomorrow then I shall not bother preparing. The key question is whether it will rain tomorrow. I can make a **full belief** prediction that "Tomorrow it will rain!" I can verify this prediction by waiting for tomorrow's actual weather. By this verification, I can say that my full belief prediction has an epistemic virtue of truth, meaning that I can assert it as a good prediction if it matches reality, or a bad prediction if it fails to match reality. What if I make a **partial belief** prediction that "It seems 70% probable to rain tomorrow"? We cannot verify this prediction through the actual result of tomorrow's weather, even if it would rain tomorrow indeed. Although we cannot verify the truth of my partial belief by one prediction, we can evaluate the *quality* of my partial beliefs in a long series of predictions.

Formally, consider binary classification tasks. Given a set of samples $(X, y) \in \mathcal{D}$, if $\forall \beta \in [0,1]$, we have $fr\big(X|pr(X) = \beta\big) = \beta$, where $fr(X)$ represents the frequency of $X$ being a positive sample, and $pr(X)$ represents the predicted probability of $X$ being a positive sample, we say the predicted probabilities $pr(X)$ are calibrated. Otherwise, we say they are uncalibrated, and thus calibration is needed.

**Technical Mechanism of Calibration** In Machine Learning, probability calibration is also a family of techniques to obtain well-calibrated probabilities. Some statistical classifiers produce scores that are NOT probabilities, such as the Support Vector Machine [40]. Calibration can be used to convert these scores into probabilities [150]. Sometimes even probabilistic classifiers do not produce calibrated distribution, due to over-fitting or other reasons [159]. In this case, calibration methods can also be applied.

In principle, we can see the original classifiers $f(x|w)$ as feature extractors [159]. They take raw input data and produce scores or (uncalibrated) probabilities as features of the input data. Then a calibration model, essential a simple probabilistic classifier $p[f(x|w)|\theta]$ like logistic regression, can be applied to learn from these extracted features, and output calibrated probabilities. We call this method post-calibration, meaning post-processing the output of the original model to produce well-calibrated proba-

bilities.

Why does this post-calibration idea work? Why do those original models (even probabilistic ones) fail to produce well-calibrated probabilities? A recent paper [78] gives an explanation. If we assume there exists a "true distribution", then this true distribution is well-calibrated by nature. Once our probabilistic model successfully learns that true distribution, then our learned distribution will also be well-calibrated. However, that's not the case. Our model may get over-fitted to the training set. The more complicated our model, like deep neural networks, the more possible it is to get over-fitting. Another reason could be that the feature space and/or the landscape of the loss function are too complex that our learning model may be trapped in local optima, during optimisation. Because of these, our model (even probabilistic) usually learns an approximate distribution possibly far away from the true distribution, and thus produces uncalibrated probabilities. Post-calibration techniques use simple probabilistic models, such as logistic regression, or one-layer probabilistic neural networks to learn from a simple feature space, which is the original output of the model, such as scores of an SVM model [40], or uncalibrated probabilities of a deep probabilistic neural network.

When using post-calibration methods, we should have at least 3 datasets for training $(X_{train}, y_{train})$, calibration $(f(X_{cal}|w), y_{cal})$, and testing $(X_{test}, y_{test})$. We train the original model on the training set, train a calibrator on the calibration set, and evaluate the performance, including how accurate the prediction is and how well-calibrated the probabilities on the test set.

How do we check if a forecast distribution is well-calibrated? In the last section, we decomposed the Brier Score into the calibration term and refinement term. In theory, the calibration term

$$C = \frac{1}{K}\sum_{j=1}^{K} n_j(p_j - r_j)^2$$

can serve as a metric. Yet there are practical difficulties. The calibration term requires grouping events by their exact probability $p_j$, and computing the relative frequency of these $n_j$ events. This requires the $n_j$ to be big enough for every unique probability value $p_j$ to produce a frequency value of statistical significance. But this is rare in practice. Mostly we have events whose probability value $p_j$ occurs only once, and the corresponding $n_j$ equals 1, where it is impossible to calculate a meaningful frequency value. As a compromise, we sometimes use directly the Brier Score to

evaluate probability calibration. Or we make some relaxation to the grouping criteria, not to group events by their **exact** probabilities, but **similar** probabilities, so that each group can have enough events to calculate a meaningful frequency. This is how the widely used ECE(Expected Calibration Error) [135] works.

$$ECE = \frac{1}{n}\sum_{i}^{n}|pr_i - fr_i|$$

where $pr_i$ represents the average probability of the $i$th group and $fr_i$ represents the relative frequency of the $i$th group. Here is a naive example:

$$P = [(0.11, 0.12, 0.13, 0.14, 0.15), (0.81, 0.82, 0.83, 0.84, 0.85)]$$

$$Y = [(0, 0, 0, 0, 1), (1, 1, 1, 1, 0)]$$

$$ECE = \frac{1}{2}(|0.13 - 0.2| + |0.83 - 0.8|) = 0.10$$

In this example, there are only events whose predicted probability value occurs only once. We group the events with close probabilities into 2 clusters, and calculate the average probabilities and relative frequencies of these 2 clusters, and compare the cumulative distances, which is 0.10, a relatively small and satisfying result.

## 7.3 Expit-Transformed Scores as Probabilities?

Recall that some works suggested that the scores $f_{embed}(\langle\vec{s},\vec{p},\vec{o}\rangle)$ can be converted into probabilities via *expit transform* [139, 183]. Depending on the scoring function of KG embedding models, expit-transformed scores sometimes can be interpreted as probabilities but sometimes not. I was not the first to point out this issue. It has even been noted in some libraries documentation[2]. For instance, TransE adopts a distance-based scoring function:

$$f_{TransE}(\langle\vec{s},\vec{p},\vec{o}\rangle) = -||\vec{s} + \vec{p} - \vec{o}||_2$$

Hence $f_{TransE}(\langle\vec{s},\vec{p},\vec{o}\rangle) \in [-\infty, 0]$, and thus $\sigma\big(f_{TransE}(s,p,o)\big) \in [0, 0.5]$. That is to say, the expit-transformed scores of TransE are always lower than 0.5, which can hardly be recognised as probabilities, regardless of the truth of a triple. Any embedding

[2] `https://pykeen.readthedocs.io/en/stable/reference/models.html`

models adopting distance-based scoring functions as TransE, such as TransD [90], TransR [203], TransH [110], RotatE [182], PairRE [32] and BoxE [3] do suffer from this problem.

Some may suggest it is not a problem because we can always map the scale to the unit interval, for example, doubling the scale of expit-transformed scores of TransE so that now the range turns from $[0, 0.5]$ to $[0, 1]$, and obey the probability axioms [194]. In our later experiments (§7.4.1), the expit-transformed values of TransE did achieve relatively high accuracy in the triple classification task. Nevertheless, it is not the case when it comes to other embedding models. As shown in Figure 7.2, the doubled expit-transformed values of TransR and RotatE[3] are still lower than 0.5.

Then, scores of which models can apply expit-transform? One necessary condition (but not sufficient) is that a) the range of the scoring function should be the whole real interval: $f_{embed}(\langle \vec{s}, \vec{p}, \vec{o} \rangle) \in [-\infty, +\infty]$; b) the model optimises the Negative Log Loss or Binary Cross Entropy during the training course. Optimising a Negative Log Loss or Binary Cross Entropy is equivalent to performing Maximum Likelihood Estimation, so the whole training procedure is equivalent to estimating the probability distribution of the fitted dataset. Because of this, we can, mathematically, regard these expit-transformed values as probabilities.

Whether the expit-transformed scores are probabilities could be arguable, but in the following experiments, I can show that even if we consider them as probabilities, they are uncalibrated, and thus require calibration before being used in high-stake applications.

## 7.4 Experiment and Results

I conducted experiments[4] to examine the following hypothesis:

1. Expit-transformed probabilities of current KG Embedding Models are uncalibrated, but off-the-shelf calibration techniques can effectively make the uncalibrated probabilities calibrated, producing more accurate probability estimations (see §7.4.1).

2. To perform calibration, Binning-based techniques, such as Isotonic Regression and Histogram Binning, generally work better than scaling-based ones (Platt

[3] These two models are not implemented in Ampligraph, so I used the PyKEEN [9] library implementations.

[4] Code available at `https://github.com/TREAT-UOE/kgcal`

| | $Train_E$ | $Train_C$ | $Test$ | Entities | Relations |
|---|---|---|---|---|---|
| FB13k | 316232 | 11816 | 47464 | 75043 | 13 |
| WN11 | 110361 | 4877 | 19706 | 38194 | 11 |
| YAGO39 | 354994 | 18471 | 18507 | 37612 | 37 |
| DBpedia50 | 32388 | 246 | 4196 | 24624 | 351 |
| UMLS | 5216 | 1304 | 1322 | 135 | 46 |
| Kinship | 8544 | 2136 | 2148 | 104 | 25 |

Table 7.1: Size of the datasets used in this experiment. The $Train_E$ sets are used to train embedding models, while $Train_C$ sets are just held-out triples that used to train calibration models.

Scaling and Beta Calibration) when large datasets (e.g. over 10k triples) are available (see §7.4.2).

Extending the setting of the previous work by Tabacof et al [183], in our experiment, I trained 4 typical KG embedding models, TransE [22], ComplEx [187], DistMult [211], and HoLE [140] on 6 datasets: FB13k [175], WN11 [175], YAGO39 [56], DBpedia50 [172], Kinship [93], and UMLS [93]. Each dataset is split into 3 subsets for training, calibration, and testing. Statstics of datasets are summarised in Table 7.1. The calibration and testing sets of FB13, WN11 and YAGO39 have ground truth negative samples, while the other 4 don't. Therefore, I generated synthetic negative samples via corruption: randomly sampling triples and replacing the subject or object entities with some random entities. In all datasets, we have balanced positive and negative samples.

We used the implementation of Knowledge Graph Embedding Models from AmpliGraph[5] [41] and the implementation of calibration techniques from NetCal[6] [99]. Following the similar setting of Tabacof et al [183], I trained each model for 500 epochs to optimise the Negative Log Loss, using early-stopping to avoid over-fitting. The vector dimensionality is set to 100. I used the Adam optimiser [94] with an initial learning rate of $1e-4$.

[5] https://github.com/Accenture/AmpliGraph

[6] https://github.com/fabiankueppers/calibration-framework

### 7.4.1 Uncalibrated Probabilities

To evaluate hypothesis (1), our goal is to compare the expit-transformed probabilities and calibrated probabilities and show whether the former incurs higher calibration errors. Firstly, I trained KG embedding models on a training set ($Train_E$) and computed the expit-transformed probabilities of triples in the test set. Specifically, I doubled the expit-transformed values of TransE so that the range of them is turned from $[0, 0.5]$ to $[0, 1]$. Then, I trained a calibration model on a held-out set ($Train_C$) and obtained the calibrated probabilities of triples in the testing set via the calibration model. I compared the expit-transformed probabilities and the calibrated probabilities in Figure 7.1, which illustrates that expit-transformed values get higher ECEs and BSs than calibrated ones, meaning that the KG embedding models are more or less uncalibrated, and almost all calibration techniques produced better-calibrated probabilities than the expit-transformed ones.

The results suggest that calibration is a better way than expit transform to convert embedding scores into more calibrated and accurate probabilities. Expit-transformed probabilities, after the range adapted to [0,1], should be used only when no extra data (the calibration set $Train_C$) is available to train a calibration model.

### 7.4.2 Binning-based Calibration

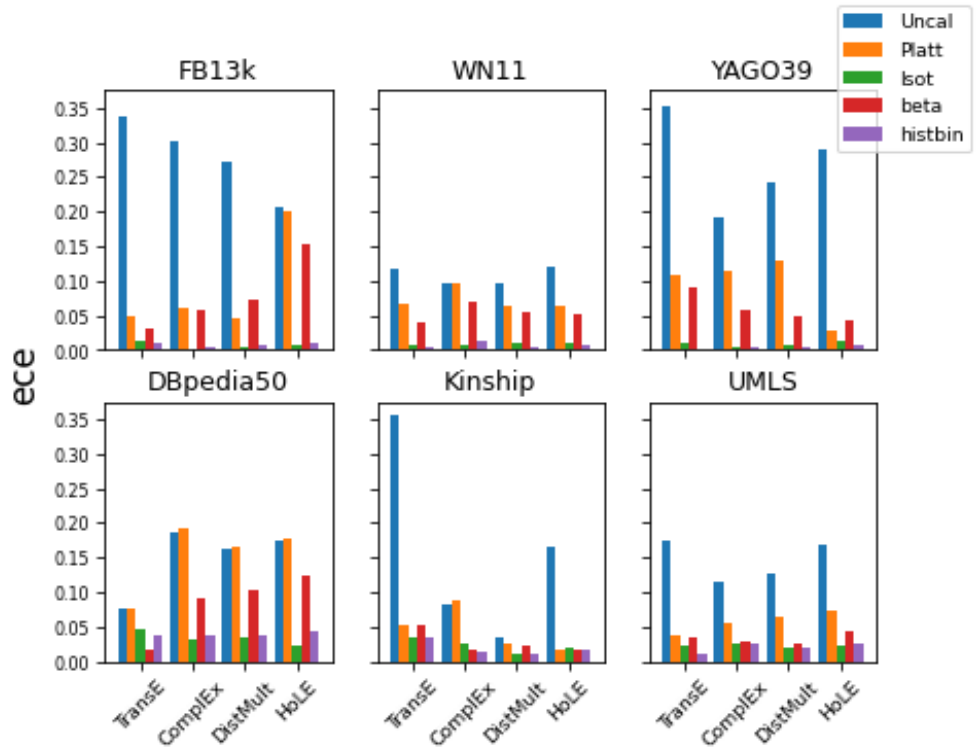
During the experiment, we can observe that binning-based calibration (Isotonic and Histogram) performs better in general. We can also notice that binning-based methods dominated in FB13k, WN11 and YAGO39, which has more data than the rest. Previous work also suggested that binning-based methods tend to overfit, especially on smaller datasets [141]. Thus, to evaluate hypothesis (2), we took these 3 datasets, and gradually shrunk the size of the calibration sets by randomly sampling k% of them, and comparing the number of wins in terms of BS, NLL, and ECE between binning-based and scaling-based methods. I plotted the results in Figure 7.3.

Results show that the performance of binning-based calibration techniques dominates at the beginning. As the size of the calibration sets shrinks, the winning count of Isotonic Regression and Histogram Binning decreases, while that of Platt Scaling and Beta Calibration increases. This implies that we should prefer binning-based calibration when large datasets are available (e.g. over 10k triples). When the dataset is relatively small, determining which calibration technique is better requires careful empirical evaluation.

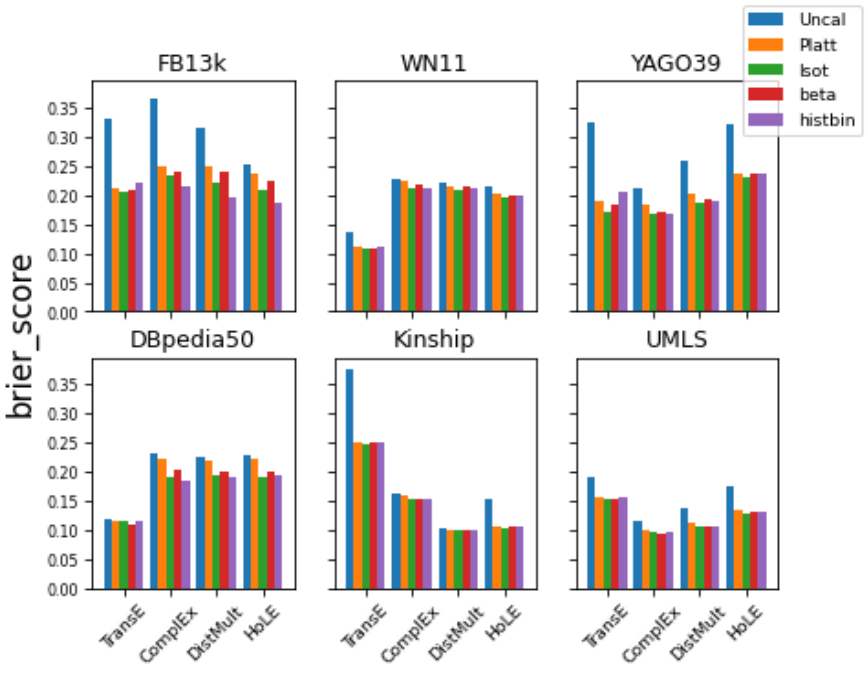
## 7.5 Summary and Discussion

In this chapter, I presented a closer look at probability calibration of knowledge graph embedding. I stressed that not all expit-transformed scores are appropriate to be interpreted as probabilities. What is worse, probabilities obtained by expit transform are generally uncalibrated for various KG embedding scores on various datasets. However, off-the-shelf calibration techniques can effectively calibrate these probabilities. If large datasets (over 10k triples) are available, binning-based techniques, including Isotonic Regression and Histogram Binning produce the best calibrated probabilities. In the long run, we will still need to compare the usefulness of probability against other kinds of uncertainties, like possibility [152, 153] and fuzziness [145, 179]. What's more, in this research I only focused on those widely used embedding models. In the future, I will look at the recently proposed models, such as DualE [28] and JointE [218].

Though probability calibration is a great tool to help assign probabilities to KG triples, unfortunately, I found that inapplicable in TREAT. The main reason is that assigning probabilities to KG triples by calibration would require training a KG embedding model first, and then scoring each triple, after which calibration can turn the scores into probabilities. Yet the KG of the TREAT framework keeps updated and evolving along with the assisted software system. Each time the KG gets updated, the embedding model will need to be retrained, which can incur unaffordable computing costs. Considering the weakness, eventually, I decided not to integrate the calibration technique into the TREAT framework.




(a) Expected Calibration Error




(b) Brier Score

Figure 7.1: Bar charts of ECE and BS for the probabilities produced by expit transform and the probabilities produced by various calibration techniques per model per dataset. The smaller ECE or BS, the better calibrated.

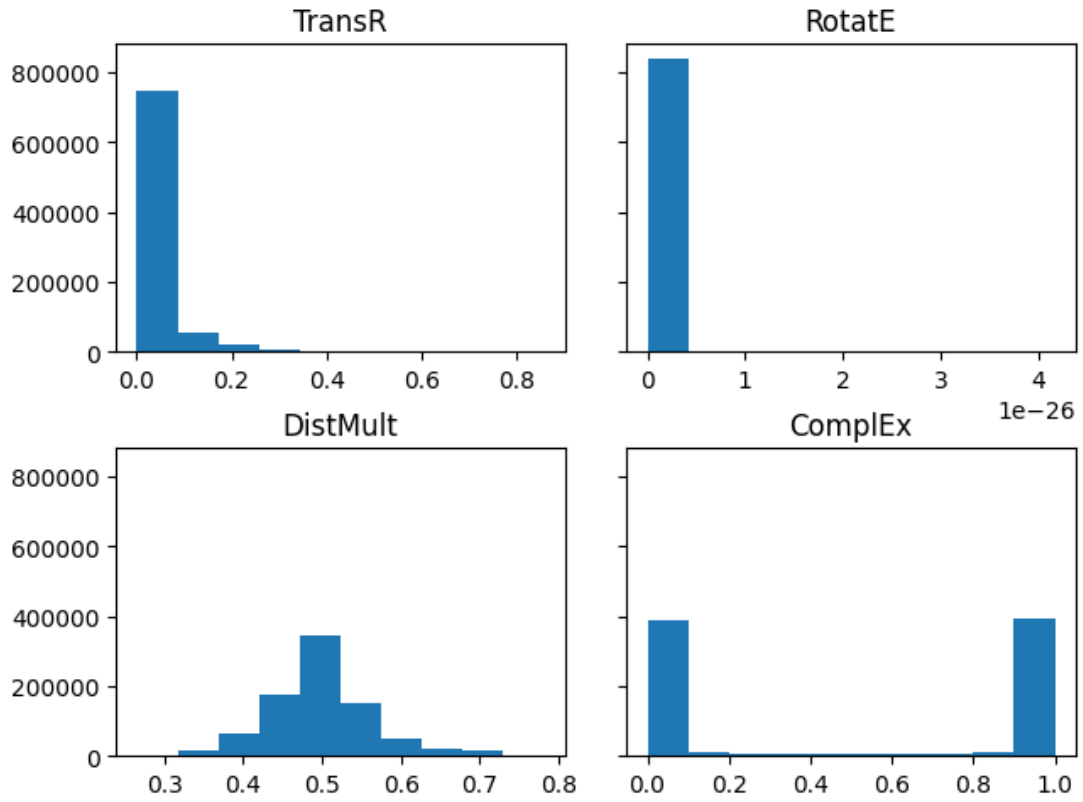



Figure 7.2: Histograms of doubled expit-transformed values of TransR, and RotatE, compared with DistMult and ComplEx (not doubled). Models were trained on UMLS dataset, optimising the NLL loss, with 500 epochs and early-stopping trick.

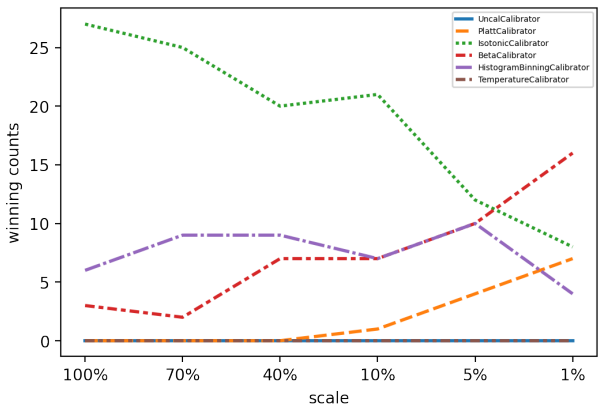



Figure 7.3: Number of winning counts for different calibration techniques for the 4 KG embedding models when the calibration sets of FB13, WN11, and YAGO 39 shrink. For each calibration result, we compute all the 3 metrics (BS, NLL, and ECE), so that every bin in the figure has 36 counts in total.

# Chapter 8

# Conclusion

In this thesis, motivated by the problem of more and more difficult O&M tasks in increasingly complex large software systems, we explored the approach of KBs construction and updating from system log streams, which we believed to be under-explored yet effective, efficient and low-cost.

## 8.1 Contributions

We presented the TREAT framework (§4) which is able to extract knowledge from a stream of logs and maintain a self-evolving knowledge base. The TREAT framework can serve as an intelligence engine plugged into any large software system that continually produces a stream of logs. TREAT then takes this log stream as input, extracts structured knowledge from the logs via log template mining and parsing, and then builds and updates a knowledge base via updating rules.

The TREAT framework is a "proof-by-construction" to the hypothesis that *it is feasible to extract knowledge from a stream of system logs, and keep track of the system changes by continually updating a knowledge base, thus supporting various downstream applications*. To the best of our knowledge, compared with previous works in the research community (§2), the TREAT framework is the first effort dedicated to extracting symbolic knowledge from logs and establishing a continually updating mechanism for addition/deletion of new knowledge once extracted.

We demonstrated the effectiveness of the TREAT framework by applying it in a typical downstream Operation & Maintenance task called fault localisation (§5). Empirical evidence shows that the TREAT-powered KB can successfully support a knowledge-based solution to the fault localisation task, meaning that the knowledge

extracted in the TREAT-powered KB is useful, and up-to-date along with the changes of its assisted system.

The TREAT framework also derived 2 other research contributions. Firstly, as the evaluation of TREAT by fault localisation relies on external factors, such as the Prolog-based fault localisation algorithm and our self-deployed test bed system, I would like to find a less dependent evaluation method. The one I produced is called the LP-Measure (§6), which assesses the quality of a knowledge graph by means of link prediction tasks, in particular the robustness and redundancy of the knowledge graph. The biggest advantage of our LP-Measure is that it can be run in a fully automated manner, without human labour to provide gold standards.

Secondly, during the research, I explored a potential way to quantify the uncertainty in knowledge extraction and assign probabilities to triples in the KB, I found probability calibration an interesting technique to convert scores with no probability semantics to mathematical probabilities. I took a further step to investigate probability calibration, and its application in knowledge graph embeddings (§7). With extensive experiments, I explored the phenomenon of uncalibrated probabilities in knowledge graph embeddings and provided empirical evidence for a useful rule of thumb for how to choose specific calibration techniques from the existing toolbox.

## 8.2 Limitations

The TREAT framework is far from a perfect approach. Recall that the updating mechanism of the TREAT-powered KB relies on a set of carefully defined updating rules, which are triggered on new extracted log events and issue updating operations upon the KB. It is a challenging problem to obtain these rules. Currently, we delegate this problem to human experts. This becomes the bottleneck of the whole framework, because the quality of extracted knowledge depends on the expert-crafted rules, and once new log types get introduced (software upgraded), the rules may not be able to cope with them.

What's more, the experiment conducted in our evaluation did not fully examine all aspects of the TREAT framework. For example, the test bed system I used for the experiment is quite small, so we can't be sure the limit of scalability of the TREAT framework can scale well to the size of truly large systems. Apart from scalability, industrial scenarios always bring about wield problems of "unknown unknown", which were not revealed in our experiments.

In terms of LP-Measure, we found that it is only applicable to large-scale world knowledge graphs or commonsense knowledge graphs such as Freebase, WordNet, and Yago, but not dynamic knowledge graphs that update frequently, such as the TREAT-powered KBs. Hence, we eventually cannot apply the developed method in evaluating the TREAT framework. Nevertheless, the LP-Measure is a novel and convenient method to probe the quality of large-scale world knowledge graphs. The same pity exists in our investigation of probability calibration in knowledge graph embeddings, as assigning probabilities to KG triples by probability calibration only applies to static KGs, not dynamic and ever-evolving ones like the TREAT-powered KBs. But fortunately, the findings we made may still benefit the general research community.

## 8.3 Future Work

There are several potential future works remaining to be done. Regarding the limitations of the TREAT framework, since it still demands some human labour and expertise, especially in manually crafting the updating rules for knowledge base updating, it is desirable to improve it to be a fully automated framework, at least to increment the level of automation. An apparent direction is to exploit the recently developed large language model technologies, such as prompt engineering or fine-tuning a foundation model [111]. Besides, more extensive experiments need to be conducted in order to thoroughly examine the potential and limits of the TREAT framework in various aspects, e.g., generality and scalability. we are keen to establish deeper collaboration with companies that allow us to deploy and test of methods in their industrial test bed and products. What's more, as TREAT is a complicated framework, it would always be a good idea to think about developing easier methods of evaluation, which boost the speed of feedback collection and iterative development.

# Glossaries

**5G** The 5th Generation Mobile Communication System.. 19

**AD** Anomaly Detection. A typical Operation and Maintenance job of detecting whether an anomaly happens within the running system.. 2

**FL** Fault Localisation. A typical Operation and Maintenance job of localising the faulty components when a system fault occurs.. iii, 2

**ILP** Inductive Logic Programming. This is the task of inducing a set of rules given a set of facts and desired (plus undesired) logical consequences. ILP could be regarded as the reverse problem of logic programming.. 51

**KB** Knowledge Base. A KB is an organised collection of facts about the domain of interest. Though the term is loosely coined, it usually refers to a structured representation of symbolic/logical statements. Initially, it is a used to describe the early Expert Systems. . 1, 14, 97, 98

**KG** Knowledge Graph. A KG is a type of knowledge base to organise knowledge as entities and their relations in multi-graph representation. . 7

**LLM** Large Language Model. As the name indicate, LLMs are language models in a large scale. The recent LLMs that achieve huge success are those neural models based on the Transformer architecture [192] with billions of neural weight parameters. . 9–11

**LP** Logic Programming. A paradigm of programming in logic. It is a declarative programming paradigm that uses a set of facts and rules to express computational problems and invoke an inference engine to solve the problems.. 51

**NF** Network Function. The basic unit of functionality of 5G systems, including Access Management Function (AMF), Session Management Function (SMF), and Network Repository Functions (NRF). More NFs are defined in relative 3GPP standards.. 19

**O&M** Operation and Maintenance. The general terms to refer a collection of jobs to operate and maintain a complex system. Typical jobs include fault diagnosis and resource planning, etc.. 1, 19, 97

**RCA** Root Cause Analysis. A typical Operation and Maintenance job of finding out the root cause of a fault when it happens.. 2

# Appendix A

# Worked Example

Below is a worked example of how my implemented TREAT framework extracts knowledge from log data, continually updates the knowledge, and uses the TREAT-powered knowledge base to address the fault localisation problem. As mentioned in Chapter 5.1, I will demonstrate this case study based on the Open5GS testbed system.

We run the testbed system and use a script to continuously simulate user operations. Occasionally, The script also injects some faults $(\tau_1, S_1), ..., (\tau_m, S_m)$ into the test bed system, where $tau_i$ represent time and $S_i$ represent the faulty component, e.g., AMF, SMF, and other network functions of Open5GS. The previous fault would be corrected before a new fault is injected, so that at the same time at most 1 component becomes faulty.

Listing A.1: fault injection records

```
2023-08-07 19:28:17,616 - INJECT: nssf
2023-08-07 19:28:43,856 - USER REQUEST
2023-08-07 19:29:03,035 - RECOVERED: nssf
2023-08-07 19:29:31,052 - INJECT: amf
2023-08-07 19:30:00,630 - RECOVERED: amf
2023-08-07 19:30:15,646 - USER REQUEST
2023-08-07 19:30:34,180 - INJECT: nssf
2023-08-07 19:31:16,200 - USER REQUEST
2023-08-07 19:32:02,722 - USER REQUEST
2023-08-07 19:32:32,855 - RECOVERED: nssf
```

During this process, the logs generated by each component are collected and stored in respective log files. Here is an excerpt of the logs from `amf.log`.

Listing A.2: example logs from amf.log

```
01/24 10:36:07.280: [app] INFO: Configuration: '/open5gs/
   install/etc/open5gs/amf.yaml' (../lib/app/ogs-init.c
   :126)
01/24 10:36:07.280: [app] INFO: File Logging: '/open5gs/
   install/var/log/open5gs/amf.log' (../lib/app/ogs-init.c
   :129)
01/24 10:36:07.286: [metrics] INFO: metrics_server() [http
   ://172.20.0.210]:9091 (../lib/metrics/prometheus/context
   .c:510)
01/24 10:36:07.286: [sbi] INFO: NF Service [namf-comm] (../
   lib/sbi/context.c:1334)
01/24 10:36:07.286: [sbi] INFO: nghttp2_server() [http
   ://172.20.0.210]:7777 (../lib/sbi/nghttp2-server.c:237)
01/24 10:36:07.286: [amf] INFO: ngap_server()
   [172.20.0.210]:38412 (../src/amf/ngap-sctp.c:61)
01/24 10:36:07.286: [sctp] INFO: AMF initialize...done (../
   src/amf/app.c:33)
01/24 10:36:07.288: [sbi] INFO: [e8ff155e-9bd2-41ed-9a11
   -0158331531c3] NF registered [Heartbeat:10s] (../lib/sbi
   /nf-sm.c:214)
01/24 10:36:16.322: [sbi] INFO: [ee622a4a-9bd2-41ed-989a-
   bd451676dded] (NRF-notify) NF registered (../lib/sbi/
   nnrf-handler.c:632)
01/24 10:36:16.322: [sbi] INFO: [ee622a4a-9bd2-41ed-989a-
   bd451676dded] (NRF-notify) NF Profile updated (../lib/
   sbi/nnrf-handler.c:642)
01/24 10:36:18.238: [amf] INFO: gNB-N2 accepted
   [172.20.0.223]:54772 in ng-path module (../src/amf/ngap-
   sctp.c:113)
```

## A.1 Knowledge Extraction and Updating

The first step of TREAT is to mine log templates from the collected logs. Below are some example log templates mined from the above logs. Note that the templates are represented as JSON key-value objects, whose keys are the log type and values are the corresponding log template, with several instances of the template placeholders.

Listing A.3: log templates

```
{
  "log_type_1": {
    "template": "Configuration: <:*:>",
    "extracted": [
      [
        "'/open5gs/install/etc/open5gs/amf.yaml'"
      ],
      [
        "'/open5gs/install/etc/open5gs/smf.yaml'"
      ],
      [
        "'/open5gs/install/etc/open5gs/udm.yaml'"
      ]
    ]
  },
  "log_type_2": {
    "template": "File Logging: <:*:>",
    "extracted": [
      [
        "'/open5gs/install/var/log/open5gs/amf.log'"
      ],
      [
        "'/open5gs/install/var/log/open5gs/smf.log'"
      ],
      [
        "'/open5gs/install/var/log/open5gs/udm.log'"
      ]
    ]
```

```
},
"log_type_8": {
  "template": "[<:*:>] NF registered [Heartbeat:10s]",
  "extracted": [
    [
      "[e8ff155e-9bd2-41ed-9a11-0158331531c3]"
    ],
    [
      "[c8a841ba-9d8d-41ed-9e3e-a3911d88e970]"
    ],
    [
      "[63483ac2-b41d-41ed-a57f-d1adf4fc2ee6]"
    ]
  ]
},
"log_type_13": {
  "template": "[Added] Number of <:*:> is now <:*:>",
  "extracted": [
    [
      "gNBs",
      "1"
    ],
    [
      "gNB-UEs",
      "1"
    ],
    [
      "AMF-UEs",
      "1"
    ]
  ]
},
"log_type_35": {
  "template": "[<:*:>] NF de-registered",
  "extracted": [
```

```
          [
            "[5dd8edb8-b420-41ed-a8e5-f34e82ad16f1]"
          ],
          [
            "[5dd8edb8-b420-41ed-a8e5-f34e82ad16f1]"
          ],
          [
            "[961f63aa-f750-41ed-9c48-87a0bd6fb563]"
          ]
        ]
      }
    }
```

So far, the outputs of the mining step are the log templates with nameless placeholders, such as `"Configuration: <:*:>"`. We want to assign names to the placeholders ”¡:*:¿” so that in the following steps we can extract key-value pairs from the log messages with the aid of these annotated templates. we can simply ask human experts to annotate the wildcards with meaningful names. This is a bit ad-hoc, but it is a one-time job. We can also prompt LLMs, such as GPT[1] and Claude[2], to annotate the placeholders first, and then ask human experts to review and make minor revisions, which can further reduce human workload. Below is the prompt I used to ask GPT-4.

```
Here is a log template with placeholders "<:*:>":
   {template}
Here are some example information that instantiate the
   placeholders:
   {instances}
What is the best possible name for the placeholders? Please
   directly output the names separated by comma, e.g., "name1,
   name2".
```

The outputs of GPT-4 are appended to the templates in the `fields`:

Listing A.4: log templates

```
{
  "log_type_1": {
```

[1] https://chatgpt.com/
[2] https://claude.ai/

```
    "template": "Configuration: <:*:>",
    "extracted": [
      [
        "'/open5gs/install/etc/open5gs/amf.yaml'"
      ],
      [
        "'/open5gs/install/etc/open5gs/amf.yaml'"
      ],
      [
        "'/open5gs/install/etc/open5gs/amf.yaml'"
      ]
    ],
    "fields": [
      "config_file_path"
    ]
  },
  "log_type_2": {
    "template": "File Logging: <:*:>",
    "extracted": [
      [
        "'/open5gs/install/var/log/open5gs/amf.log'"
      ],
      [
        "'/open5gs/install/var/log/open5gs/smf.log'"
      ],
      [
        "'/open5gs/install/var/log/open5gs/udm.log'"
      ]
    ],
    "fields": [
      "log_file_path"
    ]
  },
  "log_type_8": {
    "template": "[<:*:>] NF registered [Heartbeat:10s]",
```

```
      "extracted": [
        [
          "[e8ff155e-9bd2-41ed-9a11-0158331531c3]"
        ],
        [
          "[c8a841ba-9d8d-41ed-9e3e-a3911d88e970]"
        ],
        [
          "[63483ac2-b41d-41ed-a57f-d1adf4fc2ee6]"
        ]
      ],
      "fields": [
        "nf_instance_id"
      ]
    },
    "log_type_13": {
      "template": "[Added] Number of <:*:> is now <:*:>",
      "extracted": [
        [
          "gNBs",
          "1"
        ],
        [
          "gNB-UEs",
          "1"
        ],
        [
          "AMF-UEs",
          "1"
        ]
      ],
      "fields": [
        "item_type",
        "item_count"
      ]
```

```
  },
  "log_type_35": {
    "template": "[<:*:>] NF de-registered",
    "extracted": [
      [
        "[5dd8edb8-b420-41ed-a8e5-f34e82ad16f1]"
      ],
      [
        "[5dd8edb8-b420-41ed-a8e5-f34e82ad16f1]"
      ],
      [
        "[961f63aa-f750-41ed-9c48-87a0bd6fb563]"
      ]
    ],
    "fields": [
      "nf_instance_id"
    ]
  }
}
```


Afterward, we can make use of these annotated log templates to parse the log messages into structured log events:

Listing A.5: Extracted log events


```
{"log_id": "evt_9b8e88be-3f6c-4020-b933-c813ca68aa8e", "
   timestamp": 1674556567280, "component": "app", "severity
   ": "INFO", "codeline": "../lib/app/ogs-init.c:126", "
   log_type_id": "1", "config_file_path": "'/open5gs/
   install/etc/open5gs/amf.yaml'"}
{"log_id": "evt_c3c8b9b2-a2ac-44d2-892b-2f6fcb75d503", "
   timestamp": 1674556567280, "component": "app", "severity
   ": "INFO", "codeline": "../lib/app/ogs-init.c:129", "
   log_type_id": "2", "log_file_path": "'/open5gs/install/
   var/log/open5gs/amf.log'"}
{"log_id": "evt_461e539d-c5f0-436f-8660-c4d2594d814e", "
   timestamp": 1674556567286, "component": "metrics", "
```

severity": "INFO", "codeline": "../lib/metrics/
prometheus/context.c:510", "log_type_id": "3", "ip":
"172.20.0.210"}
{"log_id": "evt_9ab0c803-5a34-46c8-9851-79607a37f6d1", "
timestamp": 1674556567286, "component": "sbi", "severity
": "INFO", "codeline": "../lib/sbi/context.c:1334", "
log_type_id": "4", "service_name": "namf-comm"}
{"log_id": "evt_5f83d3b8-4ad7-421e-9646-d1fadfad3a93", "
timestamp": 1674556567288, "component": "sbi", "severity
": "INFO", "codeline": "../lib/sbi/nf-sm.c:214", "
log_source": "amf.log", "log_type_id": "8", "
nf_instance_id": "e8ff155e-9bd2-41ed-9a11-0158331531c3"}

The log events parsed from log messages in the above step will be fed to an event processing engine that captures the changes in the system from the log events and performs relative changes to the knowledge base. The event processing engine relies on a set of predefined updating rules. Updating rules connect log events and updating events. For simplicity, here we design such rules, which focus on tracking the addition and deletion of network functions:

Listing A.6: updateing rules

```
{
    "log_type_8": {
        "add": [
            "nf_{nf_instance_id};typeOf;{log_file}"
        ]
    },
    "log_type_33": {
        "remove_entity": ["nf_{nf_instance_id}"]
    },
    "log_type_34": {
        "remove_entity": ["nf_{nf_instance_id}"]
    },
    "log_type_38": {
        "remove_entity": ["nf_{nf_instance_id}"]
    }
```

```
}
```

Each rule is a map from a log event type (log template) to a set of updating operations. For example, the first rule says that when a log comes and its type (template) is `log_type_8`, then a triple `(nf_{nf_instance_id}; typeOf; {log_file})` is initiated. The symbols in the curly brackets are parameters that will be instantiated by the extracted log parameters. Ideally, these rules should be obtained in an automated manner. However, this is a huge challenge, and currently, we can only obtain high-quality rules manually. In the future, we will explore how to derive usable rules via LLM technologies.

When all the updating rules are ready, we can execute the event processing pipeline. When the event processing engine keeps consuming the events, its maintained knowledge base also keeps being updated. We may dump the snapshots of the evolving knowledge base at some desired time points, which can serve as the knowledge representation of the system at the corresponding timepoints. Below is the dumped snapshot of the knowledge base after the TREAT framework processes all the log events.


```
{
    "entities": [
      "ausf",
      "scp",
      "nf_7357ad38_3566_41ee_8b3b_55200db0a294",
      "nrf",
      "nf_16080ef8_356a_41ee_ac4b_6f21805f7d19",
      "udr",
      "amf",
      "udm",
      "nf_cdd54ba6_9d8d_41ed_a30a_45676ecf6589",
      "nf_b9ddadc8_3568_41ee_b59c_2fee4387a34e",
      "nf_3d40bc78_3569_41ee_9994_530712d94060",
      "nssf",
      "nf_c6edf27a_9d8d_41ed_96db_73930f919161",
      "pcf",
      "nf_864b66ac_3569_41ee_8fac_7f2339d9a9b0",
      "smf",
      "nf_89ce4084_3568_41ee_a169_0598f8d6b34f",
```

```
      "nf_a0e74ecc_3569_41ee_a561_21c4794bc1ef"
    ],
    "predicates": [
      "typeOf"
    ],
    "triples": [
      "nf_c6edf27a_9d8d_41ed_96db_73930f919161;typeOf;nrf",
      "nf_16080ef8_356a_41ee_ac4b_6f21805f7d19;typeOf;nrf",
      "nf_864b66ac_3569_41ee_8fac_7f2339d9a9b0;typeOf;nrf",
      "nf_c6edf27a_9d8d_41ed_96db_73930f919161;typeOf;scp",
      "nf_cdd54ba6_9d8d_41ed_a30a_45676ecf6589;typeOf;nrf",
      "nf_7357ad38_3566_41ee_8b3b_55200db0a294;typeOf;udr",
      "nf_7357ad38_3566_41ee_8b3b_55200db0a294;typeOf;nrf",
      "nf_89ce4084_3568_41ee_a169_0598f8d6b34f;typeOf;nrf",
      "nf_a0e74ecc_3569_41ee_a561_21c4794bc1ef;typeOf;nrf",
      "nf_b9ddadc8_3568_41ee_b59c_2fee4387a34e;typeOf;nrf",
      "nf_a0e74ecc_3569_41ee_a561_21c4794bc1ef;typeOf;ausf",
      "nf_3d40bc78_3569_41ee_9994_530712d94060;typeOf;smf",
      "nf_3d40bc78_3569_41ee_9994_530712d94060;typeOf;nrf",
      "nf_89ce4084_3568_41ee_a169_0598f8d6b34f;typeOf;amf",
      "nf_864b66ac_3569_41ee_8fac_7f2339d9a9b0;typeOf;udm",
      "nf_16080ef8_356a_41ee_ac4b_6f21805f7d19;typeOf;nssf",
      "nf_b9ddadc8_3568_41ee_b59c_2fee4387a34e;typeOf;pcf"
    ]
  }
```


## A.2 Perform fault localisation

The dumped KB snapshots, together with the fault injection records, will serve as the learning samples and ground truths of machine learning. Here we use Inductive Logic Programming to perform fault localisation. Each time a fault occur in a system and localisation is required, we call that a *case*. Every dumped snapshot at the time of fault injection is also a case. Suppose we have two snapshots of the knowledge base, $KB_1$ and $KB_2$, and the faults that occurred are $fault_1$ and $fault_2$. The following code listing

shows the two cases represented in Prolog syntax.

```
% specific background knowledge for case 1
typeOf(case1, nf_d4a8bef4_0a1b_41ee, net_func) .
typeOf(case1, nf_c6edf27a_9d8d_41ed, net_func) .
typeOf(case1, nf_d48324f2_0a19_41ee, net_func) .
typeOf(case1, nf_3c68a5c8_0a1b_41ee, net_func) .
typeOf(case1, nf_aae405d8_0a1b_41ee, ausf) .
typeOf(case1, nf_e5b4272e_0a1b_41ee, pcf) .
typeOf(case1, nf_c6edf27a_9d8d_41ed, scp) .
typeOf(case1, nf_6b25cdd6_3550_41ee, net_func) .
typeOf(case1, nf_e5b4272e_0a1b_41ee, net_func) .
typeOf(case1, nf_d48324f2_0a19_41ee, udm) .
typeOf(case1, nf_3c68a5c8_0a1b_41ee, smf) .
typeOf(case1, nf_aae405d8_0a1b_41ee, net_func) .
typeOf(case1, nf_6b25cdd6_3550_41ee, amf) .
typeOf(case1, nf_cdd54ba6_9d8d_41ed, net_func) .
typeOf(case1, nf_d4a8bef4_0a1b_41ee, udr) .

% specific background knowledge for case 2
typeOf(case2, nf_d4a8bef4_0a1b_41ee, net_func) .
typeOf(case2, nf_c6edf27a_9d8d_41ed, net_func) .
typeOf(case2, nf_d48324f2_0a19_41ee, net_func) .
typeOf(case2, nf_3c68a5c8_0a1b_41ee, net_func) .
typeOf(case2, nf_aae405d8_0a1b_41ee, ausf) .
typeOf(case2, nf_e5b4272e_0a1b_41ee, pcf) .
typeOf(case2, nf_c6edf27a_9d8d_41ed, scp) .
typeOf(case2, nf_c5e117bc_3550_41ee, nssf) .
typeOf(case2, nf_e5b4272e_0a1b_41ee, net_func) .
typeOf(case2, nf_c5e117bc_3550_41ee, net_func) .
typeOf(case2, nf_d48324f2_0a19_41ee, udm) .
typeOf(case2, nf_3c68a5c8_0a1b_41ee, smf) .
typeOf(case2, nf_aae405d8_0a1b_41ee, net_func) .
typeOf(case2, nf_cdd54ba6_9d8d_41ed, net_func) .
typeOf(case2, nf_d4a8bef4_0a1b_41ee, udr) .
```

```
% positive and negative learning examples for case 1
positive(faulty(case1, nssf)). % the true fault of case 1
   is nf3
negative(faulty(case1, smf)). % the generated negative
   fault of case 1
negative(faulty(case1, pcf)).
negative(faulty(case1, ausf)).

% positive and negative learning examples for case 2
positive(faulty(case2, amf)).
negative(faulty(case2, udr)).
negative(faulty(case2, udm)).
negative(faulty(case2, nssf)).
```

Before we can run the ILP task, some background knowledge of the problem should be provided to the ILP engine. Here is the background knowledge we used. It just declares all the available network functions, and one inference rule saying that at some time is there is a triple *(Inst; typeOf; NF)* then *NF* is the network function used at *Case*.

Listing A.7: background knowledge for the fault localisation inductive logic programming

```
nf(amf).
nf(smf).
nf(udm).
nf(pcf).
nf(udr).
nf(ausf).
nf(nssf).
nf(scp).

inuse_nf(Case, NF) :- nf(NF), typeOf(Case, Inst, NF).
```

Feeding this background knowledge and learning examples to an ILP engine, the following inference rule will be induced:

Listing A.8: inference rules

```
faulty(A,B) :- \+inuse_nf(A,B), nf(B).
```

meaning that if B is a network function and it is not in use in case A, then it is the faulty component of case A. Once we induce these inference rules, we can use them to infer the faulty component of new cases by means of Logic Programming. Below is the example results of the new case `case999`

```
(logkg-rca2) D:\TREAT\logkg-fl\data\exp1\fl\test>cat temp.pl
typeOf(case999,nf_c6edf27a_9d8d_41ed_96db_73930f919161, nrf) .
typeOf(case999,nf_864b66ac_3569_41ee_8fac_7f2339d9a9b0, nrf) .
typeOf(case999,nf_c6edf27a_9d8d_41ed_96db_73930f919161, scp) .
typeOf(case999,nf_cdd54ba6_9d8d_41ed_a30a_45676ecf6589, nrf) .
typeOf(case999,nf_7357ad38_3566_41ee_8b3b_55200db0a294, udr) .
typeOf(case999,nf_7357ad38_3566_41ee_8b3b_55200db0a294, nrf) .
typeOf(case999,nf_89ce4084_3568_41ee_a169_0598f8d6b34f, nrf) .
typeOf(case999,nf_a0e74ecc_3569_41ee_a561_21c4794bc1ef, nrf) .
typeOf(case999,nf_b9ddadc8_3568_41ee_b59c_2fee4387a34e, nrf) .
typeOf(case999,nf_a0e74ecc_3569_41ee_a561_21c4794bc1ef, ausf) .
typeOf(case999,nf_3d40bc78_3569_41ee_9994_530712d94060, smf) .
typeOf(case999,nf_3d40bc78_3569_41ee_9994_530712d94060, nrf) .
typeOf(case999,nf_89ce4084_3568_41ee_a169_0598f8d6b34f, amf) .
typeOf(case999,nf_864b66ac_3569_41ee_8fac_7f2339d9a9b0, udm) .
typeOf(case999,nf_b9ddadc8_3568_41ee_b59c_2fee4387a34e, pcf) .

nf(amf).
nf(smf).
nf(udm).
nf(pcf).
nf(udr).
nf(ausf).
nf(nssf).
nf(scp).

% if there is a statement asserting an the NF type of an entity, then in this case
% this NF is in use
inuse_nf(Case, NF) :- typeOf(Case, _, NF), nf(NF).
faulty(A,B)  :-  \+inuse_nf(A,B), nf(B).

faulty(A,B)  :- nf(B),  \+inuse_nf(A,B).

(logkg-rca2) D:\TREAT\logkg-fl\data\exp1\fl\test>swipl temp.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 9.1.18)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

1 ?- faulty(case999, X).
X = nssf .
```

Figure A.1: An example of using the induced inference rule to infer the faulty component in a new case by Logic Programming.

# Bibliography

[1] 3GPP. 5g management and orchestration; levels of autonomous network. 3GPP Technical Specification, 2022. 3GPP TS 28.100 version 17.0.0.

[2] 3GPP. Architecture enhancements for 5g system (5gs) to support network data analytics services. 3GPP Technical Specification, 2022.

[3] Ralph Abboud, Ismail Ceylan, Thomas Lukasiewicz, and Tommaso Salvatori. Boxe: A box embedding model for knowledge base completion. *Advances in Neural Information Processing Systems*, 33:9649–9661, 2020.

[4] Ashutosh Adhikari, Achyudh Ram, Raphael Tang, and Jimmy Lin. Docbert: Bert for document classification. *arXiv preprint arXiv:1904.08398*, 2019.

[5] Ashutosh Adhikari, Xingdi Yuan, Marc-Alexandre Côté, Mikuláš Zelinka, Marc-Antoine Rondeau, Romain Laroche, Pascal Poupart, Jian Tang, Adam Trischler, and Will Hamilton. Learning dynamic belief graphs to generalize on text-based games. *Advances in Neural Information Processing Systems*, 33:3045–3057, 2020.

[6] Monica Agrawal, Stefan Hegselmann, Hunter Lang, Yoon Kim, and David Sontag. Large language models are few-shot clinical information extractors. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 1998–2022, 2022.

[7] Ekin Akyürek, Tolga Bolukbasi, Frederick Liu, Binbin Xiong, Ian Tenney, Jacob Andreas, and Kelvin Guu. Towards tracing knowledge in language models back to the training data. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 2429–2446, 2022.

[8] Carlos E Alchourrón, Peter Gärdenfors, and David Makinson. On the logic of

theory change: Partial meet contraction and revision functions. *The journal of symbolic logic*, 50(2):510–530, 1985.

[9] Mehdi Ali, Max Berrendorf, Charles Tapley Hoyt, Laurent Vermue, Sahand Sharifzadeh, Volker Tresp, and Jens Lehmann. PyKEEN 1.0: A Python Library for Training and Evaluating Knowledge Graph Embeddings. *Journal of Machine Learning Research*, 22(82):1–6, 2021.

[10] Gabor Angeli, Melvin Jose Johnson Premkumar, and Christopher D Manning. Leveraging linguistic structure for open domain information extraction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 344–354, 2015.

[11] Tarmo Anttalainen. *Introduction to telecommunications network engineering*. Artech House, 2003.

[12] Hiba Arnaout, Simon Razniewski, Gerhard Weikum, and Jeff Z Pan. Negative statements considered useful. *Journal of Web Semantics*, 71:100661, 2021.

[13] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. A simple but tough-to-beat baseline for sentence embeddings. In *International conference on learning representations*, 2017.

[14] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer, 2007.

[15] Franz Baader, Diego Calvanese, Deborah McGuinness, Peter Patel-Schneider, Daniele Nardi, et al. *The description logic handbook: Theory, implementation and applications*. Cambridge university press, 2003.

[16] Caleb Belth, Xinyi Zheng, Jilles Vreeken, and Danai Koutra. What is normal, what is strange, and what is missing in a knowledge graph: Unified characterization via inductive summarization. In *Proceedings of The Web Conference 2020*, pages 1115–1126, 2020.

[17] Magdalena Biesialska, Katarzyna Biesialska, and Marta R Costa-jussà. Continual lifelong learning in natural language processing: A survey. In *Proceed-*

*ings of the 28th International Conference on Computational Linguistics*, pages 6523–6541, 2020.

[18] Carlos Bobed, Pierre Maillot, Peggy Cellier, and Sébastien Ferré. Data-driven assessment of structural evolution of rdf graphs. *Semantic Web*, 11(5):831–853, 2020.

[19] Jasmin Bogatinovski, Sasho Nedelkoski, Alexander Acker, Florian Schmidt, Thorsten Wittkopp, Soeren Becker, Jorge Cardoso, and Odej Kao. Artificial intelligence for it operations (aiops) workshop white paper. *arXiv preprint arXiv:2101.06054*, 2021.

[20] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[21] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250, 2008.

[22] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26, 2013.

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

[24] Alan Bundy and Michael Chan. Towards ontology evolution in physics. In *International Workshop on Logic, Language, Information, and Computation*, pages 98–110. Springer, 2008.

[25] Chang Cai, Zhengyi Jiang, Hui Wu, Junsheng Wang, Jiawei Liu, and Lei Song. Research on knowledge graph-driven equipment fault diagnosis method for intelligent manufacturing. *The International Journal of Advanced Manufacturing Technology*, 130(9):4649–4662, 2024.

[26] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, 2018.

[27] Ruichu Cai, Siyu Wu, Jie Qiao, Zhifeng Hao, Keli Zhang, and Xi Zhang. Thps: Topological hawkes processes for learning causal structure on event sequences. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[28] Zongsheng Cao, Qianqian Xu, Zhiyong Yang, Xiaochun Cao, and Qingming Huang. Dual quaternion knowledge graph embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 6894–6902, 2021.

[29] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R Hruschka, and Tom M Mitchell. Toward an architecture for never-ending language learning. In *Twenty-Fourth AAAI conference on artificial intelligence*, 2010.

[30] Michael Chan, Jos Lehmann, and Alan Bundy. Galileo: A system for automating ontology evolution. *ARCOE-11*, page 46, 2011.

[31] Chia-Hui Chang, Mohammed Kayed, Moheb R Girgis, and Khaled F Shaalan. A survey of web information extraction systems. *IEEE transactions on knowledge and data engineering*, 18(10):1411–1428, 2006.

[32] Linlin Chao, Jianshan He, Taifeng Wang, and Wei Chu. Pairre: Knowledge graph embeddings via paired relation vectors. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4360–4369, 2021.

[33] Haihua Chen, Gaohui Cao, Jiangping Chen, and Junhua Ding. A practical framework for evaluating the quality of knowledge graph. In *Knowledge Graph and Semantic Computing: Knowledge Computing and Language Understanding: 4th China Conference, CCKS 2019, Hangzhou, China, August 24–27, 2019, Revised Selected Papers 4*, pages 111–122. Springer, 2019.

[34] Jiaoyan Chen, Pan Hu, Ernesto Jimenez-Ruiz, Ole Magnus Holter, Denvar Antonyrajah, and Ian Horrocks. Owl2vec*: Embedding of owl ontologies. *Machine Learning*, 110(7):1813–1845, 2021.

[35] Yuanfang Chi, Yanjie Dong, Z Jane Wang, F Richard Yu, and Victor CM Leung. Knowledge-based fault diagnosis in industrial internet of things: A survey. *IEEE Internet of Things Journal*, 9(15):12886–12900, 2022.

[36] Bonggeun Choi, Daesik Jang, and Youngjoong Ko. Mem-kgc: Masked entity model for knowledge graph completion with pre-trained language model. *IEEE Access*, 9:132025–132032, 2021.

[37] Ching-Yao Chuang, Joshua Robinson, Yen-Chen Lin, Antonio Torralba, and Stefanie Jegelka. Debiased contrastive learning. *Advances in neural information processing systems*, 33:8765–8775, 2020.

[38] Keith L Clark. Negation as failure. *Logic and data bases*, pages 293–322, 1978.

[39] William F Clocksin and Christopher S Mellish. *Programming in PROLOG*. Springer Science & Business Media, 2003.

[40] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20:273–297, 1995.

[41] Luca Costabello, Sumit Pai, Chan Le Van, Rory McGrath, Nicholas McCarthy, and Pedro Tabacof. AmpliGraph: a Library for Representation Learning on Knowledge Graphs, March 2019.

[42] Marc-Alexandre Côté, Akos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, et al. Textworld: A learning environment for text-based games. In *Computer Games: 7th Workshop, CGW 2018, Held in Conjunction with the 27th International Conference on Artificial Intelligence, IJCAI 2018, Stockholm, Sweden, July 13, 2018, Revised Selected Papers 7*, pages 41–75. Springer, 2019.

[43] Jim Cowie and Wendy Lehnert. Information extraction. *Communications of the ACM*, 39(1):80–91, 1996.

[44] Andrew Cropper and Sebastijan Dumančić. Inductive logic programming at 30: a new introduction. *Journal of Artificial Intelligence Research*, 74:765–850, 2022.

[45] Gianpaolo Cugola and Alessandro Margara. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys (CSUR)*, 44(3):1–62, 2012.

[46] John Dagdelen, Alexander Dunn, Sanghoon Lee, Nicholas Walker, Andrew S Rosen, Gerbrand Ceder, Kristin A Persson, and Anubhav Jain. Structured information extraction from scientific text with large language models. *Nature Communications*, 15(1):1418, 2024.

[47] Yingnong Dang, Qingwei Lin, and Peng Huang. Aiops: real-world challenges and research innovations. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 4–5. IEEE, 2019.

[48] Rajarshi Das, Tsendsuren Munkhdalai, Xingdi Yuan, Adam Trischler, and Andrew McCallum. Building dynamic knowledge graphs from text using machine reading comprehension. In *International Conference on Learning Representations*, 2019.

[49] A. P. Dawid. Calibration-based empirical probability. *The Annals of Statistics*, 13(4):1251–1274, 1985.

[50] Nicola De Cao, Wilker Aziz, and Ivan Titov. Editing factual knowledge in language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6491–6506, 2021.

[51] Luc De Raedt. Logical settings for concept-learning. *Artificial Intelligence*, 95(1):187–201, 1997.

[52] Luc De Raedt and Luc Dehaspe. Learning from satisfiability. In *Ninth Dutch Conference on Artificial Intelligence (NAIC'97)*, pages 303–312, 1997.

[53] Luc De Raedt, Anton Dries, Ingo Thon, Guy Van den Broeck, and Mathias Verbeke. Inducing probabilistic relational rules from probabilistic examples. In *Proceedings of 24th international joint conference on artificial intelligence (IJCAI)*, volume 2015, pages 1835–1842. IJCAI-INT JOINT CONF ARTIF INTELL, 2015.

[54] Jeremy Debattista, Sören Auer, and Christoph Lange. Luzzu—a methodology and framework for linked data quality assessment. *Journal of Data and Information Quality (JDIQ)*, 8(1):1–32, 2016.

[55] Matt Dennis, Kees Van Deemter, Daniele Dell'Aglio, and Jeff Z Pan. Computing authoring tests from competency questions: experimental validation. In

*The Semantic Web–ISWC 2017: 16th International Semantic Web Conference, Vienna, Austria, October 21–25, 2017, Proceedings, Part I 16*, pages 243–259. Springer, 2017.

[56] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018.

[57] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[58] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 601–610, 2014.

[59] Nick Drummond and Rob Shearer. The open world assumption. In *eSI Workshop: The Closed World of Databases meets the Open World of the Semantic Web*, volume 15, page 1, 2006.

[60] Min Du and Feifei Li. Spell: Streaming parsing of system event logs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 859–864. IEEE, 2016.

[61] Andreas Ekelhart, Fajar J Ekaputra, and Elmar Kiesling. The slogert framework for automated log knowledge graph construction. In *European Semantic Web Conference*, pages 631–646. Springer, 2021.

[62] ETSI. Gana - generic autonomic networking architecture, 2016. ETSI White Paper No. 16.

[63] ETSI. Autonomous networks, supporting tomorrow's ict business, 2020. ETSI White Paper No. 40.

[64] ETSI. Vocabulary for 3gpp specifications. White Paper, 2022. 3GPP TR 21.905.

[65] ETSI. Experiential networked intelligence (eni); representing, inferring, and proving knowledge in eni. ETSI Group Specification, 2023. ETSI GS ENI 019 V3.1.1.

[66] ETSI. Unlocking digital transformation with autonomous networks, 2023. ETSI White Paper No. 56.

[67] ETSI. Zero-touch network and service management (zsm); intent-driven autonomous networks; generic aspects. ETSI Group Report, 2023. ETSI GR ZSM 011 V1.1.1.

[68] Daniel Faria, Alfio Ferrara, Ernesto Jiménez-Ruiz, Stefano Montanelli, and Catia Pesquita. Crowd-assessing quality in uncertain data linking datasets. *The Knowledge Engineering Review*, 35:e33, 2020.

[69] William Feller and Philip M Morse. An introduction to probability theory and its applications, 1958.

[70] Dieter Fensel, Monika Zickwolff, Markus Wiese, et al. Are substitutions the better examples? learning complete sets of clauses with frog. In *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 453–474. Citeseer, 1995.

[71] TM Forum. Autonomous networks: Empowering digital transformation for the telecoms industry. TM Forum White Paper, 2019.

[72] Qiang Fu, Jian-Guang Lou, Yi Wang, and Jiang Li. Execution anomaly detection in distributed systems through unstructured log analysis. In *2009 ninth IEEE international conference on data mining*, pages 149–158. IEEE, 2009.

[73] Junyang Gao, Xian Li, Yifan Ethan Xu, Bunyamin Sisman, Xin Luna Dong, and Jun Yang. Efficient knowledge graph accuracy evaluation. *Proceedings of the VLDB Endowment*, 12(11), 2019.

[74] Peter Gärdenfors. *Belief Revision*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1992.

[75] Ralph Grishman. Information extraction. *IEEE Intelligent Systems*, 30(5):8–15, 2015.

[76] Nicola Guarino and Christopher Welty. Evaluating ontological decisions with ontoclean. *Communications of the ACM*, 45(2):61–65, 2002.

[77] Liangke Gui, Borui Wang, Qiuyuan Huang, Alexander G Hauptmann, Yonatan Bisk, and Jianfeng Gao. Kat: A knowledge augmented transformer for vision-and-language. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 956–968, 2022.

[78] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International conference on machine learning*, pages 1321–1330. PMLR, 2017.

[79] Xiaoqi Han, Ru Li, Hongye Tan, Wang Yuanlong, Qinghua Chai, and Jeff Pan. Improving sequential model editing with fact retrieval. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 11209–11224, 2023.

[80] Pinjia He, Jieming Zhu, Pengcheng Xu, Zibin Zheng, and Michael R Lyu. A directed acyclic graph approach to online log parsing. *arXiv preprint arXiv:1806.04356*, 2018.

[81] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE international conference on web services (ICWS)*, pages 33–40. IEEE, 2017.

[82] Shilin He, Pinjia He, Zhuangbin Chen, Tianyi Yang, Yuxin Su, and Michael R Lyu. A survey on automated log analysis for reliability engineering. *ACM computing surveys (CSUR)*, 54(6):1–37, 2021.

[83] Pascal Hitzler, Aaron Eberhart, Monireh Ebrahimi, Md Kamruzzaman Sarker, and Lu Zhou. Neuro-symbolic approaches in artificial intelligence. *National Science Review*, 9(6):nwac035, 2022.

[84] Alfred Horn. On sentences which are true of direct unions of algebras1. *The Journal of Symbolic Logic*, 16(1):14–21, 1951.

[85] Alan Hájek. Interpretations of Probability. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Fall 2019 edition, 2019.

[86] Katsumi Inoue, Tony Ribeiro, and Chiaki Sakama. Learning from interpretation transition. *Machine Learning*, 94:51–79, 2014.

[87] ITU-T. Architecture framework for autonomous networks. ITU-T Technical Specification, 2022.

[88] ITU-T. Trustworthiness evaluation for autonomous networks including imt-2020 and beyond. ITU-T Technical Specification, 2022.

[89] Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. Few-shot learning with retrieval augmented language models. *arXiv preprint arXiv:2208.03299*, 2022.

[90] Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. Knowledge graph embedding via dynamic mapping matrix. In *Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing (volume 1: Long papers)*, pages 687–696, 2015.

[91] Shengbin Jia, Yang Xiang, Xiaojun Chen, and Kun Wang. Triple trustworthiness measurement for knowledge graph. In *The World Wide Web Conference*, pages 2865–2871, 2019.

[92] Nora Kassner, Benno Krojer, and Hinrich Schütze. Are pretrained language models symbolic reasoners over knowledge? In *Proceedings of the 24th Conference on Computational Natural Language Learning*, pages 552–564, 2020.

[93] Charles Kemp, Joshua B Tenenbaum, Thomas L Griffiths, Takeshi Yamada, and Naonori Ueda. Learning systems of concepts with an infinite relational model. In *AAAI*, volume 3, page 5, 2006.

[94] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[95] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Ag-

nieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[96] Dimitris Kontokostas, Patrick Westphal, Sören Auer, Sebastian Hellmann, Jens Lehmann, Roland Cornelissen, and Amrapali Zaveri. Test-driven evaluation of linked data quality. In *Proceedings of the 23rd international conference on World Wide Web*, pages 747–758, 2014.

[97] Robert Kowalski. Predicate logic as programming language. In *IFIP congress*, volume 74, pages 569–544, 1974.

[98] Robert Kowalski and Marek Sergot. A logic-based calculus of events. *New generation computing*, 4:67–95, 1986.

[99] Fabian Kuppers, Jan Kronenberger, Amirhossein Shantia, and Anselm Haselhoff. Multivariate confidence calibration for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 326–327, 2020.

[100] Kabul Kurniawan, Andreas Ekelhart, Elmar Kiesling, Gerald Quirchmayr, and A Min Tjoa. Krystal: Knowledge graph-based framework for tactical attack discovery in audit data. *Computers & Security*, 121:102828, 2022.

[101] Patrick J Laub, Young Lee, and Thomas Taimre. *The elements of Hawkes processes*. Springer, 2021.

[102] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196. PMLR, 2014.

[103] Jos Lehmann, Michael Chan, and Alan Bundy. A higher order approach to ontology evolution in physics. *Journal on Data Semantics*, 2(4):163–187, 2013.

[104] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[105] Xue Li, Alan Bundy, and Alan Smaill. Abc repair system for datalog-like theories. In *KEOD*, pages 333–340, 2018.

[106] Xue Li, Alan Bundy, Ruiqi Zhu, Fangrong Wang, Stefano Mauceri, Lei Xu, and Jeff Z Pan. Abc in root cause analysis: Discovering missing information and repairing system failures. In *International Conference on Machine Learning, Optimization, and Data Science*, pages 346–359. Springer, 2022.

[107] Zhenyuan Li, Jun Zeng, Yan Chen, and Zhenkai Liang. Attackg: Constructing technique knowledge graph from cyber threat intelligence reports. In *Computer Security–ESORICS 2022: 27th European Symposium on Research in Computer Security, Copenhagen, Denmark, September 26–30, 2022, Proceedings, Part I*, pages 589–609. Springer, 2022.

[108] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

[109] Zhuotong Li, Yongli Zhao, Yajie Li, Sabidur Rahman, Feng Wang, Xiangjun Xin, and Jie Zhang. Fault localization based on knowledge graph in software-defined optical networks. *Journal of Lightwave Technology*, 39(13):4236–4246, 2021.

[110] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

[111] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35, 2023.

[112] Shanshan Liu, Xin Zhang, Sheng Zhang, Hui Wang, and Weiming Zhang. Neural machine reading comprehension: Methods and trends. *Applied Sciences*, 9(18):3698, 2019.

[113] Xiao-Yang Liu, Yimeng Zhang, Yukang Liao, and Ling Jiang. Dynamic updating of the knowledge base for a large-scale question answering system. *ACM Transactions on Asian and Low-Resource Language Information Processing (TALLIP)*, 19(3):1–13, 2020.

[114] John W Lloyd. *Foundations of logic programming*. Springer Science & Business Media, 2012.

[115] Yaojie Lu, Qing Liu, Dai Dai, Xinyan Xiao, Hongyu Lin, Xianpei Han, Le Sun, and Hua Wu. Unified structure generation for universal information extraction. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5755–5772, 2022.

[116] Andrea Madotto, Mahdi Namazifar, Joost Huizinga, Piero Molino, Adrien Ecoffet, Huaixiu Zheng, Dian Yu, Alexandros Papangelis, Chandra Khatri, and Gokhan Tur. Exploration based language learning for text-based games. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 1488–1494, 2021.

[117] Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60, 2014.

[118] Frank Manola, Eric Miller, Brian McBride, et al. Rdf primer. *W3C recommendation*, 10(1-107):6, 2004.

[119] J McCarthy and PJ Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.

[120] John McCarthy. Circumscription—a form of non-monotonic reasoning. *Artificial intelligence*, 13(1-2):27–39, 1980.

[121] Deborah L McGuinness, Frank Van Harmelen, et al. Owl web ontology language overview. *W3C recommendation*, 10(10):2004, 2004.

[122] Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. *Advances in Neural Information Processing Systems*, 35:17359–17372, 2022.

[123] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

[124] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[125] Rob Miller and Murray Shanahan. Some alternative formulations of the event calculus. In *Computational Logic: Logic Programming and Beyond: Essays in Honour of Robert A. Kowalski Part II*, pages 452–490. Springer, 2002.

[126] Marvin Minsky. A framework for representing knowledge, 1974.

[127] Tom Mitchell, William Cohen, Estevam Hruschka, Partha Talukdar, Bishan Yang, Justin Betteridge, Andrew Carlson, Bhavana Dalvi, Matt Gardner, Bryan Kisiel, et al. Never-ending learning. *Communications of the ACM*, 61(5):103–115, 2018.

[128] Behrang Mohit. Named entity recognition. In *Natural language processing of semitic languages*, pages 221–245. Springer, 2014.

[129] Erik T Mueller. Event calculus. *Foundations of Artificial Intelligence*, 3:671–708, 2008.

[130] Stephen Muggleton. Inductive logic programming. *New generation computing*, 8:295–318, 1991.

[131] Stephen Muggleton. Inverse entailment and progol. *New generation computing*, 13:245–286, 1995.

[132] Stephen Muggleton. Learning from positive data. In *International conference on inductive logic programming*, pages 358–376. Springer, 1996.

[133] Allan H. Murphy and Edward S. Epstein. Verification of probabilistic predictions : A brief review. *Journal of Applied Meteorology (1962-1982)*, 6(5):748–755, 1967.

[134] Stephen S Mwanje and Christian Mannweiler. Towards cognitive autonomous networks in 5g. In *2018 ITU Kaleidoscope: Machine Learning for a 5G Future (ITU K)*, pages 1–8. IEEE, 2018.

[135] Mahdi Pakdaman Naeini, Gregory Cooper, and Milos Hauskrecht. Obtaining well calibrated probabilities using bayesian binning. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[136] Zara Nasar, Syed Waqar Jaffry, and Muhammad Kamran Malik. Named entity recognition and relation extraction: State-of-the-art. *ACM Computing Surveys (CSUR)*, 54(1):1–39, 2021.

[137] Jorge Navarro-Ortiz, Pablo Romero-Diaz, Sandra Sendra, Pablo Ameigeiras, Juan J Ramos-Munoz, and Juan M Lopez-Soler. A survey on 5g usage scenarios and traffic models. *IEEE Communications Surveys & Tutorials*, 22(2):905–929, 2020.

[138] Dai Quoc Nguyen, Tu Dinh Nguyen, Dat Quoc Nguyen, and Dinh Phung. A novel embedding model for knowledge base completion based on convolutional neural network. *arXiv preprint arXiv:1712.02121*, 2017.

[139] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2015.

[140] Maximilian Nickel, Lorenzo Rosasco, and Tomaso Poggio. Holographic embeddings of knowledge graphs. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 1955–1961. AAAI Press, 2016.

[141] Alexandru Niculescu-Mizil and Rich Caruana. Predicting good probabilities with supervised learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 625–632, 2005.

[142] Prakhar Ojha and Partha Talukdar. Kgeval: Accuracy estimation of automatically constructed knowledge graphs. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1741–1750, 2017.

[143] OpenAI. Gpt-4 technical report. *ArXiv*, abs/2303.08774, 2023.

[144] Jeff Z Pan. Resource description framework. In *Handbook on ontologies*, pages 71–90. Springer, 2009.

[145] Jeff Z. Pan, Giorgos Stamou, Vassilis Tzouvaras, and Ian Horrocks. f-SWRL: A Fuzzy Extension of SWRL. In *Proc. of the International Conference on Artificial Neural Networks (ICANN 2005)*, 2005.

[146] Jeff Z Pan, Guido Vetere, Jose Manuel Gomez-Perez, and Honghan Wu. *Exploiting linked data and knowledge graphs in large organisations*. Springer, 2017.

[147] J.Z. Pan, G. Vetere, J.M. Gomez-Perez, and H. Wu, editors. *Exploiting Linked Data and Knowledge Graphs for Large Organisations*. Springer, 2017.

[148] Heiko Paulheim. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic web*, 8(3):489–508, 2017.

[149] Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. Language models as knowledge bases? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2463–2473, 2019.

[150] John Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74, 1999.

[151] Axel Polleres, Romana Pernisch, Angela Bonifati, Daniele Dell'Aglio, Daniil Dobriy, Stefania Dumbrava, Lorena Etcheverry, Nicolas Ferranti, Katja Hose, Ernesto Jiménez-Ruiz, et al. How does knowledge evolve in open knowledge graphs? *Transactions on Graph Data and Knowledge*, 1(1):11–1, 2023.

[152] Guilin Qi, Jeff Z. Pan, and Qiu Ji. A Possibilistic Extension of Description Logics. In *Proc. of 2007 International Workshop on Description Logics (DL2007)*, 2007.

[153] Guilin Qi, Jeff Z. Pan, and Qiu Ji. Extending Description Logics with Uncertainty Reasoning in Possibilistic Logic. In *the Proc. of the 9th European Conference on Symbolic and Quantitave Approaches to Reasoning with Uncertainty (ECSQARU'2007)*, pages 828–839, 2007.

[154] M Ross Quillian. *Semantic memory*. Air Force Cambridge Research Laboratories, Office of Aerospace Research . . . , 1966.

[155] M Ross Quillian. Word concepts: A theory and simulation of some basic semantic capabilities. *Behavioral science*, 12(5):410–430, 1967.

[156] J. Ross Quinlan. Learning logical definitions from relations. *Machine learning*, 5:239–266, 1990.

[157] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding with unsupervised learning. *Technical report*, 2018.

[158] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67, 2020.

[159] Amir Rahimi, Kartik Gupta, Thalaiyasingam Ajanthan, Thomas Mensink, Cristian Sminchisescu, and Richard Hartley. Post-hoc calibration of neural networks. *arXiv preprint arXiv:2006.12807*, 2, 2020.

[160] Mohammad Rashid, Marco Torchiano, Giuseppe Rizzo, Nandana Mihindukulasooriya, and Oscar Corcho. A quality assessment approach for evolving knowledge bases. *Semantic Web*, 10(2):349–383, 2019.

[161] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017.

[162] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, 2019.

[163] R REITHER. On closed world data bases. *Logic and Data Bases*, pages 55–76, 1978.

[164] Yuan Ren, Artemis Parvizi, Chris Mellish, Jeff Z Pan, Kees Van Deemter, and Robert Stevens. Towards competency question-driven ontology authoring. In *The Semantic Web: Trends and Challenges: 11th International Conference, ESWC 2014, Anissaras, Crete, Greece, May 25-29, 2014. Proceedings 11*, pages 752–767. Springer, 2014.

[165] Mark Bishop Ring. *Continual learning in reinforcement environments*. PhD thesis, University of Texas at Austin, 1994.

[166] Adam Roberts, Colin Raffel, and Noam Shazeer. How much knowledge can you pack into the parameters of a language model? In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5418–5426, 2020.

[167] Andrea Rossi, Denilson Barbosa, Donatella Firmani, Antonio Matinata, and Paolo Merialdo. Knowledge graph embedding for link prediction: A comparative analysis. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 15(2):1–49, 2021.

[168] SAE. Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles. SAE Standard, 2018. SAE J 3016-2018.

[169] Mohammad Javad Saeedizade, Najmeh Torabian, and Behrouz Minaei-Bidgoli. Kgrefiner: Knowledge graph refinement for improving accuracy of translational link prediction methods. *arXiv preprint arXiv:2106.14233*, 2021.

[170] M Series. Imt vision–framework and overall objectives of the future development of imt for 2020 and beyond. *Recommendation ITU*, 2083(0), 2015.

[171] Wei Shen, Jianyong Wang, and Jiawei Han. Entity linking with a knowledge base: Issues, techniques, and solutions. *IEEE Transactions on Knowledge and Data Engineering*, 27(2):443–460, 2014.

[172] Baoxu Shi and Tim Weninger. Open-world knowledge graph completion. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18. AAAI Press, 2018.

[173] Kumar Shridhar, Alessandro Stolfo, and Mrinmaya Sachan. Distilling reasoning capabilities into smaller language models. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 7059–7073, 2023.

[174] Abraham Silberschatz, Henry Korth, and Shashank Sudarshan. *Database systems concepts*. McGraw-Hill, Inc., 2005.

[175] Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. Reasoning with neural tensor networks for knowledge base completion. *Advances in neural information processing systems*, 26, 2013.

[176] Jacob Solawetz and Stefan Larson. Lsoie: A large-scale dataset for supervised open information extraction. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 2595–2600, 2021.

[177] Jacopo Soldani and Antonio Brogi. Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey. *ACM Computing Surveys (CSUR)*, 55(3):1–39, 2022.

[178] Gabriel Stanovsky and Ido Dagan. Creating a large benchmark for open information extraction. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2300–2305, 2016.

[179] Giorgos Stoilos, Giorgos B. Stamou, and Jeff Z. Pan. Handling imprecise knowledge with fuzzy description logic. In *Proceedings of the 2006 International Workshop on Description Logics (DL2006)*, 2006.

[180] Ljiljana Stojanovic, Alexander Maedche, Boris Motik, and Nenad Stojanovic. User-driven ontology evolution management. In *International Conference on Knowledge Engineering and Knowledge Management*, pages 285–300. Springer, 2002.

[181] Yicheng Sui, Yuzhe Zhang, Jianjun Sun, Ting Xu, Shenglin Zhang, Zhengdan Li, Yongqian Sun, Fangrui Guo, Junyu Shen, Yuzhi Zhang, et al. Logkg: Log failure diagnosis through knowledge graph. *IEEE Transactions on Services Computing*, 2023.

[182] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. *arXiv preprint arXiv:1902.10197*, 2019.

[183] Pedro Tabacof and Luca Costabello. Probability calibration for knowledge graph embedding models. In *International Conference on Learning Representations*, 2020.

[184] Paul Thagard. Frames, knowledge, and inference. *Synthese*, 61:233–259, 1984.

[185] Sebastian Thrun and Tom M Mitchell. Lifelong robot learning. *Robotics and autonomous systems*, 15(1-2):25–46, 1995.

[186] Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, pages 57–66, Beijing, China, July 2015. Association for Computational Linguistics.

[187] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *International conference on machine learning*, pages 2071–2080. PMLR, 2016.

[188] Efthimis Tsilionis, Alexander Artikis, and Georgios Paliouras. Incremental event calculus for run-time reasoning. *Journal of Artificial Intelligence Research*, 73:967–1023, 2022.

[189] Risto Vaarandi. A data clustering algorithm for mining patterns from event logs. In *Proceedings of the 3rd IEEE Workshop on IP Operations & Management (IPOM 2003)(IEEE Cat. No. 03EX764)*, pages 119–126. Ieee, 2003.

[190] Risto Vaarandi and Mauno Pihelgas. Logcluster-a data clustering and pattern mining algorithm for event logs. In *2015 11th International conference on network and service management (CNSM)*, pages 1–7. IEEE, 2015.

[191] Bas C Van Fraassen. Calibration: A frequency justification for personal probability. In *Physics, philosophy and psychoanalysis*, pages 295–319. Springer, 1983.

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

[193] Boris Villazon-Terrazas, Nuria Garcia-Santa, Yuan Ren, Alessandro Faraotti, Honghan Wu, Yuting Zhao, Guido Vetere, and Jeff Z Pan. Knowledge graph foundations. In *Exploiting Linked Data and Knowledge Graphs in Large Organisations*, pages 17–55. Springer, 2017.

[194] Susan Vineberg. Dutch Book Arguments. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Spring 2016 edition, 2016.

[195] Somin Wadhwa, Silvio Amir, and Byron C Wallace. Revisiting relation extraction in the era of large language models. In *Proceedings of the conference.*

*Association for Computational Linguistics. Meeting*, volume 2023, page 15566. NIH Public Access, 2023.

[196] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, 2018.

[197] Fangrong Wang, Alan Bundy, Xue Li, Ruiqi Zhu, Kwabena Nuamah, Lei Xu, Stefano Mauceri, and Jeff Z. Pan. LEKG: A system for constructing knowledge graphs from log extraction. In *The 10th International Joint Conference on Knowledge Graphs*, IJCKG'21, page 181–185, New York, NY, USA, 2021. Association for Computing Machinery.

[198] Hailin Wang, Ke Qin, Rufai Yusuf Zakari, Guoming Lu, and Jin Yin. Deep neural network-based relation extraction: an overview. *Neural Computing and Applications*, pages 1–21, 2022.

[199] Liang Wang, Wei Zhao, Zhuoyu Wei, and Jingming Liu. Simkgc: Simple contrastive knowledge graph completion with pre-trained language models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4281–4294, 2022.

[200] Meihong Wang, Linling Qiu, and Xiaoli Wang. A survey on knowledge graph embeddings for link prediction. *Symmetry*, 13(3):485, 2021.

[201] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743, 2017.

[202] Song Wang, Yaochen Zhu, Haochen Liu, Zaiyi Zheng, Chen Chen, et al. Knowledge editing for large language models: A survey. *arXiv preprint arXiv:2310.16218*, 2023.

[203] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, AAAI'14, page 1112–1119. AAAI Press, 2014.

[204] David S Warren. Introduction to prolog. In *Prolog: The Next 50 Years*, pages 3–19. Springer, 2023.

[205] Peter West, Chandra Bhagavatula, Jack Hessel, Jena Hwang, Liwei Jiang, Ronan Le Bras, Ximing Lu, Sean Welleck, and Yejin Choi. Symbolic knowledge distillation: from general language models to commonsense models. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4602–4625, 2022.

[206] Kemas Wiharja, Jeff Z Pan, Martin J Kollingbaum, and Yu Deng. Schema aware iterative knowledge graph completion. *Journal of Web Semantics*, 65:100616, 2020.

[207] W Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. A survey on software fault localization. *IEEE Transactions on Software Engineering*, 42(8):707–740, 2016.

[208] William A Woods. What's in a link: Foundations for semantic networks. In *Representation and understanding*, pages 35–82. Elsevier, 1975.

[209] Wentao Wu, Hongsong Li, Haixun Wang, and Kenny Q Zhu. Probase: A probabilistic taxonomy for text understanding. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 481–492, 2012.

[210] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[211] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*, 2014.

[212] Yunzhi Yao, Peng Wang, Bozhong Tian, Siyuan Cheng, Zhoubo Li, Shumin Deng, Huajun Chen, and Ningyu Zhang. Editing large language models: Problems, methods, and opportunities. *arXiv preprint arXiv:2305.13172*, 2023.

[213] Amrapali Zaveri, Anisa Rula, Andrea Maurino, Ricardo Pietrobon, Jens Lehmann, and Soeren Auer. Quality assessment for linked data: A survey. *Semantic Web*, 7(1):63–93, 2016.

[214] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pages 3987–3995. PMLR, 2017.

[215] Keli Zhang, Marcus Kalander, Min Zhou, Xi Zhang, and Junjian Ye. An influence-based approach for root cause alarm discovery in telecom networks. In *International Conference on Service-Oriented Computing*, pages 124–136. Springer, 2020.

[216] Zihan Zhang, Meng Fang, Ling Chen, Mohammad-Reza Namazi-Rad, and Jun Wang. How do large language models capture the ever-changing world knowledge? a review of recent advances. *arXiv preprint arXiv:2310.07343*, 2023.

[217] Shengjia Zhao, Michael Kim, Roshni Sahoo, Tengyu Ma, and Stefano Ermon. Calibrating predictions to decisions: A novel approach to multi-class calibration. *Advances in Neural Information Processing Systems*, 34:22313–22324, 2021.

[218] Zhehui Zhou, Can Wang, Yan Feng, and Defang Chen. Jointe: Jointly utilizing 1d and 2d convolution for knowledge graph embedding. *Knowledge-Based Systems*, 240:108100, 2022.

[219] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R Lyu. Tools and benchmarks for automated log parsing. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 121–130. IEEE, 2019.

[220] Ricky Zhu, Alan Bundy, Fangrong Wang, Xue Li, Kuwabena Nuamah, Lei Xu, Stefano Mauceri, and Jeff Z Pan. Assessing the quality of a knowledge graph via link prediction tasks. In *7th International Conference on Natural Language Processing and Information Retrieval*, pages 1–10. Association for Computing Machinery, 2023.

[221] Ruiqi Zhu, Fangrong Wang, Alan Bundy, Xue Li, Kwabena Nuamah, Lei Xu, Stefano Mauceri, and Jeff Z Pan. A closer look at probability calibration of

knowledge graph embedding. In *Proceedings of the 11th International Joint Conference on Knowledge Graphs*, pages 104–109, 2022.