Empirical Software Engineering (2022) 27: 93
https://doi.org/10.1007/s10664-022-10159-w

SoftNER: Mining knowledge graphs from cloud incidents

Manish Shetty[1] · Chetan Bansal[2] · Sumit Kumar[3] · Nikitha Rao[1] · Nachiappan Nagappan[2]

Accepted: 17 March 2022 / Published online: 28 April 2022

Abstract

The move from boxed products to services and the widespread adoption of cloud computing has had a huge impact on the software development life cycle and DevOps processes. Particularly, incident management has become critical for developing and operating large-scale services. Prior work on incident management has heavily focused on the challenges with incident triaging and de-duplication. In this work, we address the fundamental problem of structured knowledge extraction from service incidents. We have built SoftNER, a framework for mining Knowledge Graphs from incident reports. First, we build a novel multi-task learning based BiLSTM-CRF model which leverages not just the semantic context but also the data-types for extracting factual information in the form of named entities. Next, we present an approach to mine relations between the named entities for automatically constructing knowledge graphs. We have deployed SoftNER at Microsoft, a major cloud service provider and have evaluated it on more than 2 months of cloud incidents. We show that SoftNER's unsupervised pipeline learns the software entity types from unstructured incident data with high precision of 0.96 (at rank 50) and 0.77 (at rank 100). We also evaluate and show that SoftNER's unsupervised pipeline accurately labels data with a precision of 0.94. Further, our multi-task learning based deep learning model also outperforms the state-of-the-art NER models with an average F1 of 0.96. Lastly, using the knowledge extracted by SoftNER, we are able to build accurate models for tasks such as incident triaging and recommending entities based on their relevance to incident titles.

Keywords Cloud services · Service incidents · Knowledge extraction · Knowledge graphs · Deep learning · Machine learning

Communicated by: Sigrid Eldh, Davide Falessi and Burak Turhan

This article belongs to the Topical Collection: *Software Engineering in Practice*

This work was done when Nachiappan was with Microsoft Research. He is currently with Facebook.

✉ Manish Shetty
manish.shetty.m@outlook.com

Extended author information available on the last page of the article.

1 Introduction

In the last decade, two major paradigm shifts have revolutionized the software industry. First is the move from boxed software products to *services*. Large software organizations like Adobe and Microsoft[1] which pre-date the internet revolution have been aggressively trying to move from selling boxed products to subscription based services. This has primarily been driven by the benefits of subscription based services such as scalability, faster releases, insights from telemetry, and, better revenue stability. The second major shift has been the widespread adoption of *public clouds*. More and more software companies are moving from on-premises data centers to public clouds like Amazon AWS, Google Cloud, Microsoft Azure, etc. Gartner has forecasted[2] the public cloud market to grow to about $266 billion in revenue in 2020, out of which about 43% revenue will be from the Software as a Service (SaaS) segment. The cloud revolution has enabled companies like Netflix, Uber, etc. to build internet scale products without having to provision their own infrastructure.

These paradigm shifts have also had a transformational effect on the way software is developed, deployed, and maintained. For instance, software engineers no longer develop monolithic software. They build services that have dependencies on several 1^{st} party and 3^{rd} party services and APIs. Typically, any web application will leverage cloud services for basic building blocks like storage (relational and blob), compute, and authentication. These complex dependencies introduce a bottleneck where a single failure can have a cascading effect. In 2017, a small typo led to a major outage in the AWS S3 storage service[3], which ended up costing over $150 million to customers like Slack, Medium, etc. Microsoft had a glitch in their Active Directory in October 2019[4], which locked out customers from accessing their Office 365 and Azure accounts. These outages are inevitable and can be caused by various factors such as software bugs, misconfigurations (Mehta et al. 2020), or even environmental factors.

To keep up with these changes, DevOps processes and platforms have also evolved over time (Dang et al. 2019; Kumar et al. 2019). Most software companies these days have incident management and on-call duty as a part of the DevOps workflow, where the key motivation is to reduce impact on customers by mitigating any issue as soon as possible. We discuss the incident life-cycle and some of the associated challenges in detail in Section 2. Prior work on incident management has largely focused on two challenges: incident triaging (Chen et al. 2019a, b) and diagnosis (Bansal et al. 2020; Luo et al. 2014). Here, triaging refers to the process of identifying and routing the incident to the appropriate team for resolution. Chen et al. (2019b) did an empirical study where they found that up to 60% of incidents can be mis-triaged. They proposed DeepCT, a deep learning approach for automated incident triaging using incident data (title, summary, comments) and environmental factors.

Based on our experience from operating web-scale cloud services at Microsoft and discussions with various product teams, we observe that a key challenge in incident management lies in the lack of structured representations of these incidents. These incidents could be created by a wide variety of sources such as customers, engineers, and even automated

[1] https://www.pcworld.com/article/2038194/microsoft-says-its-boxed-software-probably-will-be-gone-within-a-decade.html

[2] https://www.gartner.com/en/newsroom/press-releases/2019-11-13-gartner-forecasts-worldwide-public-cloud-revenue-to-grow-17-percent-in-2020

[3] https://www.wsj.com/articles/amazon-finds-the-cause-of-its-aws-outage-a-typo-1488490506

[4] https://www.theregister.co.uk/2019/10/18/microsoft_azure_mfa/

monitoring systems. They are mostly unstructured and contain several types of information like incident metadata, description, stack traces, outputs of shell scripts, images, etc. As a result, on-call engineers spend a considerable amount of effort manually parsing verbose incident descriptions to understand the issue, locating key information for mitigation, and finally engaging the appropriate team to acknowledge and fix the issue. The time and effort spent here is termed as Time-To-Engage (TTE) and adds a significant delay to the ensuing tasks in the incident life-cycle.

Thus, in this work, we address the key problem of **extracting structured knowledge from service incidents**. This structured knowledge would reduce the effort spent by on-call engineers by opening up avenues for automating processes like log extraction and health checks on resources (VMs, Databases, etc.) identified within these descriptions. The extracted knowledge would also help build better models for performing downstream tasks like triaging and root-cause analysis.

Ideally, any knowledge extraction framework should have the following qualities:

1. It should be **unsupervised** because it is laborious and expensive to annotate a large amount of training data. This is important since a service's unique vocabulary is unknown.
2. It should be **domain agnostic** so that it can scale to a high volume and a large number of information entity types. Unlike in the web domain, where there is a small set of key entities such as people, places, and organizations, for incidents, we don't know these entities apriori.
3. It should be **extensible** so that we can adapt the bootstrapping techniques to incidents from other services, or even other data sets such as bug reports. This is critical because each service (e.g. compute, networking, storage) could have its unique vocabulary and terminology.

We have designed **Soft**ware artifact K**N**owledge **E**xt**R**action (SoftNER), a framework for unsupervised knowledge extraction from service incidents, which has these three qualities: unsupervised, domain agnostic, and extensible. As shown in Fig. 1, we break down knowledge extraction into three steps. First, we use Named-entity recognition (NER) for extraction of factual and structured information from the incidents. We leverage syntactic pattern extractors for bootstrapping the training data. Further, we incorporate a novel multi-task BiLSTM-CRF deep learning model with an attention mechanism. Next, we enrich these entities by mining binary relations between the entities. Lastly, we automatically construct knowledge graphs using the entities and relations extracted. We have evaluated and deployed SoftNER at Microsoft, a major cloud service provider. We show that our unsupervised knowledge graph mining has high precision. Our multi-task deep learning model also outperforms existing state-of-the-art models on the NER task. Lastly, using the extracted knowledge, we show that we can build more accurate models for key downstream tasks like incident triaging and also improve tooling in incident management platforms.

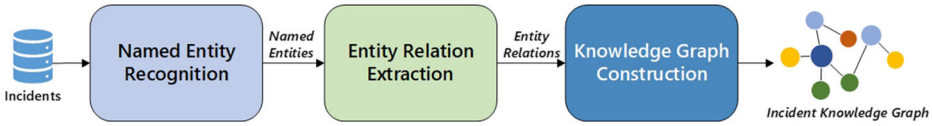

Fig. 1 SoftNER Overview

In this work, we make the following main contributions:

1. We propose SoftNER, the first approach for completely unsupervised Knowledge Graph mining from service incidents.
2. We build a novel multi-task learning based deep learning model for named-entity recognition which leverages not just the semantic features but also the data-types. Our evaluation shows that it outperforms the existing state-of-the-art NER models.
3. We propose a novel approach to mine relations between the extracted entities for construction of knowledge graphs.
4. We do an extensive evaluation of SoftNER on over 2 months of cloud service incidents from Microsoft.[5]
5. Lastly, we have deployed SoftNER in production at Microsoft where it has been used for knowledge extraction from incidents for over 6 months.

The rest of the paper is organized as follows: In Section 2, we discuss insights from the incident management processes at Microsoft. Section 3 discusses the related work and Section 4 provides an overview of the SoftNER framework. Sections 5 and 6 provide details of our approach to knowledge extraction using named-entity recognition and knowledge graph construction respectively. Section 7 describes the implementation and deployment details. In Section 8, we discuss the experimental evaluation of our approach. Section 9 describes two applications of SoftNER in detail and we address potential threats to validity in Section 10. Lastly, we the discuss the generalizability of SoftNER and future work in Section 11. We conclude the paper in Section 12.

This paper extends our prior publication (Shetty et al. 2021) presented at the 43^{rd} International Conference on Software Engineering: Software Engineering in Practice. New materials with respect to the conference version include:

1. An extension to the SoftNER framework, where we propose and evaluate an unsupervised approach to further extract entity relations and construct knowledge graphs using co-occurrence information (Section 6).
2. An application of the constructed knowledge graph by recommending required entities for incidents based on incident titles (Section 9.2). Here, we use a combination of clustering and a novel path based scoring technique to identify entity-incident relevance.
3. An evaluation of the quality of labeled data created by the unsupervised data labeling component of SoftNER (Section 8.3).
4. Additional details regarding the integration of SoftNER with the incident management platform at Microsoft (Section 7).

2 Incident Life-Cycle

In Microsoft, an incident is defined as an unplanned interruption or degradation of a product or service that is causing customer impact. For example, a slow connection, a timeout, a crash, etc. could constitute an incident. Here, we detail the incident management process which defines the various steps an incident goes through - from creation to closing. Bugs and incidents are very different, as in our case incidents may or may not lead to bugs. Furthermore, incidents often require the involvement of on-call developers who have

[5]We cannot disclose the number of incidents due to Microsoft Policy.

been designated to respond to incidents. Figure 2 presents a high level view of the incident management process. Most online services have their own specific incident management protocol. This figure is a generic process that could apply outside of Microsoft as well.

The incident management process is broadly classified into four phases. In the first phase - *alerting* phase, typically an alert is fired when the service monitoring metrics fall below a pre-defined acceptance level in terms of performance (e.g. slow response, slow transfer rate, system hang or crash, etc.). This leads to phase two - the *engagement* phase. In this phase, an incident is first created in the incident database. It is then escalated to a "related" team. The identification of the first "related" team is automatic, based on heuristics or component ownership. The team investigates the incidents and engages with relevant stakeholders or re-routes it to a more appropriate team to repeat the steps. This process is called *Incident Triage*, where the appropriate team is identified to take up the resolution of this incident. Following triage, in the *investigation* phase, the appropriate team identifies the problem cause and moves over to mitigation and root cause analysis. Then, identified bugs, if any, are filed for engineering teams to fix. In the final phase of *resolution*, the incident is resolved and bugs are fixed in the system. Our work applies directly to the engagement and investigation phases, dealing with unsupervised structured knowledge extraction from service incident descriptions.

3 Related Work

SoftNER is inspired from prior work in two main domains: (1) software engineering and (2) information retrieval. In software engineering, incident management has recently become an active area of research. Significant work has been done on specific aspects of incident management, such as automated triaging of incidents, incident diagnosis, and detection. Our work is complementary to these efforts since we focus on the fundamental problem of structured knowledge extraction from incidents. Also, in information retrieval, there have been decades of research on knowledge extraction from web data. However, majority of the research in the web space has focused on supervised or semi-supervised entity extraction, limited to a known set of entities such as people, organizations and places. Additionally, in software artifacts like incidents, the vocabulary is not limited to natural languages and has other entities such as GUIDs, Exceptions, IP Addresses, etc. Hence, SoftNER leverages an unsupervised framework along with a novel data-type aware deep learning model for knowledge extraction. In this section, we further discuss related work in these areas:

Incident management. Recent work on incident management has been focused on the problem of triaging incidents to correct teams. As per the empirical study by Chen et al. (2019a), mistriaging of incidents happen quite frequently and can lead up to a delay of over 10X in triaging time, apart from the lost revenue and customer impact. To solve this problem, they have also proposed DeepCT (Chen et al. 2019b), a deep learning

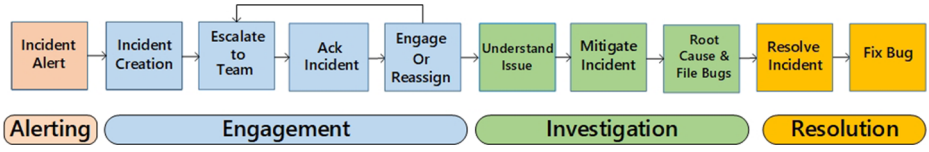

Fig. 2 Incident life-cycle

based method for routing of incidents to the correct teams. They evaluate the model on several large-scale services and are able to achieve a high mean accuracy of up to 0.7. There has also been a significant amount of work done on diagnosing and root causing of service incidents. Nair et al. (2015) uses hierarchical detectors based on time-series anomaly detection for diagnosing incidents in services. DeCaf (Bansal et al. 2020) uses random forest models for automatically detecting and categorizing performance issues in large-scale cloud services. It also categorizes the detected issues into the buckets of new, known, regressed, resolved issues based on trend analysis. Systems such as AirAlert (Chen et al. 2019) have been built using machine learning to predict critical service incidents, called outages, in large scale services. It correlates signals and dependencies from across the cloud system and uses machine learning for predicting outages. Different from existing work, we focus on the fundamental problem of structured knowledge extraction from incidents. Then, we show that using the knowledge extracted by SoftNER, we can build significantly more accurate models for these incident management tasks (Section 9).

Bug reports. Significant amount of research has been done on bug reports in the traditional software context. SoftNER is inspired by InfoZilla (Bettenburg et al. 2008) which leverages heuristics and regular expressions for extracting four elements from Eclipse bug reports: patches, stack traces, code, and lists. Unlike InfoZilla, we build a completely unsupervised deep learning based framework that enables extracting hundreds of entities without requiring any prior knowledge about them. Our work also targets incidents, which are more complex than bugs because of numerous layers of dependencies and real-time mitigation requirements. Bug triage has been an active area of research (Anvik et al. 2006; Tian et al. 2016; Bortis and Van Der Hoek 2013; Wang et al. 2014) in the software engineering community. Other aspects of bug reports such as bug classification (Zhou et al. 2016) and bug fix prediction time (Ardimento and Dinapoli 2017) have also been explored. Similar to incidents, existing work on bug reports have largely used the unstructured attributes like bug description as it is. Though we have focused on incidents, SoftNER can be applied to bug reports for extracting structured information and building models for tasks like bug triaging, classification, etc.

Information retrieval. Knowledge and entity extraction has been studied in depth by the information retrieval community. Search engines like Google and Bing rely heavily on entity knowledge bases for tasks like intent understanding (Pantel et al. 2012) and query reformulation (Xu et al. 2008). *Named-Entity Recognition* (NER) (Nadeau and Sekine 2007), specifically, is a well explored task of parsing unstructured text to detect entities and classify them into specific categories. Prior work on NER has explored new domains, such as social media platforms (Finin et al. 2010; Ritter et al. 2011), (Limsopatham and Collier 2016; Aguilar et al. 2019) and biomedical text (Kulkarni et al. 2018; Greenberg et al. 2018). Several of these use supervised methods for NER that require a large amount of training data, which can be cost prohibitive to collect. Hence, search engines commonly use semi-supervised methods which leverage a small seed set to bootstrap the entity extraction process. For instance, the expert editors would seed the entity list for a particular entity type, let's say fruits with some initial values such as {apple, mango, orange}. Then using pattern or distributional extractors, the list would be expanded to cover the entire list of fruits. In this work, our goal was to build a fully unsupervised system where we don't need any pre-existing list of entity types or seed values. This is primarily because every service and organization is unique and manually bootstrapping SoftNER would be very laborious.

While NER is well explored in the web space, in software engineering, however, there has been relatively less work on NER. Ye et al. (2016) annotated 5K sentences from `StackOverflow` with a limited set of five named entity types (Programming Language, Platform, API, Tool-Library-Framework and Software Standard). The authors then used a traditional feature-based CRF model to recognize these entities. In contrast, with SoftNER we aim to perform NER and knowledge extraction at a much larger scale (tens of thousands of incidents with 50-100 entity types that are unknown apriori) using an unsupervised approach. Additionally, incident reports, unlike web data, contain not just natural language tokens but also other entities such as GUIDs and IP Addresses. Hence, SoftNER leverages a novel data-type aware multi-task model for knowledge extraction, specifically designed for the software domain.

Lastly, in information retrieval and natural language processing, NER is commonly followed by *Relation Extraction* (Pawar et al. 2017). A *relation* usually denotes a relationship between two or more named entities, and the task of relation extraction (RE) consists of extracting mentions of relations of interest in each sentence. Similar to NER, several statistical (McDonald et al. 2005; Zelenko et al. 2003), machine learning, as well as deep neural models (Hashimoto et al. 2015; Hendrickx et al. 2019) have been proposed for relation extraction. In this work, we follow a similar approach to Zelenko et al. (2003) and McDonald et al. (2005) to use co-occurrence as a signal for entity relations. In contrast to these approaches, we further use a co-occurrence based metric (NPMI) to score extracted relations and filter noisy candidates.

4 SoftNER Overview

Incident management is key to running large scale services in an efficient way. However, there is a lot of scope for optimization which can inturn increase customer satisfaction, reduce on-call fatigue, and provide revenue savings. Existing work on incident management has primarily focused on incident triaging. The state-of-the-art incident triaging methods (Chen et al. 2019b) use novel deep learning methods which take raw unstructured incident descriptions as input. In this work, we focus on the fundamental problem of structured knowledge extraction from these unstructured incidents. With the structured information, we can save time and effort for on-call engineers by automating manual processes such as running automated health checks on identified resources as described in an example at the end of this section. At the same time, with this structured representation, we can build simpler yet better machine learning models for tasks like incident triaging.

To solve the challenges with incident management, we have designed the SoftNER framework. It is the first automated approach for structured knowledge extraction from service incidents. We frame the initial structured knowledge extraction problem as a *Named-Entity Recognition* (NER) task, which has been well explored in the Information Retrieval (IR) domain (Nadeau and Sekine 2007; Lample et al. 2016). Named-Entity Recognition is defined as the task of parsing unstructured text to not only detect entities but also classify them into specific categories. An entity can be any chunk of text which belongs to a given type or category. As an example, here is the input and output of a NER task for a news headline:

Input: Over 320 million people have visited the Disneyland in Paris since it opened in 1992.

Output: Over **[320 million $_{\text{COUNT}}$]** people have visited the **[Disneyland $_{\text{ORG}}$]** in **[Paris $_{\text{LOC}}$]** since it opened in **[1992 $_{\text{YEAR}}$]**.

Framing the knowledge extraction problem as a NER task enables us to not only extract factual information from the incidents but also classify them as specific entities. For instance, if we just extract a *GUID* from the text, it provides limited context. However, identifying that *GUID* as a *Resource Id* is much more useful to on-call engineers, who can then identify affected resources, or to other models that perform tasks like triaging. This makes it more suitable for the incident management scenario when compared to other solutions like text summarization.

One key limitation of any supervised machine learning pipeline is the requirement of huge amounts of labeled data which can be cost prohibitive to manually generate. In service incidents, the lack of existing training data prevents us from using any supervised or semi-supervised techniques. SoftNER uses pattern extractors which leverage the *key-value* and *tabular* structural patterns in the incident descriptions to bootstrap the training data. We then use label propagation to generalize the training data beyond these patterns. We also incorporate a novel multi-Task deep learning model that is able to extract named-entities from the unstructured incident descriptions with very high accuracy. The model not only leverages the semantic features but also the data type of the individual tokens (such as GUID, URI, Boolean, Numerical, IP Address, etc.).

Example: Let's consider a real incident reported by a customer of the Cloud Networking service operated by Microsoft. The incident was caused due to an error in deleting a Virtual Network resource. The key information required to triage and mitigate this incident is actually the '*Problem Type*' and the '*Virtual Network Id*'. This information is already present in an unstructured form within the incident description and the challenge is to extract it automatically in a structured format. Using SoftNER, we can not only provide this key information to the on-call engineers but also automate some of the manual tasks such as running health checks. For instance, in this example, we can automatically look up the current status/logs of the resource using the identifier (*Virtual Network Id*) before the on-call engineer engages.

Service incidents can be created by external customers or even automated monitoring systems. They contain unstructured information in various forms, like statements, conversations, stack traces, etc. As stated before, this makes incident descriptions rich in information that are identifiable as entities. Although extracting all entities is useful, certain entities are more important for the investigation and mitigation of an incident. To capture complete knowledge that can be used for other aspects, such as entity relevance, it is important to mine interactions and relations between entities. For instance, it would be quite useful to map a given *Identifier* and *IP Address* to the same *Virtual Machine* resource. Another example, would be to identify *Source* and *Destination* entity relations in an incident dealing with physical networking devices. With these insights, SoftNER uses an unsupervised approach to extract entity relations using co-occurring entity pairs and pointwise mutual information. Having identified related entities, SoftNER automatically constructs an undirected incident knowledge graph. As shown in Fig. 3, the knowledge graph nodes represent cloud services, incidents, and entities extracted from incidents, and edges represent relatedness. The knowledge graph captures information that can be queried like traditional databases, analyzed like graph data structures, and allows inference of new knowledge.

Lastly, the structured knowledge extracted by SoftNER opens a wide range of applications. For instance, the extracted entities can be used as features to build more accurate

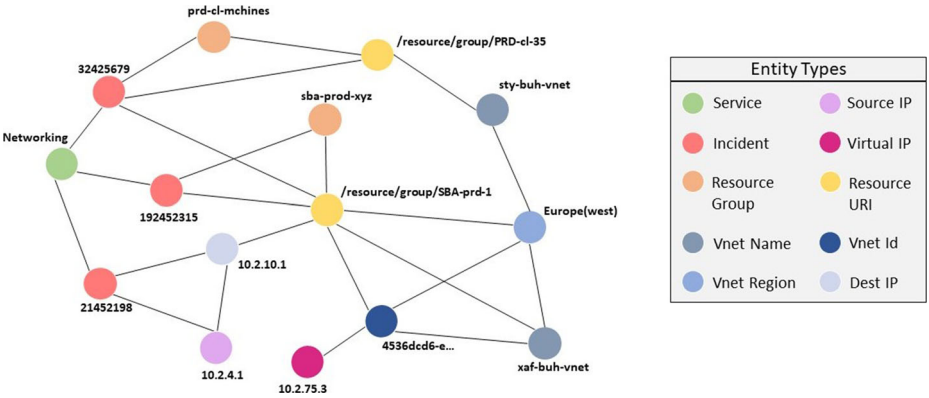

Fig. 3 Incident knowledge graph

learning models for predictive tasks like triaging, root causing, etc. The knowledge graph can also be utilized to improve tooling by intelligently recommending expected entities based on their relevance to the issue described in the incident. Below we list some of the terms which we will be using throughout the rest of the paper. These terms identify different aspects of named-entities.

- **Entity Name:** N-gram indicating the name of the entity. In the current implementation, N can range from 1 to 3.
- **Entity Value:** The value of the named-entity for a given instance.
- **Data Type:** The data type of the values for the named-entity.
- **Entity Relation:** A relationship between 2 or more entities.

5 Named Entity Recognition

Here, we describe our approach in implementing SoftNER's named-entity recognition pipeline in detail. As shown in Fig. 4, we start with the data cleaning process, followed by unsupervised data labeling. Then we describe the label propagation process and the architecture of the deep learning model.

5.1 Data Cleaning

Service incident descriptions and summaries are created by various sources such as external customers, feature engineers and even automated monitoring systems. The information

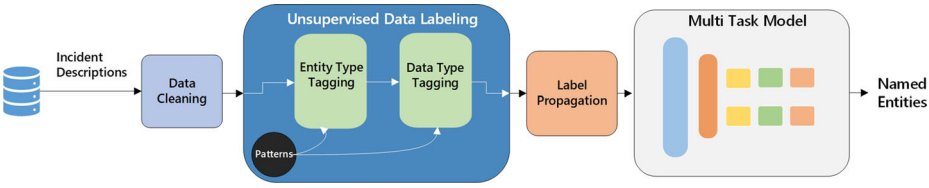

Fig. 4 Named-entity recognition pipeline

could be in various forms, like textual statements, conversations, stack traces, shell scripts, images, etc., all of which make the information unstructured and hard to interpret. Subsequently, we first prune tables in the incident description that have more than two columns and get rid of HTML tags using regexes and parsers. To do so, we use standard python libraries such as `BeautifulSoup` and `html` to parse HTML data. In this process, we also segment the information into sentences using newline characters. Next, we process individual sentences by cleaning up extra spaces and tokenize them into words. Our tokenization technique is custom implemented to handle camel-case tokens and URLs as well.

5.2 Unsupervised Data Labelling

A major challenge in our case, was the lack of a pre-existing labelled data set which can be used for a supervised NER task. It would also be very expensive to manually label data since entity types are unknown and also vary across different services. Thus, SoftNER uses an unsupervised framework to create a labeled corpus. Here are the steps followed to automatically generate a labeled corpus for named-entity extraction:

5.2.1 Entity Type Tagging

As mentioned above, we neither have a pre-existing labeled data set nor a predefined fixed set of entity types. Thus, in this phase, we first identify a candidate entity set. We then clean that set and eliminate noisy entities. This final set of entities is used to tag the incident data set.

Step 1A (Candidate Identification). Since we don't have a list of entity types apriori, we first bootstrap the framework with a candidate set of entity name and value pairs that are identified without manual effort. For this, we have built pattern extractors using some structural patterns commonly found in the incident descriptions:

- **Key-Value pairs** - This pattern is commonly used in the incident descriptions to specify various entities where the Entity Type (key) and Value are joined by a separator like ':'. For instance, "*Status code: 401*" or "*Problem type: VM not found*". We split the sentence on the separator and extract the first half as the *Entity Type* and the second half as the *Entity Value*.
- **Tables** - Tables also occur quite frequently in incident descriptions, especially the ones which are created by bots or monitoring services. In a two-column html table, we extract the first column values as *Entity Types* and the second as *Entity Values*.

The above commonly occurring patterns are used to extract a candidate set of entity-value pairs after parsing the data set. Note that the above listed patterns do not constitute an exhaustive set. More patterns can be used to generate labeled data using methods like weak supervision (Ratner et al. 2017; Rao et al. 2020).

Step 1B (Candidate Elimination). Now, we have a candidate set of entity names and values. However, the candidate set is noisy since we have extracted all text which satisfies these patterns. Thus, we filter out candidate entity names that contain symbols or numbers, as they are noisy labels. We further extract n-grams (n: 1 to 3) from the candidate entity names and take the top K most frequently occurring n-grams. Here, K is a parameter that can be chosen by the user to retrieve a fixed number of entity types. In this process, less frequent and thus noisy candidate entity types, such as *"token acquisition*

started" and *"database connected"*, are pruned. We manually analyze the effect of K, in Section 8.2, using a precision-vs-rank plot. Furthermore, with this n-gram approach, entity-value variations such as [*"My Subscription Id", "6572"*] and [*"The Subscription Id", "6572"*] would be transformed to [*"Subscription Id", "6572"*] since *"Subscription Id"* is a commonly occurring bi-gram in the candidate set. After performing the above steps, a final set of entity types, represented as n-grams, is determined. With this set, every occurrence of these n-grams, in the context of the 2 chosen patterns - key-value pairs and tables - are tagged in a single pass over the data set. Please refer to Table 1 for examples of entities extracted using the unsupervised approach.

5.2.2 Data-Type Tagging

For the refined candidate set, we next infer the data type of the entity values using in-built Python functions such as "isnumeric" along with custom regexes. This step, in addition to the entity-type tagging, is leveraged in SoftNER's multi-task learning model, where we jointly train to predict both the entity type and the data type. These tasks are complementary and help improve the accuracy for each of the individual prediction tasks. Based on discussions with the service engineers, we have defined the following data types:

1. **Basic Types**: `Numeric`, `Boolean`, `Alphabetical`, `Alphanumeric`, `Non-Alphanumeric`
2. **Complex Types**: `GUID`, `URI`, `IP Address`
3. **Other**

To infer the data type for a given entity, we compute it for each occurrence of a named entity in the data set independently. To identify each data type, we make use of either simple regular expressions (e.g., $d\{1, 3\}.d\{1, 3\}.d\{1, 3\}.d\{1, 3\}$ for `IP Address`) or inbuilt library functions in python (e.g., `isAlpha`, `isAlnum`). For each entity value, following a priority order, we run these expressions/functions against the value and pick the first match as the data type. In case no expression matches, we fall back to `Other`. The priority order we use is: `IP Address` > `Numeric` > `Boolean` > `Alphabetical` > `URI` > `GUID` > `AlphaNumeric` > `Non-AlphaNumeric` > `Other`.

Table 1 Examples of entities extracted by SoftNER

Entity name	Data type	Example
Problem type	Alphabetical	VNet Failure
Exception message	Alphabetical	The vpn gateway deployment operation failed due to an intermittent error
Failed operation name	Alphabetical	Create and Mount Volume
Resource Id	URI	/resource/2aa3abc0-7986-1abc-a98b-443fd7245e6f/resourcegroups/cs-net/providers/network/frontdoor/
Tenant Id	GUID	4536dcd6-e2e1-3465-a22b-d25f62456233
Vnet Id	GUID	45ea1234-123b-7969-adaf-e0255045569e
Link with details	URI	https://thephone-company.com/caseview?cid=12
Source IP	IP Address	198.168.0.1
Status Code	Numeric	500

This priority order has been implemented to ensure that we pick the most specific data type, in case of values that might match multiple data types. For instance, the priority order `GUID > Non-AlphaNumeric`, ensures that all GUIDs (e.g., Subscription ID, Deployment ID) are tagged as the `GUID` data type, although they might also match the `Non-AlphaNumeric` type. This procedure enforces that across our dataset, a given value is always tagged as the same data type. But, it is possible for other values for a given entity type to have different data types.

5.3 Label Propagation

With the unsupervised tagging, we have bootstrapped the training data using pattern extractors and heuristics. While this allows us to generate a seed data set, the recall would suffer since the entities could occur outside the context of the chosen key-value or tabular patterns. In the absence of ground truth or labeled data, it's a nontrivial problem to solve. Thus, to avoid overfitting the deep learning model on specific patterns that were used to bootstrap labeled data, we want to generalize and diversify the labels.

We use the process of *label propagation* to solve this challenge. We use the entity values extracted in the bootstrapping process and propagate their types to the entire corpus. In this procedure, we first ensure to perform *label propagation* only for untagged values that are of specific data types (such as GUID, URI, and IP addresses). For instance, we skip entities that are of Boolean data type (e.g. Grant Permission) or descriptive Alphabetic text (e.g. Problem Type, Exception). This significantly reduces the errors we make while using a simple technique to augment our dataset. Next, for each untagged value, we identify the distinct possible entity types from the bootstrapped dataset. Here, we avoid propagating labels for any value with $>= 3$ distinct possible entity types, as a random decision may incur mistakes. For example, we do not propagate the label for "127.0.0.1" as it could be a "Source IP", "Destination IP", or even a "VNet Virtual IP". This further reduces errors that maybe incurred where the decision is harder to make using a simple approach.

Finally, we are left with values that have either one or two possible entity types. In these remaining cases, from our analysis we observed that a large majority (98%) of the values have a single possible entity type. For these we simply propagate the label. For the minority (2%) cases, where there are two possible entities for a value, we resolve conflicts based on popularity, i.e., the value is tagged with the entity type which occurs more frequently across the corpus. Note, that we provide some descriptive statistics regarding the impact of this simple approach and also potential threats to it's validity in Section 10.

5.4 Multi-Task Named-Entity Recognition Model

The previous sections explain the phases of the SoftNER NER pipeline, as shown in Fig. 4, that automate the significant task of creating labeled data. Here, we propose a novel Multi Task deep learning model that further generalizes entity extraction. The model solves two entity recognition tasks simultaneously - Entity Type recognition and Data Type recognition. The model uses an architecture, as described in Fig. 5, that shares some common parameters and layers for both tasks, but also has task specific layers. Incident descriptions are converted to word level vectors using a pre-trained GloVe Embedding layer. This sequence of vectors is interpreted by a Bi-directional LSTM layer. We then have distinct layers for the two tasks. The attention mechanism helps the model learn important sections of the sentences. Finally, the CRF layer produces a valid sequence of output labels. We perform back

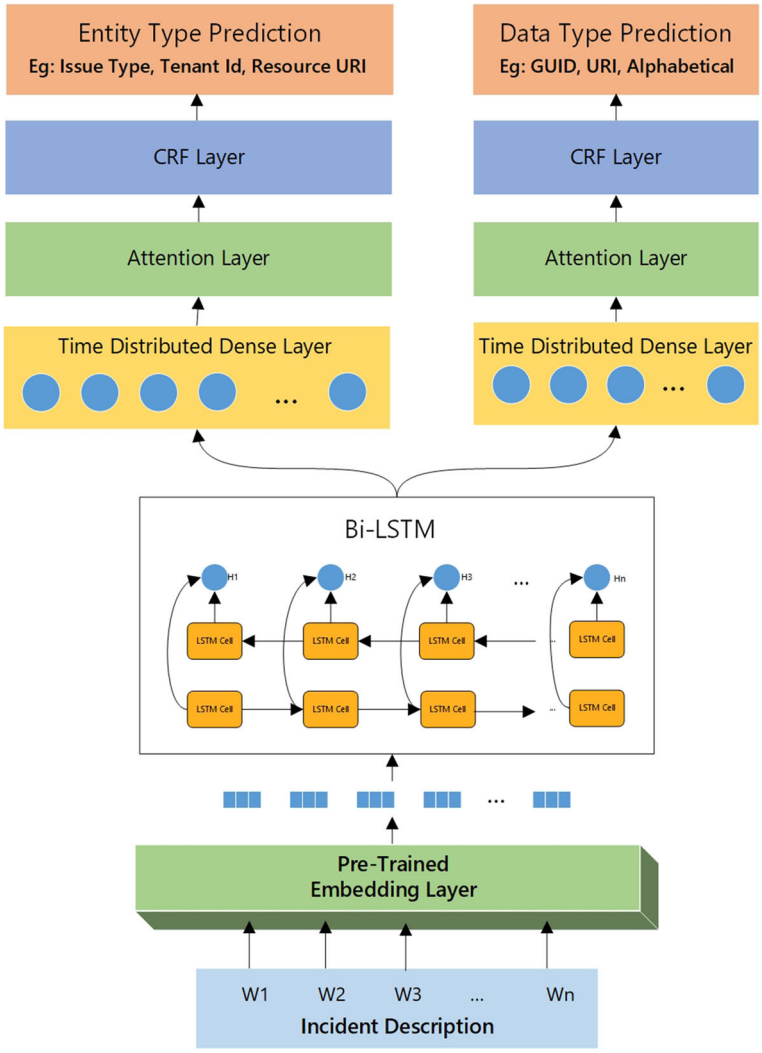

Fig. 5 Multi-task model architecture

propagation using a combination of loss functions during training and evaluate tag level precision, recall, and F1 metrics. In the following sub sections we describe the important layers and approaches used in our model.

5.4.1 Word Embeddings

Language models in the semantic vector space, require real valued vectors as word representations (Collobert et al. 2011). GloVe (Pennington et al. 2014) vectors, demonstrated on tasks such as word analogy and named entity recognition in (Pennington et al. 2014), outperform various other word representations. Therefore, we use GloVe by creating an embedding layer with pre-trained GloVe weights loaded in our model as well as our baselines.

5.4.2 Bi-directional LSTM

Recurrent Neural Networks (RNN) have been the basis for numerous language modelling tasks in the past (Mikolov et al. 2010). But, RNNs tend to be biased towards more recent updates in long sequence scenarios. Long Short-term Memory (LSTM) networks (Hochreiter and Schmidhuber 1997) were designed to overcome the problems associated with vanilla RNNs. Their architecture allows them to capture long range dependencies using several gates. These gates control the portion of the input to give to the memory cell, and the portion from the previous hidden state to forget. Given a sentence as a sequence of real valued vectors $(x_1, x_2, .., x_n)$, the layer computes $\overrightarrow{h_t}$ which represents the leftward context of the word at the current time step t. A representation of a word receiving context from words occurring after it is achieved with a second LSTM that interprets the same sequence in reverse, returning $\overleftarrow{h_t}$ at each time step. This combination of forward and backward LSTM is referred to as Bi-Directional LSTM (BiLSTM) (Graves and Schmidhuber 2005). The final representation of the word is produced by concatenating the left and right context, $h_t = [\overrightarrow{h_t}; \overleftarrow{h_t}]$.

5.4.3 Neural Attention Mechanism

In recent years attention mechanism has become increasingly popular in various NLP applications like neural machine translation (Bahdanau et al. 2014), sentiment classification (Chen et al. 2017) and parsing (Li et al. 2016). Novel architectures like transformers (Vaswani et al. 2017) and BERT (Devlin et al. 2018) have proven the effectiveness of such a mechanism for various downstream tasks. We implement attention at the word level as a neural layer, with a weight parameter W_a. It takes as input the the hidden states from the BiLSTM, transposed to output dimensions using a time distributed dense layer. Let $h = [h_1, h_2, ..h_T]$ be the input to the attention layer. The attention weights and final representation h^* of the sentence is formed as follows:

$$scores = W_a^T h \tag{1}$$

$$\alpha = softmax(scores) \tag{2}$$

$$r = h\alpha^T \tag{3}$$

$$h^* = \tanh(r) \tag{4}$$

We visualize the attention vector α for a test sentence in Fig. 6, where we observe that it learns to give more emphasis to tokens that have a higher likelihood of being entities. In Fig. 6, the darkness in the shade of blue is proportional to the degree of attention. In case of long sequences, this weighted attention to certain sections of the sequence, that are more likely to contain entities, helps improve the model's recall/sensitivity.

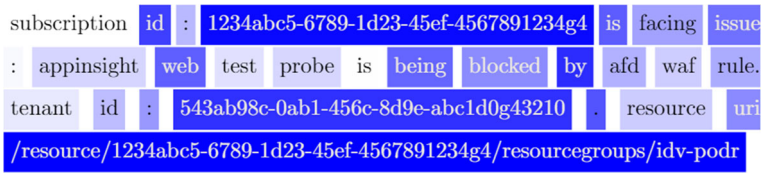

Fig. 6 Attention visualization on a sample input

5.4.4 Conditional Random Fields

Simply using hidden state representations (h_t) as word features to make independent tagging decisions at the word level leaves behind inherent dependencies across output labels in tasks like Named Entity Recognition. Our NER task also has this characteristic since the initial SoftNER heuristics enforce structural constraints, e.g. separators between key-value and html table tags. In learning these dependencies and generalizing them to sentences without these constraints, we model tagging decisions jointly using conditional random fields (Lafferty et al. 2001).

For an input sequence $\boldsymbol{X = (x_1, x_2, .., x_n)}$, let $\boldsymbol{y = (y_1, y_2, .., y_n)}$ a potential output sequence, where n is the no. of words in the sentence. Let P, the output of the BiLSTM network passed through the dense and attention layers, be the matrix of probability scores of shape $n \times k$, where k is the number of distinct tags. That is $P_{i,j}$ is a score that the i^{th} word corresponds to the j^{th} tag. We define CRF as a layer in the model, whose working is as follows.

$$s(\boldsymbol{X}, \boldsymbol{y}) = \sum_{i=0}^{n} A_{y_i, y_{i+1}} + \sum_{i=0}^{n} P_{i, y_i} \tag{5}$$

$$p(\boldsymbol{y}|\boldsymbol{X}) = \frac{e^{s(\boldsymbol{X}, \boldsymbol{y})}}{\sum_{y' \in Y} e^{s(\boldsymbol{X}, \boldsymbol{y'})}} \tag{6}$$

Here A represents the matrix of transition scores where $A_{i,j}$ is the score for the transition from tag_i to tag_j. The score $s(X, y)$ is converted to a probability for the sequence y to be the right output using a softmax over $\boldsymbol{Y}$(all possible output sequences). The model learns by maximizing the log-probability of the correct y. While extracting the tags for the input, we predict the output sequence with the highest score - $y^* = argmax_{y' \in Y}\, p(y'|X)$.

5.4.5 Multi-Task Learning

Caruana (1997) defines Multi-Task Learning (MTL) as an approach to improve generalization in models by using underlying common information shared among related tasks. Some well known applications of MTL are multi-class and multi-label classification. In the context of classification or sequence labelling, MTL improves performance of individual tasks by learning them jointly.

In SoftNER, named-entity recognition is the primary task. In this task, models mainly learn from context words that support occurrences of entities. But we also observe that incorporating a complementary task of predicting the data-type of a token reinforces intuitive constraints indirectly on model training. For example in an input like "*The SourceIPAddress is 127.0.0.1*", the token *127.0.0.1* is identified more accurately by our model, as the entity type *Source Ip Address*, because it is also identified as the data-type *Ip Address*, in parallel. This supplements the intuition that all *Source Ip Addresses* are *Ip Addresses*, thus, improving model performance.

As shown in Fig. 5, we use a multi-head architecture, where the lower level features generated by the BiLSTM layers are shared, whereas the other layers are task specific. We define the entity type prediction as the main task and that of data type prediction as the auxiliary task. For both tasks we use categorical cross entropy as the loss function. We first calculate loss individually for both objectives, say l_1 and l_2. We then compute and minimize a combined loss by averaging: $loss_c = (l_1 + l_2)/2$. During training, the objective we minimize is the combined $loss_c$. But, during back-propagation, we make sure to use the individual task losses (l_1 and l_2) to update weights of task-specific branches of the network

(Fig. 5). With such an approach, the shared layer (BiLSTM) is trained by both tasks, because both l_1 and l_2 update it via back-propagation. Whereas the task specific layers (Attn and CRF) are trained only on their respective individual loss functions.

6 Knowledge Graph Construction

Next, we describe our approach for mining entity relations and automatically constructing knowledge graphs.

6.1 Entity Relation Extraction

Once named-entities in incidents are tagged by SoftNER's trained NER model, we recognize pairs of related entities, that is, binary relations. Here, instead of directly classifying all possible relation instances (n-ary), we first identify whether a given pair of entities is related or not. With the aim of building an unsupervised framework, similar to prior work (McDonald et al. 2005; Zelenko et al. 2003), we use co-occurrence of entity pairs in a sentence to extract binary relations. By extracting all possible entity pairs that follow this assumption, we get a noisy candidate set of binary entity relations based solely on co-occurrence.

Consequently, we then score each candidate tuple using a co-occurrence based measure and filter noisy candidates. In information theory, *mutual information* (MI) (Fano 1961) of 2 random variables is a measure of the "amount of information" obtained about one variable through observing the other. While *mutual information* averages the measure over all possible outcomes, *pointwise mutual information* (PMI) (Church and Hanks 1990) is defined for a single event (i.e. pair of outcomes). Mathematically, PMI is defined as described in (7).

$$\text{pmi}(x;\, y) \equiv \log \frac{p(x,\, y)}{p(x)p(y)} = \log \frac{p(x|y)}{p(x)} = \log \frac{p(y|x)}{p(y)} \tag{7}$$

$$\text{pmi}(x;\, y) = \text{pmi}(y;\, x) \tag{8}$$

PMI has been applied for finding collocations and associations between tokens (Church and Hanks 1990; Thanopoulos et al. 2002) by leveraging frequency of occurrences to approximate probabilities. We observe that our aim to score co-occurring entity pairs on their relatedness is analogous to these applications. Thus, we use a variant of PMI - ***normalized pointwise mutual information*** **(NPMI)** (Bouma 2009), to score each entity pair - (e_1, e_2), as described in (9)–(11).

$$\text{npmi}(e_1; e_2) = \frac{\text{pmi}(e_1; e_2)}{-\log p(e_1, e_2)} \tag{9}$$

$$\text{pmi}(e_1; e_2) = \log \frac{p(e_1, e_2)}{p(e_1)p(e_2)} \tag{10}$$

$$p(e_1) = \frac{f_1}{f_{total}};\; p(e_2) = \frac{f_2}{f_{total}};\; p(e_1, e_2) = \frac{f_{joint}}{f_{total}}; \tag{11}$$

In the above equations and in Table 2, f_1 & f_2 are frequencies of the entities e_1 & e_2 respectively, f_{joint} is the frequency of co-occurrence of the entity pair, and f_{total} is the total frequency of all entities. Note that, from (8), PMI, and consequently NPMI is symmetric. Therefore, all binary entity relations extracted are undirected in nature. NPMI scores $\in [-1, 1]$, resulting in -1 for never occurring together, 0 for independence, and +1 for complete co-occurrence. Using this, we eliminate all entity pairs with NPMI < 0 as noise, leaving us with a final set of binary entity relations. Table 2 shows some examples of entity pairs and their NPMI scores.

Table 2 Examples of extracted binary entity relations

Entity pair	(f_1, f_2, f_{joint})	NPMI
(*Remote Port Range, Remote Address*)	(564, 558, 557)	0.99
(*Tunnel Name, Encap Type*)	(761, 654, 486)	0.88
(*VNet Name, VNet Id*)	(860, 985, 432)	0.78
(*Gateway Id, VNet Id*)	(1071, 985, 124)	0.47
(*Tunnel Name, Destination IP*)	(761, 72908, 1)	-0.38
(*Destination IP, Subscription Name*)	(72908, 19125, 11)	-0.55

6.2 Entity Knowledge Graph

A knowledge graph formally represents semantics by describing entities and relationships. It captures information that can be queried like traditional databases, analyzed like graph data structures, and allows inference of new knowledge. In the incident management space, this unlocks various applications, such as mining complex interactions of cloud resources (Table 3), or inferring the relevance of an entity to the issue described in an incident (Section 9.2).

Having identified binary relations, we construct an undirected knowledge graph $\boldsymbol{G = (V, E)}$, where nodes $\boldsymbol{V}$ are entities and edges $\boldsymbol{E}$ are binary relations between pairs of entities. We also assign weights $\boldsymbol{W_{e_i,e_j}} = \text{npmi}(\boldsymbol{e_i; e_j})$, to the edges between all entity pairs $(\boldsymbol{e_i}, \boldsymbol{e_j})$. Figure 7 shows a sub-graph of related entity types in the entity knowledge graph constructed from our incident data set. These relations are utilized to construct the complete knowledge graph, as shown in Fig. 3.

We briefly explore mining more complex n-ary relations from our knowledge graph. One simple approach is to view n-ary relations as graph cliques - a subset of vertices of an undirected graph such that every pair of vertices are adjacent (pairwise related entities) (McDonald et al. 2005). To overcome overlaps in cliques, we query maximal-cliques, that is, those cliques that are not subsets of other cliques. Table 3 shows some examples of complex relations extracted with this method and what they describe. While we do not perform statistical analysis, examples from Table 3 suggest that the constructed knowledge graph is able to fairly represent complex relationships by factoring them as binary relations.

Table 3 Examples of extracted Complex (n-ary) Entity Relations

Related entities	Relation description
(*Destination IP, AS path, Output packets*)	Describing a BGP routing incident causing connection issues.
(*Correlation Id, Allocation Id, VNet Id, MAC Address, Container Id*)	Describing various tasks in network manager setup for a VM.
(*VNet Name, VNet Id, Gateway Id, Tunnel Name, Encap Type*)	Describing a VNet gateway instance that is down.
(*Error Code, Error message, is retriable exception, is user error*)	Entities that describe errors and exceptions.
(*VNet Name, VNet id, VNet region, v net name, v net id, Resource URI*)	Entities co-referring a single resource related to the incident.

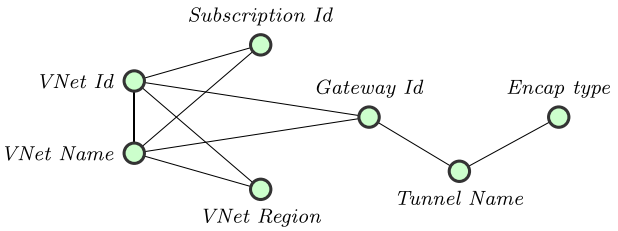

Fig. 7 Sub-graph of related entities

7 Implementation

The SoftNER implementation and deployment comprises of various modules. First, we train the ML models using historical incident data from Microsoft's services. Next, we deploy the models to a scalable REST API using the Flask web framework. Lastly, we integrate the SoftNER API into the incident management system at Microsoft. Here, we describe each of these in detail:

7.1 Model Training

We have implemented SoftNER and all the machine learning models using Python 3.7.5, with Keras-2.2.4 and the tensorflow-1.15.0 backend. The hyper-parameters for the deep learning models are set as follows: word embedding dimension used is 100, hidden LSTM layer size is set to 200 cells, and, maximum length of a sequence is limited to 300. These optimal hyper-parameters were chosen to train a robust yet light weight model and were re-used among all models. The embedding layer uses pre-trained weights from stanford-nlp's glove.6B.100d. We train the model for a maximum of 100 epochs, but use an early stopping strategy to stop training when model performance on a validation dataset starts to degrade. This helps prevent the model from overfitting on train data. For training the model efficiently, instead of setting a constant learning rate, we use the Adam optimizer (Kingma and Ba 2015) which computes individual adaptive learning rates for different parameters from the gradients. Our models are trained on an Ubuntu 16.04 LTS machine, with 24-core Intel Xeon E5-2690 v3 CPU (2.60GHz), 112 GB memory and 64-bit operating system. The machine also has a Nvidia Tesla P100 GPU with 16 GB RAM.

7.2 Model Deployment

We have also deployed SoftNER as a REST API developed using the Python Flask web app framework. The REST API offers a POST endpoint which takes the incident description as input and returns the extracted entities in JSON format. We have deployed it on Microsoft's Azure cloud platform which allows us to automatically scale the service based on the variation in request volume. This enables the service to be cost efficient since majority of the incidents are created during the day. We have also enabled application monitoring which alerts us in case the availability or the latency regresses.

7.3 Integration

At Microsoft, we have thousands of production services built and operated by tens of thousands of developers. In order to effectively and efficiently handle the issues and regressions

in these services, we have a dedicated incident management platform called IcM. This platform allows internal and external partners to create an incident against various services and teams. The IcM platform provides an extensibility mechanism using which custom modules can be enabled which can subscribe to various events such as incident creation, update, mitigate and resolution. We have integrated SoftNER API with the IcM platform to surface the insights directly into the IcM portal. We have enabled multiple integration points based on the user scenario:

- *Manual Trigger* - Any developer can enter an incident Id and see the SoftNER results in real time. This is useful when a team or a developer wants to try out SoftNER.
- *Auto Trigger* - Service owners can also enable SoftNER for the incidents belonging to their respective teams. This way, the insights are added to the incident automatically even before an on-call engineer is engaged.
- *Integration with other modules* - The entities extracted using SoftNER can be used to trigger other diagnostic modules. For example, a network diagnostic module may require the VNet Id and the Source IP Address to localize a given incident. SoftNER is able to extract these entities automatically and can feed them to such diagnostic modules.

8 Evaluation

SoftNER solves the problem of knowledge extraction from unstructured text descriptions of incidents. To evaluate the SoftNER framework in its entirety, we propose a four phase evaluation:

- **Entity Types**: How does SoftNER's unsupervised approach perform in recognizing distinct entity types?
- **Unsupervised Labeling**: How does SoftNER's unsupervised labeling pipeline perform in creating an accurately labeled data set?
- **Named-Entity Recognition**: How does SoftNER's Multi-Task model compare to state-of-the-art deep learning approaches for the NER task?
- **Entity Relations**: How does SoftNER's unsupervised approach perform in extracting and scoring entity relations using NPMI?

8.1 Study Data

In the following evaluation experiments, we apply SoftNER to service incidents at Microsoft, a major cloud service provider. These are incidents retrieved from large scale online service systems, which have been used by a wide distribution of users. In particular, we collected incidents spanning over a time period of 2 months. Each incident is described by its unique Id, title, description, last-modified date, owning team name and, also, whether the incident was resolved or not. Incident description is the unstructured text with an average of 472 words, showing us how verbose the incident descriptions are. Owning Team Name here, refers to the team to which the incident has been assigned.

8.2 Entity Type Evaluation

Here, we evaluate the effectiveness of SoftNER's unsupervised approach to identify the distinct types of entities in the study data. To do so, we first allow SoftNER to identify

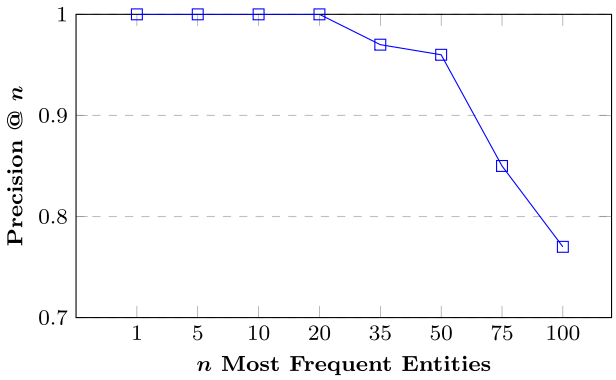

Fig. 8 Precision vs rank curve for entity types

the top-100 (limited to 100 since evaluation is manual) most frequent distinct entity types. We then manually evaluate the correctness of each entity type by verifying if it is a valid software entity type of interest or a noisy/stray n-gram. Lastly, we compute precision as the fraction of entity types that are valid software entity types.

Since the precision of SoftNER's entity type extraction depends on the frequency of occurrence of entities, we further plot precision against a cut off rank n. Figure 8 summarizes the precision of SoftNER's entity type extraction against the top n entities extracted, where $n \in [1, 100]$. From this analysis, we see that SoftNER is able to identify entity types at a high precision of 0.96 at rank 50. These top-50 distinct entities account for 69.6% of total entities found in the dataset. Further, we see that SoftNER is able to extract 77 valid entities per 100 entities. The top-100 entities account for 83.6% of total entities found in the dataset. In this experiment, n corresponds to the rank of the entity extracted with respect to frequency of occurrence. That is, a higher n refers to an entity with low frequency of occurrence, which in turn can be extrapolated as an entity that is less important. We thus see an expected decrease in precision, as n increases, due to noisy tokens (false positives) like *"to troubleshoot issue"* and *"for cleanup delay"*. SoftNER's unsupervised entity type extraction has a minimal precision variation, also known as fall out rate, of 0.23 for an n value as high as 100. This strengthens the hypothesis that SoftNER's pattern extractors can pick up entities from unstructured text effectively, in a completely unsupervised manner.

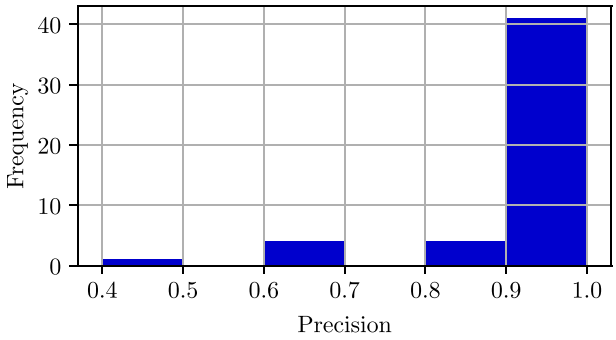

Fig. 9 Distribution of precision of unsupervised labeling

Table 4 NER model evaluation

Metric	BiLSTM-CRF	BiLSTM-CRF Attention	SoftNER Model
Avg F1 (± std)	0.8803 (± 0.223)	0.8822 (± 0.211)	**0.9572** (± 0.075)
Weighted Avg F1 (± std)	0.9401 (± 0.223)	0.9440 (± 0.211)	**0.9682** (± 0.075)
Avg Precision (± std)	0.9160 (± 0.211)	0.9088 (± 0.194)	**0.9693** (± 0.073)
Avg Recall (± std)	0.8669 (± 0.244)	0.8764 (± 0.234)	**0.9525** (± 0.098)

Entries in boldface represent the best result in the experiment

8.3 Unsupervised Labeling Evaluation

A key component of SoftNER is its unsupervised labeling and bootstrapping pipeline. Here, we manually evaluate the effectiveness of this pipeline to generate an accurately labeled data set on which machine learning models can be trained. To do so, we first select the top-50 (limited to 50 since evaluation is manual) most frequent distinct entity types. Then, we create an evaluation set of 250 data points by randomly selecting 5 labeled occurrences for each of the top-50 entities from the unsupervised labeled data set. We then manually evaluate each occurrence by verifying if it is correctly or incorrectly labeled. We then compute the precision of unsupervised labeling for each entity type (i.e., # correctly labeled entities/5).

From this analysis, we find that the unsupervised labeling pipeline has a high average precision of 0.94. Figure 9 further shows the distribution of this precision for the top-50 entities. As shown, 82% (41/50) of the entities had a precision $>$ 0.9 and 90% (45/50) of the entities had a precision $>$ 0.8. This shows that SoftNER's unsupervised framework can effectively produce good quality labeled data, although it uses syntactic patterns and simple heuristics in a completely unsupervised manner. Note, that while we observed a high precision for the vast of majority of cases, we also observe some false positives due to the limitations of methods used. We address the impact of these limitations and potential threats to validity in Section 10.

8.4 Named-Entity Recognition Evaluation

Here, we evaluate the SoftNER deep learning model on the Named-Entity Recognition Task. We compare SoftNER's multi task model, described in Section 5.4 and Fig. 5, against two baseline models, BiLSTM-CRF and BiLSTM-CRF with attention mechanism. Here, note that the BiLSTM-CRF Attention model is similar to SoftNER's model architecture but without multi-task learning. These baseline models are state-of-the-art for NER (Huang et al. 2015; Lample et al. 2016; Chiu and Nichols 2016) and other NLP tasks as well. The models are compared on a fixed test set of tens of thousands of incidents[6], that accounts for 20% of the ground-truth data set labeled using the unsupervised approach (Section. 5.2). Note, that we also ensure the incidents in the test set occur after those in the training set temporally. The number of distinct entity types (labels) in our test dataset is 60. The total number of entities in our test data set is 69371. We use average precision, recall, and F1 metrics to evaluate and compare the models on the NER task. In Table 4, we report the mean and the standard deviation of these metrics over the 60 distinct entities types tagged by the model.

[6]We cannot disclose the exact number of incidents due to Microsoft Policy.

As shown in Table 4, we observe that the baseline BiLSTM-CRF, with and without attention mechanism, achieves an average F1 score of around 0.88. Whereas, SoftNER's Multi Task Model, as described in Section 5.4, achieves a higher average F1 score of around 0.96, i.e., a $\Delta F1\%$ of 8.7%. This significant increase in performance showcases the value of multi-task learning in SoftNER that leverages both context and complementary information to perform named-entity recognition. We also observe a high average recall of 0.95, reflective of a robust ability to extract a lot of relevant information from descriptions which directly correlates with the ease of understanding the problem and identifying resources affected by the incident.

We further analyze the generalization of the model by analyzing test examples that were falsely labeled. Table 5 shows a few examples of sentences and the entities extracted from them. Note that we refer to false positives as FP, and, false negatives as FN in the table. We observe that some of the FPs are actually correct and were mislabeled in the test set because of the limitations of the pattern extractors. Let's take Example 1 from Table 5 for instance. Here, the unsupervised labelling component was only able to label ***"2aa3abc0-7986-1abc-a98b-443fd7245e6"*** as **Subscription Id**, but not ***"vaopn-uk-vnet-sc"*** as **Vnet Name**, due to restrictions with pattern extractors and label propagation. But the SoftNER model was able to extract both the entities from the sentence, proving it's ability to generalize beyond obvious structural pattern rules visible in the training data. Row 2 shows a similar false positive example with the extraction of ***192.168.0.5*** as **IP Address**. We also show a few contrasting false negatives, in rows 4 and 5, where the model was unable to extract entities **Ask** and **Ip Address** respectively.

8.5 Entity Relation Evaluation

Next, we evaluate our unsupervised approach to extract binary entity relations and score them using NPMI. These binary entity relations and their scores represent the crux of the knowledge graph constructed in Section 6. In turn, any inference performed on the graph, like entity recommendation (Section 9.2), depends on the quality of these relations. Consequently, we manually evaluate the precision of NPMI scores given to each entity pair. First, the top 150 (limited to 150 since evaluation is manual) most frequent entity pairs are sampled. We then manually evaluate the correctness of each NPMI score computed for an entity pair by verifying if the NPMI score reflects the expected relationship between entities. For example, the pair *(VNet Name, VNet Id)* are expected to be related and we verify that the computed NPMI of 0.66 is a valid score. Similarly, the pair *(Input Packets, Destination IP)*

Table 5 FP and FN examples

Sentence	Result	Entities tagged
SubscriptionId:2aa3abc0-7986-1abc-a98b-443fd7245e6 unable to delete vnet name vaopn-uk-vnet-sc	FP	2aa3abc0-7986-1abc-a98b-443fd7245e6, vaopn-uk-vnet-sc
Device Name: njb02-23gmk, pa: 192.168.0.5 could not be configured!	FP	njb02-23gmk, 192.168.0.5
The customer's main ask: Need help to access cloud storage	FN	–
The loopback (ipv4) address (primary) is 192.131.75.235	FN	ipv4

are expected to be related, so the computed NPMI of -0.27 is marked as an invalid score. Lastly, we compute precision as the fraction of entity pairs that have a valid NPMI score that reflects their relationship.

We further plot precision against rank $n \in [0, 150]$ in Fig. 10. Here, n corresponds to the rank of the entity pair extracted with respect to frequency of co-occurrence (f_{joint}). From this analysis, we see that NPMI scores are valid for 90 per 100 entity pairs. A precision of 0.9 for a n value as high as 100, strengthens the hypothesis that binary entity relations can be effectively extracted and scored with our approach.

9 Applications

Automated knowledge extraction from service incidents can unlock several applications and scenarios. Here, we explore and evaluate the value of extracted knowledge for two applications. First, we show that entities extracted by SoftNER can be utilized to improve simple machine learning models for incident triaging. Next, we show that the knowledge graph can be used to build entity recommenders that can improve tooling in incident management platforms and in turn reduce customer impact.

9.1 Auto-Triaging of Incidents

Incident triaging is the process of assigning a new incident to the responsible team. This is currently manually performed by on-call engineers. It is not uncommon for an incident to be rerouted to different teams until the appropriate team is engaged, thereby reducing the accuracy and efficiency of incident management. Based on an empirical study, Chen et al. (2019a) showed that the reassignment rate for incidents can be as high as 91.58% for online services at Microsoft. Several efforts (Chen et al. 2019a, b) have been made to automate the triaging process by leveraging the title, description and other meta-data of the incidents. Here, we evaluate the effectiveness of the knowledge extracted by SoftNER for the downstream task of automated triaging of incidents. Incident triaging is essentially a multi-class classification problem since the incident could be assigned to one of many teams.

We sample 20% of resolved incidents for the 10 most common teams from the initial incident set (refer Section 8.1) and run the SoftNER model on the description to extract

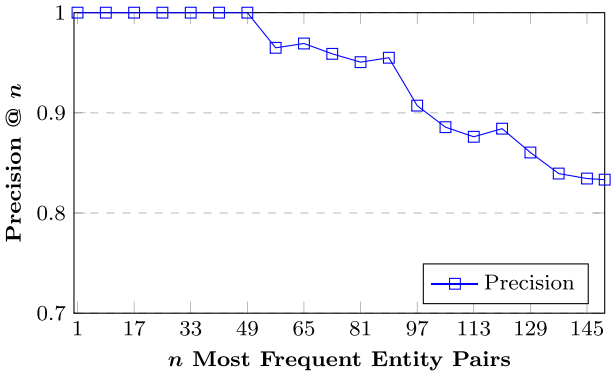

Fig. 10 Precision vs rank for entity relations

Table 6 Comparison of accuracy for auto-triaging

Feature set	Random Forest	Linear SVM	Gaussian SVM	K-Nearest Neighbors	Naive Bayes
Title + Description	74.64	85.93	87.06	81.32	69.69
SoftNER Entities	93.38	93.34	93.39	92.40	87.67
Δ %	22.31	8.26	7.02	12.76	22.85
SoftNER Entities + Title	**98.60**	**99.20**	**98.95**	**99.14**	**88.07**
Δ %	27.66	14.34	12.78	19.75	23.30

Entries in boldface represent the best result in the experiment

entities. These extracted entity values can now act as additional features to triaging models. The SoftNER entities can be broadly classified as either categorical or descriptive. While the descriptive entities are transformed to word embeddings using the same process described in Section 5.4.1, the categorical entities are encoded into one-hot vectors. We then look at different combinations of features and compare the 5-fold cross-validation accuracy on various classification models. It is evident from Table 6 that the models using the SoftNER entities as features, either on their own or along with the title, outperform the baselines that use only raw title and description information. We observe significant margins, with up to 7% - 27% increase in the cross-validation accuracies. These results reinforce that the entities extracted by SoftNER are indeed useful and can significantly help in downstream tasks. Using the entities extracted by SoftNER also reduces the input feature space since we no longer have to use the whole incident description. We also achieve high performance using simple machine learning models thereby eliminating the need for complex deep learning models which have proven to be superior in past studies (Chen et al. 2019a).

In addition to comparing accuracy, we analysed feature significance by using the feature_importances_ attribute of a random forest model trained on the various input features. As shown in Table 7, we observe that the entities extracted by SoftNER were given more importance compared to the 'Title', with top features being - 'exception message', 'problem type', 'ask' and 'issue'. This re-emphasises that the entities extracted from SoftNER boost the performance of classification models for the downstream task of automatic triaging of incidents.

9.2 Entity-Incident Relevance and Recommendation

Although extracting all entities from an incident is useful, certain entities are more important for the investigation and mitigation of an incident. Let's consider a real incident reported by a customer of the Cloud Networking service operated by Microsoft. The incident was caused due to an error in deleting a Virtual Network resource. The key information required to mitigate this incident is actually the ***Virtual Network (VNet) Id***. But the customer had not mentioned the *VNet Id* in their description. In this case, it would be useful to either

Table 7 Importance scores for top features

Feature	Exception Message	Problem Type	Ask	Issue	Title
Importance	0.0133	0.0111	0.0097	0.0051	0.0009

intelligently recommend or incorporate such entities as mandatory in incident reporting forms to alleviate downstream mitigation tasks. Thus, here we evaluate the effectiveness of the knowledge extracted by SoftNER to infer the relevance of entities to the issue described in the incident for recommendation.

Previously, statistical entity-topic models (Newman et al. 2006; Kim et al. 2012) have been studied to map named-entities to topics for document topic analysis. Bhargava et al. (2019) proposed multiple methods to learn to map wikidata entities to pre-defined topics. The challenge, in our case, was the lack of a labeled set of incidents based on pre-defined topics. Also, manually identifying topics apriori would be difficult to scale to a multitude of services. As a result, in our approach, we use clustered incident titles as representatives of topics and map a subset of ranked related entities leveraging the previously constructed knowledge-graph.

9.2.1 Clustering Incident Titles

Inferring a subset of entities for each incident individually poses challenges. It is difficult to apply rule-based approaches since they are computationally expensive and also cannot be generalized to unseen incident examples. Hence, we first aim to group incidents by clustering their titles, which are generally a representative summary of the incident. Here, we convert incident titles to 100 dimension embeddings by averaging GloVe (Pennington et al. 2014) vectors for all tokens in the title. For clustering, we make use of DBSCAN - Density-Based Spatial Clustering of Applications with Noise (Ester et al. 1996), a clustering algorithm that does not require us to define the number of clusters apriori. The DBSCAN hyper-parameter ϵ, i.e. maximum distance between two samples for them to be in the same neighborhood, was tuned using the elbow method and plotting k-distance graphs, as suggested in the original paper (Ester et al. 1996). This provided us with over 50 clusters on our incident data set. From Table 8, we observe that clustering reduces complexity and also uncovers underlying topics in incidents.

9.2.2 Inferring Related Entities

Having clustered incident titles, and as a result the corresponding incidents, we now infer a related entity set for each cluster. First, for each cluster, we extract the top-5 most frequently

Table 8 Titles and top-5 related entities

Clustered titles	Top-5 related entities
Unable to delete Vnet Vnet stuck in updating state Vnet: Unable to delete Public IP	(VNet Id, 1.0) (Allocator Service URI, 0.70) (Availability Zone, 0.68) (VNet Name, 0.66) (Tunnel Name, 0.63)
VM Network Profile fails to load High IO latency networking suspicious Ping Mesh Drops networking suspicious	(Subscription Id, 1.0) (VNet Name, 0.41) (Deployment Id, 0.39) (Resource Id, 0.35) (VNet Id, 0.33)
MAC Request - Port Provisioning Port provisioning 1 device Ports mop sent for review	(Closed in SLA, 1.0) (SLA w/o Dependency, 0.44) (Device Count, 0.40) (Dependency Time, 0.37) (Allocator Service URI, 0.31)
Cluster unhealthy - Interface Down Unhealthy cluster - BGP Session Down Cluster is unhealthy : Device Reloaded	(SNMP Interface Index, 1.0) (Allocator Service URI, 0.32) (Availability Zone , 0.31) (Resource Id, 0.28) (Interface Address, 0.27)

occurring entities in the cluster's incidents (ranked by frequency). Next, we loop over this top-5 list and search for the first entity that exists in the knowledge graph $G = (V, E)$ constructed previously. Let this entity be called the "***primary entity***" ($\boldsymbol{e_p}$) of a cluster C. We then use the *primary entity* as the source node to find the shortest paths to every other reachable entity. We hypothesize that every entity ($\boldsymbol{e_x}$) reachable from the primary entity ($\boldsymbol{e_p}$) of a cluster is related to the incidents of that cluster. Then, the relatedness between any reachable entity ($\boldsymbol{e_x}$) and a cluster (C) is scored as the average of the edge weights in the path taken from the *primary entity* to the entity of interest ($\boldsymbol{e_x}$). Mathematically, it is defined as stated in (12). We then rank these entities based on the score computed and choose the top-K entities as entities related to the cluster. Note, that the primary entity itself is given a score of 1.0. Also, we use averaging instead of product here, since the edge weights $\boldsymbol{W_{e_i,e_j}} = \text{npmi}(\boldsymbol{e_i; e_j})$, and NPMI is a function of log probabilities (Section 6).

$$\text{relation-score}(e_x; C) = \frac{\sum_{\forall e_i \in path(e_p, e_x)} W_{e_i, e_{i+1}}}{\sum_{\forall e' \in path(e_p, e_x)} 1} \tag{12}$$

$$\text{where } path(\boldsymbol{e_p}, \boldsymbol{e_x}) = \text{shortest path from } \boldsymbol{e_p} \text{ to } \boldsymbol{e_x}$$

With the above approach, top-K related entities for each cluster are pre-determined and stored. For a newly created incident with a complete incident title, first, the nearest cluster for that title is found. Then, the pre-determined top-K entities for that cluster are recommended for that incident.

Table 8 shows a few examples of clustered titles and top-5 related entities extracted using the approach described above. While we leave more formal statistical evaluation of clustering for future work, we make some informal observations here, inferring from examples in Table 8. Let's take example 1 for instance. Here, the clustered titles suggest incidents where the customer is ***unable to deleted a virtual network***. From the top-5 related entities, we see that our approach correctly identifies key entities, such as ***Vnet Id*** and ***Vnet Name***, that point to the resource of interest, i.e. the virtual network. We also observe that it identifies the ***Allocator Service*** responsible for allocation/de-allocation of the virtual network. Example 2 shows a cluster of incidents that describe ***aggregated issues in the control path*** of a virtual resource. In this case, an entire hierarchy of entities, from ***Subscription*** to ***Deployment*** to an individual ***Resource***, is identified. These recommendations are effective to quickly identify resources affected, layers of dependencies, and mitigation steps to reduce customer impact.

10 Threats to Validity

10.1 Internal Validity

Here we address some threats to the internal validity of this work, that are mainly around the unsupervised approach used to create a labeled dataset and the model architecture used.

Unsupervised Labeling. A key component of SoftNER is the unsupervised labeling pipeline that bootstraps labeled data for training the models. This pipeline is composed of techniques that use syntactic pattern extractors and heuristics: (1) Entity Type Tagging (Section 5.2.1) and (2) Label Propagation (Section 5.3). These can potentially lead to poor quality labeled data, and hence, could be a threat to validity. To mitigate this, in Section 8.3 and Fig. 9, we manually validated 250 random samples of top-50 distinct entities (5 for each). Our evaluation shows that SoftNER's unsupervised labeling pipeline

has a high average precision of 0.94. We also find that 90% (45/50) entities had a precision > 0.8. However, the remaining 10%, although represent a small number of entities, had some false positives. For instance, we find cases with incorrect labels due to key-value pair having an empty value (e.g., The VNet: **[is $_{\text{VNet}}$]** down!). We also find cases where performing label propagation without considering the context leads to false positives. We specifically investigate the impact of label propagation on the validity of this work in the next section. These false positives can affect the quality of our labeled data set and change the model performances reported. Thus, in future research we will revisit this by using more accurate approaches for unsupervised labeling.

Lastly, while we look at the precision of unsupervised labeling here, analyzing recall would require manually identifying unlabeled values (false negatives) for each of the top-50 entities from a large (tens of thousands) incident dataset. Due to this we are currently unable to compute the recall metric and will revisit this with a manually labeled dataset in future.

Label Propagation. In this work, to expand our labeled dataset, we perform a *label propagation* strategy (Section 5.3). A potential threat during label propagation is when the same value can be multiple entities (e.g., 127.0.0.1 could be Source IP, Destination IP, or VNet IP). To mitigate this, we first filter any value with >= 3 possible entity types. In the remaining cases, we observed that majority (98%) of the values have only a single entity type. For the minority (2%) cases, where there are two possible entities for a value, we choose the more frequent one.

We further investigated potential mistakes caused by using a most frequent strategy to resolve conflicts. In 75.7% (of the 2% cases) of conflicts, the two conflicting entities could be used interchangeably to describe the value. For instance, (Resource URI, Resource ID) to identify a virtual resource and (Subscription ID, Product Subscription Id) to represent a cloud subscription. Here, picking the most frequent entity type has no effect. However, the remaining 24.3%, although constituting a small number of entities, might have incorrect labels. For example, we find that 171.55.66.34 can either be Destination IP or VNet IP. Propagating the most frequent entity (VNet IP) here to all occurrences might incur mistakes. This can affect the quality of our data set and vary the model performances reported. Thus, in future research we will revisit this by using context-aware approaches for label propagation.

BiLSTM-CRF Model. In this work, we use a BiLSTM-CRF based architecture for our named-entity recognition model. More recently, large pretrained language models such as BERT (Devlin et al. 2018) have been proposed to solve various NLP tasks. BERT based models are typically highly compute intensive for both training and inference. These costs can be even more when adapting such models to custom domains (e.g., software, healthcare) that are quite different from the web corpuses that they have been trained on. Due to these practical constraints, we have limited this work to BiLSTM-CRF based models. The performance of BERT based models might be different from the models used in this work, and hence, future research can extend our work by using BERT or other recent state-of-the-art models.

10.2 External Validity

In this work, we design the SoftNER framework using domain expertise from 3 different services (networking, database, mailbox) at Microsoft. The design and evaluation of

techniques used in SoftNER, especially the unsupervised bootstrapping of labeled data using syntactic patterns, are guided by analyzing incident reports from these services. We ensured that patterns and heuristics used in these techniques are generic, commonly seen across incident reports, and hence can be effectively used across services. Here, we acknowledge that the mentioned syntactic patterns, although generic, might not be applicable to some other cloud service providers depending on the format and structure of their reports. Thus, the results of this work could be different when directly applied to incident reports from other cloud service providers. However, in such cases, SoftNER can be extended to use other specific patterns for data labeling to improve performance.

11 Discussion

In this section, we first discuss the generalizability of SoftNER and then discuss some ideas for future work.

11.1 Generalizing SoftNER

Today, incident management in large-scale cloud services is largely a manual process. On-call engineers have to read through the incident reports, extract the relevant information and then do root cause analysis and mitigation. This manual intervention also causes downstream impact on service reliability and customer satisfaction. Manual understanding and parsing of incident reports is a key bottleneck for incident automation. In this work, we have designed and deployed SoftNER at Microsoft for knowledge mining from incident reports. However, the problem and applications are generic and applicable to any cloud service provider.

To that regard, SoftNER's design is inspired by incorporating insights from product teams and observations made by analyzing incident reports from 3 different services at Microsoft. For instance, we carefully selected generic syntactic patterns for unsupervised data labeling based on various report formats used across Microsoft. We then identified simple and controllable parameters like frequency of occurrence to remove noisy extractions. Lastly, we encapsulated these observations and further generalized them with the multi-task model. These decisions are domain/service agnostic, and make SoftNER easily applicable/adaptable to incident reports from numerous services (e.g., networking, database, or mailbox service) following different report formats.

Further, since we have built SoftNER in an extensible manner using unsupervised techniques, it can be trained and applied to other service providers. Other services and service providers can follow the SoftNER framework to quickly build knowledge extraction models. Furthermore, while key-value pairs and tables are commonly seen in incident reports, SoftNER can also be extended to other syntactic patterns for unsupervised labeling, if required. For instance, in case key-value pairs or tables are uncommon, one can use methods like gazetteers (dictionaries of known values), databases, part-of-speech tags, and custom regular expressions. Using such methods, SoftNER's unsupervised labeling component can be extended, while the rest of the framework (label propagation, NER model, knowledge graph construction) can be reused to build solutions for new domains, services, or even cloud providers.

11.2 Future Work

SoftNER currently mines knowledge graphs from incidents by extracting named-entities and inferring relations between the entities. As next steps, we plan to expand the knowledge mining and leverage it for more scenarios:

1. **Incident summarization** - Lot of critical information about the incidents is embedded in natural language. For instance, incident details like symptoms, reproduction and mitigation steps is described using natural language. So, we will extend the knowledge mining beyond named-entities to even parsing and understanding natural language. The extracted information can then be used for several applications such as incident summarization.
2. **Automated health-checks** - Often times, on-call engineers have to pull up telemetry and logs for the resources affected by an incident. Or, they might look up the resource allocation for a given subscription. We will integrate SoftNER with the log mining and debugging tools at Microsoft, so that, these checks can be triggered automatically before the on-call engineers are engaged.
3. **Better tooling** - The knowledge extracted by SoftNER can also be used to improve the existing incident reporting and management tools. SoftNER identified entities can be incorporated into the incident report forms, where some of these entities can even be made mandatory fields. We have already started working on this scenario with feature teams that own incident reporting tools at Microsoft.
4. **Type-aware models** - The multi-task deep learning architecture used in SoftNER uses both the semantic and the data-type context for entity extraction. As per our knowledge, this is the first usage of a multi-task and type-aware architecture in the software engineering domain. Typically, models for source code and code related activities, such as code summarization, look at code as sequential tokens. More recently, to capture complex dependencies in code, architectures like graph neural networks (GNNs) have become popular. Although GNNs capture syntax tree information, there is an opportunity to more explicitly learn from the data types of tokens in code. Here, SoftNER's model architecture (and more generally multi-task learning) can potentially be used to learn from data types as parallel source of information, and hence improve performance for source code related tasks.
5. **Predictive tasks** - In this work, we have shown that the knowledge extracted by SoftNER can be used to build more accurate machine learning models for incident triaging. Similarly, we can build models to automate other tasks such as severity prediction, root causing, etc.
6. **Bug reports** - Even though this work is motivated by the various problems associated with incident management, the challenges with lack of structure applies to bug reports as well. We plan to evaluate SoftNER on bug reports at Microsoft and, also, on the publicly available bug report data sets.

12 Conclusion

Incident management is a key part of building and operating large-scale cloud services. In this paper, we propose SoftNER, an unsupervised framework for mining knowledge graphs from incident reports that incorporates a novel multi-task BiLSTM-CRF model for software named-entity recognition. We have evaluated SoftNER on the incident data from Microsoft,

a major cloud service provider. Our evaluation shows that even though SoftNER is fully unsupervised, it has a high precision of 0.96 (at rank 50) and 0.77 (at rank 100) for learning software entity types from unstructured incident data. We also evaluate and show that SoftNER's unsupervised pipeline accurately labels data with a precision of 0.94. Further, our multi-task model architecture outperforms existing state-of-the-art models in entity extraction. Additionally, our novel approach for mining entity relations has a high accuracy of 0.9. We have deployed SoftNER at Microsoft, where it has been used for knowledge extraction from incidents for over 6 months. Lastly, we show that the extracted knowledge can be used for building significantly more accurate models for critical incident management tasks like triaging.

References

Aguilar G, Maharjan S, López-Monroy AP, Solorio T (2019) A multi-task approach for named entity recognition in social media data. arXiv preprint arXiv:1906.04135

Anvik J, Hiew L, Murphy GC (2006) Who should fix this bug? In: Proceedings of the 28th ICSE, pp 361–370

Ardimento P, Dinapoli A (2017) Knowledge extraction from on-line open source bug tracking systems to predict bug-fixing time. In: Proceedings of the 7th international conference on web intelligence, mining and semantics, pp 1–9

Bahdanau D, Cho K, Bengio Y (2014) Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473

Bansal C, Renganathan S, Asudani A, Midy O, Janakiraman M (2020) Decaf: Diagnosing and triaging performance issues in large-scale cloud services. In: 2020 IEEE/ACM 42nd international conference on software engineering: software engineering in practice (ICSE-SEIP)

Bettenburg N, Premraj R, Zimmermann T, Kim S (2008) Extracting structural information from bug reports. In: Proceedings of the 2008 international working conference on Mining software repositories

Bhargava P, Spasojevic N, Ellinger S, Rao A, Menon A, Fuhrmann S, Hu G (2019) Learning to map wikidata entities to predefined topics. In: Companion proceedings of the 2019 World Wide Web conference, pp 1194–1202

Bortis G, Van Der Hoek A (2013) Porchlight: A tag-based approach to bug triaging. In: 2013 35th international conference on software engineering (ICSE). IEEE, pp 342–351

Bouma G (2009) Normalized (pointwise) mutual information in collocation extraction. Proceedings of GSCL, pp 31–40

Caruana R (1997) Multitask learning. Mach Learn 28(1):41–75

Chen J, He X, Lin Q, Xu Y, Zhang H, Hao D, Gao F, Xu Z, Dang Y, Zhang D (2019a) An empirical investigation of incident triage for online service systems. In: 2019 IEEE/ACM 41st international conference on software engineering: software engineering in practice (ICSE-SEIP), pp 111–120

Chen J, He X, Lin Q, Zhang H, Hao D, Gao F, Xu Z, Dang Y, Zhang D (2019b) Continuous incident triage for large-scale online service systems. In: 2019 34th IEEE/ACM international conference on automated software engineering (ASE), pp 364–375

Chen P, Sun Z, Bing L, Yang W (2017) Recurrent attention network on memory for aspect sentiment analysis. In: Proceedings of the 2017 conference on empirical methods in natural language processing, pp 452–461

Chen Y, Yang X, Lin Q, Zhang H, Gao F, Xu Z, Dang Y, Zhang D, Dong H, Xu Y et al (2019) Outage prediction and diagnosis for cloud service systems. In: The World Wide Web conference, pp 2659–2665

Chiu JPC, Nichols E (2016) Named entity recognition with bidirectional lstm-cnns. Transactions of the Association for Computational Linguistics 4:357–370

Church KW, Hanks P (1990) Word association norms, mutual information, and lexicography. Computational Linguistics 16(1):22–29. [Online] Available: https://www.aclweb.org/anthology/J90-1003

Collobert R, Weston J, Bottou L, Karlen M, Kavukcuoglu K, Kuksa P (2011) Natural language processing (almost) from scratch. J Mach Learn Res 12(Aug):2493–2537

Dang Y, Lin Q, Huang P (2019) Aiops: real-world challenges and research innovations. In: 2019 IEEE/ACM 41st international conference on software engineering: Companion proceedings (ICSE-Companion). IEEE, pp 4–5

Devlin J, Chang M-W, Lee K, Toutanova K (2018) Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805
Ester M, Kriegel H-P, Sander J, Xu X et al (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. In: Kdd, vol 96, pp 226–231
Fano RM (1961) Transmission of information: A statistical theory of communications. Am J Phys 29(11):793–794
Finin T, Murnane W, Karandikar A, Keller N, Martineau J, Dredze M (2010) Annotating named entities in twitter data with crowdsourcing. In: Proceedings of the NAACL HLT 2010 workshop on creating speech and language data with amazons mechanical turk, pp 80–88
Graves A, Schmidhuber J (2005) Framewise phoneme classification with bidirectional lstm and other neural network architectures. Neural Networks 18(5-6):602–610
Greenberg N, Bansal T, Verga P, McCallum A (2018) Marginal likelihood training of bilstm-crf for biomedical named entity recognition from disjoint label sets. In: Proceedings of the 2018 conference on empirical methods in natural language processing, pp 2824–2829
Hashimoto K, Stenetorp P, Miwa M, Tsuruoka Y (2015) Task-oriented learning of word embeddings for semantic relation classification. arXiv preprint arXiv:1503.00095
Hendrickx I, Kim SN, Kozareva Z, Nakov P, Séaghdha DO, Padó S, Pennacchiotti M, Romano L, Szpakowicz S (2019) Semeval-2010 task 8: Multi-way classification of semantic relations between pairs of nominals. arXiv preprint arXiv:1911.10422
Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780
Huang Z, Xu W, Yu K (2015) Bidirectional lstm-crf models for sequence tagging. arXiv preprint arXiv:1508.01991
Kim H, Sun Y, Hockenmaier J, Han J (2012) Etm: Entity topic models for mining documents associated with entities. In: 2012 IEEE 12th international conference on data mining. IEEE, pp 349–358
Kingma DP, Ba J (2015) Adam: A method for stochastic optimization. In: Bengio Y, LeCun Y (eds) 3rd international conference on learning representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings. [Online]. Available: arXiv:1412.6980
Kulkarni C, Xu W, Ritter A, Machiraju R (2018) An annotated corpus for machine reading of instructions in wet lab protocols. arXiv preprint arXiv:1805.00195
Kumar R, Bansal C, Maddila C, Sharma N, Martelock S, Bhargava R (2019) Building sankie: An ai platform for devops. In: Proceedings of the 1st international workshop on bots in software engineering, ser. BotSE'19. IEEE Press, p 4853
Lafferty J, McCallum A, Pereira FC (2001) Conditional random fields: Probabilistic models for segmenting and labeling sequence data
Lample G, Ballesteros M, Subramanian S, Kawakami K, Dyer C (2016) Neural architectures for named entity recognition. arXiv preprint arXiv:1603.01360
Li Q, Li T, Chang B (2016) Discourse parsing with attention-based hierarchical neural networks. In: Proceedings of the 2016 conference on empirical methods in natural language processing, pp 362–371
Limsopatham N, Collier N (2016) Bidirectional lstm for named entity recognition in twitter messages
Luo C, Lou J-G, Lin Q, Fu Q, Ding R, Zhang D, Wang Z (2014) Correlating events with time series for incident diagnosis. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, pp 1583–1592
McDonald R, Pereira F, Kulick S, Winters S, Jin Y, White P (2005) Simple algorithms for complex relation extraction with applications to biomedical ie. In: Proceedings of the 43rd annual meeting of the association for computational linguistics (ACL'05), pp 491–498
Mehta S, Bhagwan R, Kumar R, Bansal C, Maddila C, Ashok B, Asthana S, Bird C, Kumar A (2020) Rex: Preventing bugs and misconfiguration in large services using correlated change analysis. In: 17th {USENIX} symposium on networked systems design and implementation ({NSDI} 20), pp 435–448
Mikolov T, Karafiát M, Burget L, Černockỳ J, Khudanpur S (2010) Recurrent neural network based language model. In: Eleventh annual conference of the international speech communication association
Nadeau D, Sekine S (2007) A survey of named entity recognition and classification. Lingvisticae Investigationes 30(1):3–26
Nair V, Raul A, Khanduja S, Bahirwani V, Shao Q, Sellamanickam S, Keerthi S, Herbert S, Dhulipalla S (2015) Learning a hierarchical monitoring system for detecting and diagnosing service issues. In: Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining, pp 2029–2038
Newman D, Chemudugunta C, Smyth P (2006) Statistical entity-topic models. In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, pp 680–686

Pantel P, Lin T, Gamon M (2012) Mining entity types from query logs via user intent modeling. In: Proceedings of the 50th annual meeting of the association for computational linguistics: long papers-Volume 1, Association for Computational Linguistics, pp 563–571

Pawar S, Palshikar GK, Bhattacharyya P (2017) Relation extraction: A survey. arXiv preprint arXiv:1712.05191

Pennington J, Socher R, Manning CD (2014) Glove: Global vectors for word representation. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)

Rao N, Bansal C, Guan J (2020) Code search intent classification using weak supervision. arXiv preprint arXiv:2011.11950

Ratner A, Bach SH, Ehrenberg H, Fries J, Wu S, Ré C (2017) Snorkel: Rapid training data creation with weak supervision. In: Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases, vol 11, no 3. NIH Public Access, pp 269. NIH Public Access

Ritter A, Clark S, Etzioni O et al (2011) Named entity recognition in tweets: An experimental study. In: Proceedings of the 2011 conference on empirical methods in natural language processing, pp 1524–1534

Shetty M, Bansal C, Kumar S, Rao N, Nagappan N, Zimmermann T (2021) Neural knowledge extraction from cloud service incidents. In: 2021 IEEE/ACM 43rd international conference on software engineering: software engineering in practice (ICSE-SEIP), pp 218–227

Thanopoulos A, Fakotakis N, Kokkinakis G (2002) Comparative evaluation of collocation extraction metrics. In: LREC, vol 2. Citeseer, pp 620–625

Tian Y, Wijedasa D, Lo D, Le Goues C (2016) Learning to rank for bug report assignee recommendation. In: 2016 IEEE 24th international conference on program comprehension (ICPC). IEEE, pp 1–10

Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser L, Polosukhin I (2017) Attention is all you need. In: Advances in neural information processing systems, pp 5998–6008

Wang S, Zhang W, Wang Q (2014) Fixercache: Unsupervised caching active developers for diverse bug triage. In: Proceedings of the 8th ACM/IEEE international symposium on empirical software engineering and measurement, pp 1–10

Xu Y, Ding F, Wang B (2008) Entity-based query reformulation using wikipedia. In: Proceedings of the 17th ACM conference on Information and knowledge management, pp 1441–1442

Ye D, Xing Z, Foo CY, Ang ZQ, Li J, Kapre N (2016) Software-specific named entity recognition in software engineering social content. In: 2016 IEEE 23rd international conference on software analysis, evolution, and reengineering (SANER), vol 1. IEEE, pp 90–101

Zelenko D, Aone C, Richardella A (2003) Kernel methods for relation extraction. J Mach Learn Res 3(Feb):1083–1106

Zhou Y, Tong Y, Gu R, Gall H (2016) Combining text mining and data mining for bug report classification. Journal of Software: Evolution and Process 28(3):150–176

Manish Shetty is a Research Fellow at Microsoft Research (India) working on intelligent tools and systems to aid software engineering. His research interests are in the intersection of machine learning, software engineering, and programming systems. His research concerns combining empirical, neural, and symbolic techniques to study and improve software engineering processes. He is an incoming Ph.D. student in Computer Science at UC Berkeley.

Chetan Bansal is a Principal Research Manager at Microsoft working in the area of AI assisted Software Engineering and Cloud Intelligence. His work has been focused on improving software engineering tools and infrastructure by leveraging Machine Learning and data-driven techniques. He has published several papers on topics related to Software Engineering, Artificial Intelligence and Information Retrieval. He serves on the editorial boards for Journal of Systems and Software and Information & Software Technology journals and has served on PCs for several conferences such as ICSE, MSR and FSE.

Sumit Kumar received his B.Tech in Electrical Engineering from Indian Institute of Technology, Kanpur and his M.S. in Computer Science from University of Wisconsin, Madison. He is a technology generalist with a software career spanning healthcare information systems, devices & smart phone software stacks, Distributed Systems, AIOps, MLOps, brief stint at entrepreneurship and lately the exciting new world of fintech & Web3.0.

Nikitha Rao is a Ph.D. student at Carnegie Mellon University where she works on AI for Software Engineering with Prof. Vincent Hellendoorn. Her work mainly focuses on improving programmer productivity by developing novel algorithms and tools using data science and machine learning techniques that can be used in the software development lifecycle. Prior to this, she spent two years as a Research Fellow at Microsoft Research (India) working on various problems in data science, information retrieval and machine learning for software engineering.

Nachiappan Nagappan was a Partner Researcher at Microsoft Research when this work was performed. He also holds an adjunct faculty appointment at IIIT New Delhi. His research interests are in the field of Data Analytics and Machine Learning for Software Engineering focusing on Analytics for Empirical Software Engineering, Software Reliability, Software Metrics, and Developer Productivity. He is a Fellow of the IEEE and ACM.

Affiliations

Manish Shetty[1] · Chetan Bansal[2] · Sumit Kumar[3] · Nikitha Rao[1] · Nachiappan Nagappan[2]

Chetan Bansal
chetanb@microsoft.com

Sumit Kumar
sumitku@gmail.com

Nikitha Rao
nikithar@andrew.cmu.edu

Nachiappan Nagappan
nnagappan@acm.org

1 Microsoft Research, "Vigyan", No. 9, Lavelle Road, Bengaluru, Karnataka, 560001, India

2 Microsoft, Research Microsoft, One Microsoft Way, Redmond, WA, 98052, USA

3 Microsoft, Microsoft, One Microsoft Way, Redmond, WA, 98052, USA