Constructing Flow Graphs from Procedural Cybersecurity Texts

Kuntal Kumar Pal[*], **Kazuaki Kashihara**[*], **Pratyay Banerjee**[*],
Swaroop Mishra, Ruoyu Wang, Chitta Baral
School of Computing, Informatics, and Decision Systems Engineering,
Arizona State University,
{kkpal, kkashiha, pbanerj6, srmishr1, fishw, chitta}@asu.edu

Abstract

Following procedural texts written in natural languages is challenging. We must read the whole text to identify the relevant information or identify the instruction-flow to complete a task, which is prone to failures. If such texts are structured, we can readily visualize instruction-flows, reason or infer a particular step, or even build automated systems to help novice agents achieve a goal. However, this structure recovery task is a challenge because of such texts' diverse nature. This paper proposes to identify relevant information from such texts and generate information flows between sentences. We built a large annotated procedural text dataset (CTFW) in the cybersecurity domain (3154 documents). This dataset contains valuable instructions regarding software vulnerability analysis experiences. We performed extensive experiments on CTFW with our LM-GNN model variants in multiple settings. To show the generalizability of both this task and our method, we also experimented with procedural texts from two other domains (Maintenance Manual and Cooking), which are substantially different from cybersecurity. Our experiments show that Graph Convolution Network with BERT sentence embeddings outperforms BERT in all three domains.

1 Introduction

Many texts in the real-world contain valuable instructions. These instructions define individual steps of a process and help users achieve a goal (and corresponding sub-goals). Documents including such instructions are called *procedural texts*, ranging from simple cooking recipes to complex instruction manuals. Additionally, *discussion in a shared forum or social media platform, teaching books, medical notes, sets of advice about social behavior, directions for use, do-it-yourself notices, itinerary guides can all be considered as procedural texts* (Delpech and Saint-Dizier, 2008). Most of these texts are in the form of natural languages and thus, lacking structures. We define *structure* as sentence-level dependencies that lead to a goal. These dependencies can vary based on the text-domain. Some examples of such dependencies are action traces, effects of an action, information leading to the action, and instruction order. Constructing structured flow graphs out of procedural texts is the foundation for natural language understanding and summarization, question-answering (QA) beyond factoid QA, automated workflow visualization, and the recovery of causal relationships between two statements. By flow-graph we mean both information and action flows in a text. However, the lack of structures in such texts makes them challenging to follow, visualize, extract inferences, or track states of an object or a sub-task, which ultimately makes constructing their flow graphs an insurmountable task.

Procedural texts are common in cybersecurity, where security analysts document how to discover, exploit, and mitigate security vulnerabilities in articles, blog posts, and technical reports, which are usually referred to as *security write-ups*. Practitioners in cybersecurity often use write-ups as educational and researching materials. Constructing structured flow graphs from security write-ups may help with automated vulnerability discovery and mitigation, exploit generation, and security education in general. However, automatically analyzing and extracting information from security write-ups are extremely difficult since they lack structure.

Figure 1 illustrates the core of a security write-up (broken into sentences) that carries instructions for exploiting a vulnerability in an online shopping service. S_1, S_3, and S_4 are the author's observations about the service's nature. Based on this information, S_5 and S_6 are two possible paths of actions.

[*] These authors contributed equally to this work.

Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021, pages 3945–3957

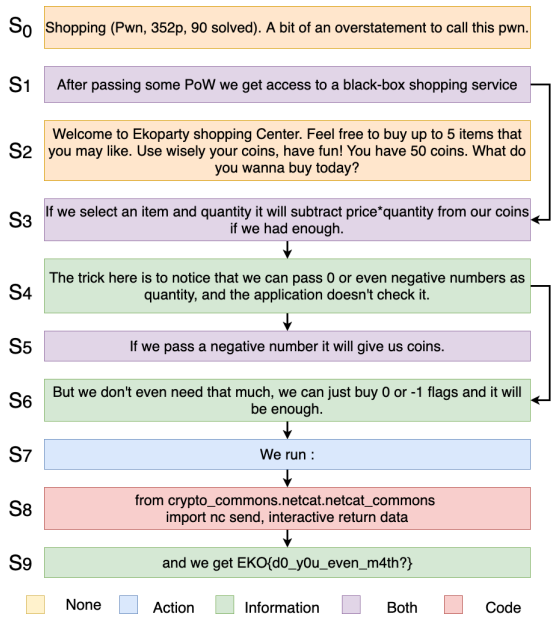

Figure 1: An example flow graph from the CTFW. Sentences in S_2 are merged into one block for clarity.

The author chose S_6 and ran the Python code in S_8 to exploit the service. S_0 and S_2 are irrelevant for the author's goal of exploiting this service.

Here we propose a novel approach to extract action paths out of structure-less, natural language texts by identifying actions and information flows embedded in and between sentences and constructing action flow graphs. Specifically, our focus is on procedural texts in the cybersecurity domain. We also show that constructing flow graphs helps extract paths of actions in domains besides cybersecurity, such as cooking and maintenance manuals.

Most previous works (Mori et al., 2014; Kiddon et al., 2015; Malmaud et al., 2014; Maeta et al., 2015; Xu et al., 2020; Mysore et al., 2019; Song et al., 2011) focus on fine-grained knowledge extraction from procedural texts in diverse domains. There are also a handful of works (Delpech and Saint-Dizier, 2008; Fontan and Saint-Dizier, 2008; Jermsurawong and Habash, 2015) that study the structure of natural language texts. Different from previous works, we extract structures and construct flow graphs from natural texts at the sentence level. This is because fine-grained domain-entity extraction tasks require a large amount of annotated data from people with specific in-depth domain knowledge, whereas text structures can be generalized.

Dataset. We built a dataset from security write-ups that are generated from past Capture The Flag competitions (CTFs). CTFs are computer security competitions that are usually open to everyone in the world. Players are expected to find and exploit security vulnerabilities in a given set of software services, and through exploiting vulnerabilities, obtain a *flag*—a unique string indicating a successful attempt—for each exploited service. Once the game is over, many players publish security write-ups that detail how they exploited services during the game. While these write-ups are a valuable educational resource for students and security professionals, they are usually unstructured and lacking in clarity. We collected 3617 CTF write-ups from the Internet, created a procedural text dataset, and invited domain experts to label each sentence for the purpose of constructing flow graphs and identifying action paths. To the best of our knowledge, this is the first attempt to use the knowledge embedded in security write-ups for automated analysis. The data and the code is publicly available [1] for future research.

This paper makes the following contributions:

- We built a new procedural text dataset, CTFW, in the cybersecurity domain. To the best of our knowledge, CTFW is the first dataset that contains valuable information regarding vulnerability analysis from CTF write-ups.
- We proposed a new NLU task of generating flow graphs from natural language procedural texts at the sentence level without identifying fine-grained named entities.
- We proposed four variations of a graph neural network-based model (LM-GNN) to learn neighbor-aware representation of each sentence in a procedural text and predict the presence of edges between any pair of sentences.
- We evaluated our models on CTFW. To the best of our knowledge, this is the first attempt in automated extraction of information from security write-ups. We also evaluated our models across three datasets in different domains and showed the generalizability of our approach.

2 Our Approach

We map each sentence of a procedural text as a node in a graph, and the action or information flows as edges. The task is then simplified into an edge prediction task: Given a pair of nodes, find if there is an edge between them. We learn feature representations of nodes using language models like BERT/RoBERTa (Devlin et al., 2018; Liu et al.,

[1] https://github.com/kuntalkumarpal/FlowGraph

Dataset Statistics	COR	MAM	CTFW
# Documents	297	575	3154
Avg size of document	9.52	8.12	17.11
Avg length of sentence	65.46	34.81	92.87
# Edges ($\|e^+\|$)	2670	5043	54539
$\|e^+\| : (\|e^+\| + \|e^-\|)$	0.18	0.12	0.07
Avg degree of node	1.83	1.76	1.88

Table 1: Dataset Statistics. $|e^+|$ is the total number of actual edges, and $|e^+| + |e^-|$ is the total number of edges possible. The in-degree of the starting node and out-degree of the end node are both 0.

2019). Then, to make the nodes aware of their neighboring sentences, we use Graph Neural Network (GNN) to update the node representations. We check for the edge between every pair of nodes in a graph and reduce the task to a binary classification during inference. This formulation enables us to predict any kind of structure from a document.

3 Dataset Creation

In this section, we present how we created three datasets on which we evaluated our approach. Table 1 shows the statistics for each datasets used.

3.1 CTF Write-ups Dataset (CTFW)

Each CTF competition has multiple challenges or tasks. Each task may have multiple write-ups by different authors. We crawled 3617 such write-ups from GitHub and CTFTime (CTFTime). Write-ups are unique and diverse but have common inherent principles. For each write-up, we provide two kinds of annotations: *sentence type* and *flow structure*. The writing style is informal with embedded code snippets and often contains irrelevant information.

Part of the annotations were provided as an optional, extra-credit assignment for the Information Assurance course. These CTF write-ups were directly related to the course-content, where students were required to read existing CTF write-ups and write write-ups for other security challenges they worked on during the course. Then students were given the option of voluntarily annotating CTF write-ups they read for extra credits in the course. For this task, we followed all the existing annotation guidelines and practices. We also ensured that (1) The volunteers were aware of the fact that their annotations would be used for a research project (2) They were aware that no PII was involved or would be used in the research project (3) They were aware that extra credits were entirely optional, and they could refrain from submitting at any point of time without any consequences (4) Each volunteer was assigned only 10-15 write-ups based on a pilot study we did ahead of time, annotating an average-length CTF write-up took about two minutes (maximum ten mins).

Remaining annotations were performed by the Teaching Assistants (TA) of the course. These annotations were done as part of the course preparation process, which was part of their work contract. All the TAs were paid bi-weekly compensation by the university or by research funding. It was also ensured that the TAs knew these annotations would be used for a research project, their PII was not involved and annotations were to be anonymized before using. We verified the annotations by randomly selecting write-ups from the set. Figure 1 shows a sample annotation.

Sentence Type Annotations. We split the documents into sentences using natural language rules. We then ask the volunteers to annotate the type of each statement as either Action (A), Information (I), Both (A/I), Codes (C), or irrelevant (None). *Action* sentences are those where the author specifies actions taken by them, whereas, *Information* statements mention observations of the author, the reasons and effects of their action. Sentences containing *codes* are assigned as C, and those which can be considered as both information and actions are marked as *Both* (A/I).

Flow structure Annotations. The second level of annotations is regarding the write-up structure. Each volunteer is given a csv file for each document with a set of sentence IDs and text for each write-up. They are asked to annotate the flow of information in the document by annotating the sentence id of some next possible sentences, which indicate the flow. We filter those write-ups which are irrelevant and those which did not have much detail (single line of valuable information). We call a write-up as irrelevant if it has no action-information annotations or if it has direct codes without any natural language description of steps to detect vulnerabilities. We only keep write-ups written in the English language for this work. Finally, we have 3154 write-ups with *sentence type* and *structure annotations*.

CTFTime website states that the write-ups are copyrighted by the authors who posted them and it is practically impossible to contact each author.

Such data is also allowed to use for academic research purposes(usc; euc). Thus, we follow the previous work using data from CTFTime (Švábenský et al., 2021), and share only the urls of those write-ups which we use. We do not provide the scraper script since it would create a local copy of the write-up files unauthorized by the users. Interested readers can replicate the simple scraper script from the instructions in Appendix A and use it after reviewing the conditions under which it is permissible to use. We, however, share our annotations for those write-up files.

3.2 Cooking Recipe Flow Corpus (COR)

This corpus (Yamakata et al., 2020) provides 300 recipes with annotated recipe named entities and fine-grained interactions between each entity and their sequencing steps. Since we attempt to generate action flow graphs without explicitly identifying each named entity, we aggregate the fine-grained interactions between recipe named entities to generate sentence-level flows for each recipe. We reject three single-sentence recipes.

3.3 Maintenance Manuals Dataset (MAM)

This dataset (Qian et al., 2020) provides multi-grained process model extraction corpora for the task of extracting process models from texts. It has over 160 Maintenance Manuals. Each manual has fine-grained interactions between each entity and its sequencing steps. We use the annotations from sentence-level classification data and semantic recognition data for generating annotations of sentence-level flows for each process. Here also, we reject single sentence processes.

4 Model Description

Our goal is to find paths or traces of actions or information between texts. This needs an understanding of each sentence's interconnection. Hence, we modeled the problem into an edge prediction task in a graph using GNNs. We represent each sentence as a node and directed edges as information flows. Since this is procedural text (unidirectional nature) of instructions, we consider only the directed edges from one sentence S_n to any of its next sentences S_{n+i}. The node representations are learned using language models (LM) and GNNs.

4.1 Document to Sentence Pre-processing

Given a natural language document, first we split the document into sentences based on simple rules and heuristics. COR and MAM datasets already have document split into separate sentences. In the flow graph creation task, we filter out irrelevant sentences for the CTFW dataset based on the sentence type annotations. After this pre-processing task, each document (D_i) is converted into a series of sentences (S_j) where n is the number of valid sentences in a document.

$$D_i = \{S_0, S_1, S_2 ... S_{n-1}\}$$

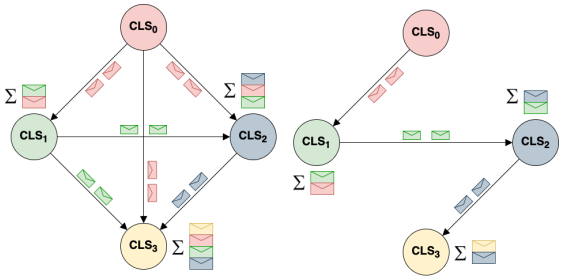

Figure 2: *Node Representation Learning for a document with four sentences in single-layer GNN*. Left: *Semi-Complete* Structure, Right: *Linear* Structure. During training, the sentence representation (CLS_i) are enriched using appropriate message passing techniques from the connected 1-hop neighbors.

4.2 Document to Graph Representation

A graph ($G = (V, E)$) is formally represented as a set of nodes ($V = \{v_0, v_1, ..\}$) connected by edges ($E = \{e_0, e_1, ..\}$ where $e_i = \{v_m, v_n\}$). We consider the sentences (S_j) of any document (D_i) as nodes of a directed graph (G_i). We experiment with two graph structure types for learning better node representation using GNN. First, we form local windows (W_N, where $N = 3, 4, 5, all$ sentences) for each sentence and allow the model to learn from all of the previous sentences in that window. We form the document graph by connecting each sentence with every other sentence in that window, with directed edges only from S_i to S_j where $i < j$. We do this since procedural languages are directional. We call this configuration *Semi-Complete*. Second, we consider connecting the nodes linearly where every S_i is connected to S_{i+1} except the last node. We call this *Linear* setting. Figure 2 shows the settings. We use LMs like BERT and RoBERTa to generate initial sentence representations. For each sentence (S_i), we extract the pooled sentence representation (CLS_{S_i}) of contextual BERT/RoBERTa embeddings (h_{S_i}).

We use CLS_{S_i} as node features for the graph (G_i).

$$h_{S_i} = BERT\,(\,[CLS]s_0s_1...s_{n-1}\,[SEP]\,)$$

4.3 Neighbor Aware Node Feature Learning

Since the LM sentence vectors are generated individually for each sentence in the document, they are not aware of other local sentences. So, through the *semi-complete* graph connection, the model can learn a global understanding of the document. However, the *linear* connection helps it learn better node representation conditioned selectively on its predecessor. We call the connected nodes as the neighbor nodes. We use Graph Convolutional Network (GCN) (Kipf and Welling, 2016) and Graph Attention Network (GAT) (Veličković et al., 2017) to aggregate the neighbor information for each node following the generic graph learning function (1)

$$\mathbf{H}^{l+1} = f(\mathbf{H}^l, \mathbf{A}) \tag{1}$$

where $\mathbf{A}$ is the adjacency matrix of the graph, $\mathbf{H}^l$ and $\mathbf{H}^{(l+1)}$ are the node representations at lth and $(l+1)$th layer of the network and f is the message aggregation function. In GCN, each node i, aggregates the representations of all of its neighbors $N(i)$ based on $\mathbf{A}$ and itself at layer l and computes the enriched representation $\mathbf{h}_i^{l+1}$ based on the weight matrix $\mathbf{\Theta}$ of the layer normalized by degrees of source $d(i)$ and its connected node $d(j)$ as per (2). In GAT, messages are aggregated based on multi-headed attention weights (α) learned from the neighbor node representations $\mathbf{h}_j^l$ following (3).

$$\mathbf{h}_i^{l+1} = \mathbf{\Theta} \sum_{j \in N(i) \cup \{i\}} \frac{1}{\sqrt{d(i)d(j)}} \mathbf{h}_j^l \tag{2}$$

$$\mathbf{h}_i^{l+1} = \alpha_{ii}\mathbf{\Theta}\mathbf{h}_i^l \quad + \sum_{j \in N(i)} \alpha_{ij}\mathbf{\Theta}\mathbf{h}_j^l \tag{3}$$

4.4 Projection

We concatenate the neighbor aware node representations of each pair of nodes ($\mathbf{h}_i$;$\mathbf{h}_j$) from a graph and pass it through two projection layers with a GELU (Hendrycks and Gimpel, 2016) non-linearity in between. We use the same non-linearity functions used by the BERT layers for consistency. We steadily decrease the parameters of each projection layer by half. During testing, given a document, we are unaware of which two sentences are connected. So, we compare each pair of nodes. This leads to an unbalanced number of existing (1) and non-existing (0) edge labels. Hence, we use weighted cross-entropy loss function as in equation (4) and (5), where L is the weighted cross-entropy loss, w_c is the weight for class c, i is the data in each mini-batch.

$$L(x, c) = w_c\Big(- x_c + log\Big(\sum_j exp(x_j)\Big)\Big) \tag{4}$$

$$L = \frac{\sum_{i=1}^{N} L(i, c_i)}{\sum_{i=1}^{N} w_{c_i}} \tag{5}$$

4.5 Training and Inference

Our training data comprises a set of sentences and the connections as an adjacency matrix for each document. Batching is done based on the number of graphs. GCN/GAT updates the sentence representations. A pair of node representations are assigned a label of 1 if there is an edge between them; otherwise, we assign them 0. Thus, we model it as a binary classification task as in equation (6) where f is the projection function, g is the softmax function, and y is the binary class output. Depending on the weighted cross-entropy loss, the node representations get updated after each epoch. During inference, the model generates node representations of each sentence in a test document, and we predict whether an edge exists between any two nodes in a given document graph.

$$y_c = \arg\max_k g(f(\mathbf{h}_i; \mathbf{h}_j), k) \quad c \in \{0, 1\} \tag{6}$$

5 Experiments

Datasets and Tasks: Each dataset is split into train, validation, and test sets in 70:10:20 ratio. The first task is identifying relevant information from raw CTF write-ups by classifying the type of each sentence. The second task is identifying information flows between sentences by predicting edges between sentence pairs, if any.

Metrics: We use *accuracy* as the evaluation metric for the Sentence Type classification task on CTFW. For the second task, because of the label imbalance we compare based on the *area under Precision-Recall curve* (PRAUC) and also report the corresponding *F1-score*. Hence do not report area under the ROC curve or accuracy.

We consider four settings for this task. The *no window* setting (W_{all}) checks whether there is an edge between any two statements in the given document. The comparisons required in this setting are directly proportional to the document's size. In CTFW, the size of each write-up is quite large. So,

	Models	CTFW		COR		MAM	
		PRAUC	F1	PRAUC	F1	PRAUC	F1
Baselines	Random	-	50.49	-	42.78	-	47.82
	Weighted Random	-	37.81	-	39.13	-	44.10
	BERT-NS	0.5751	26.12	0.5638	43.14	0.5873	29.73
	RoBERTa-NS	0.5968	32.44	0.5244	42.99	0.6236	39.65
Ours	BERT-GCN	0.7075	69.26	**0.6312**	58.13	**0.6888**	63.75
	RoBERTa-GCN	**0.7221**	69.04	0.6233	61.44	0.6802	65.73
	BERT-GAT	0.5585	61.93	0.4553	41.93	0.4568	62.18
	RoBERTa-GAT	0.5692	64.51	0.4358	24.74	0.4585	59.55

Table 2: Comparison with Baselines on Best Test Area under Precision-Recall Curve (PRAUC) and its corresponding F1 for **CTFW** (CTFwrite-up), **COR** (Cooking), **MAM** (Maintenance) datasets. NS is the next sentence based prediction approach. Our best model performance is bold, while maximum baseline performance is underlined.

to reduce unnecessary comparisons, we apply simple heuristics that instructions in procedural text, in general, does not have longer *direct* dependencies. Thus, using the windows, we can control each sentence's number of comparisons (node). To understand how the performances change we evaluate with a *sliding windows of N sentences* (W_N) where $N = 3, 4, 5$. The comparisons are only made with the next N sentences from a given sentence. For example, in case of W_5, for first sentence (S_1) we check for edges with S_2, S_3, S_4, S_5, S_6 and not S_7 on-wards. However, to have a fair comparison, we keep labeled out-of-window gold edges, if any. The ratios of existing and total edges in CTFW are 0.07 (W_{all}), 0.24 (W_5), 0.29 (W_4), 0.38 (W_3).

Training: We use Pytorch Geometric (Fey and Lenssen, 2019) for GNN and transformers (Wolf et al., 2020) for LM implementations. Training is done with AdamW (Loshchilov and Hutter, 2017) optimizer along with linear warmup scheduler on 4 Tesla V100 16GB GPUs. We use bert-base-uncased, bert-large-uncased, roberta-base and roberta-large versions as base model. We store the model with the best PRAUC score. Batch size of {4,8,16} and learning rates of {1e-5,5e-6} are used. Maximum sequence length varies between {64, 80, 128}. GNN depths are kept 128 (layer 1) and 64 (layer 2). We use a dropout of 0.4 in selected layers. For GAT, we keep four attention heads in layer 1. Details are present in Appendix C.

6 Results and Discussion

6.1 Sentence Type Classification (STC)

We use large and base versions of BERT and RoBERTa for this task to predict the type of sentences in a given text to establish a baseline for this task. This task helps to identify relevant and irrelevant sentences in a document. Each sentence is classified into any of Action, Information, Both, Code, and None. These fine-grained annotations can be used in later works for creating automated agents for vulnerability analysis. The processed data consists of 120751 samples for training, 17331 for validation, and 34263 for testing. Table 3 shows that RoBERTa-large performs better than the rest.

Model	**Val**	**Test**
BERT-Base	78.48±0.25	77.42±0.10
BERT-Large	78.19±0.48	77.13±0.20
RoBERTa-Base	78.85±0.25	77.37±0.11
RoBERTa-Large	79.02±0.16	**77.66±0.12**

Table 3: Sentence Type Classification (Mean Accuracy from three seed values). Best performance in bold.

6.2 Flow Structure Prediction

Here we present the performance results for the flow structure prediction.

Random Baseline: In the *Random* baseline, for every pair of nodes in each document we randomly select 0 (no-edge) or 1 (edge). For *Weighted Random* baseline, we choose randomly, based on the percentage of edge present in the train set. We only report F1 since there is no probability calculation.

Next Sentence-based Prediction (NS) Baseline: We use LMs like BERT and RoBERTa in a next sentence prediction setting to get the baselines. Each pair of sentences is concatenated using [SEP] token and passed through these language models. Using the pooled LM representation, we classify whether an edge exists between them or not. We show maximum PRAUC and its corresponding F1 for each dataset from the results of each of our window settings (W_3, W_4, W_5, W_{all}).

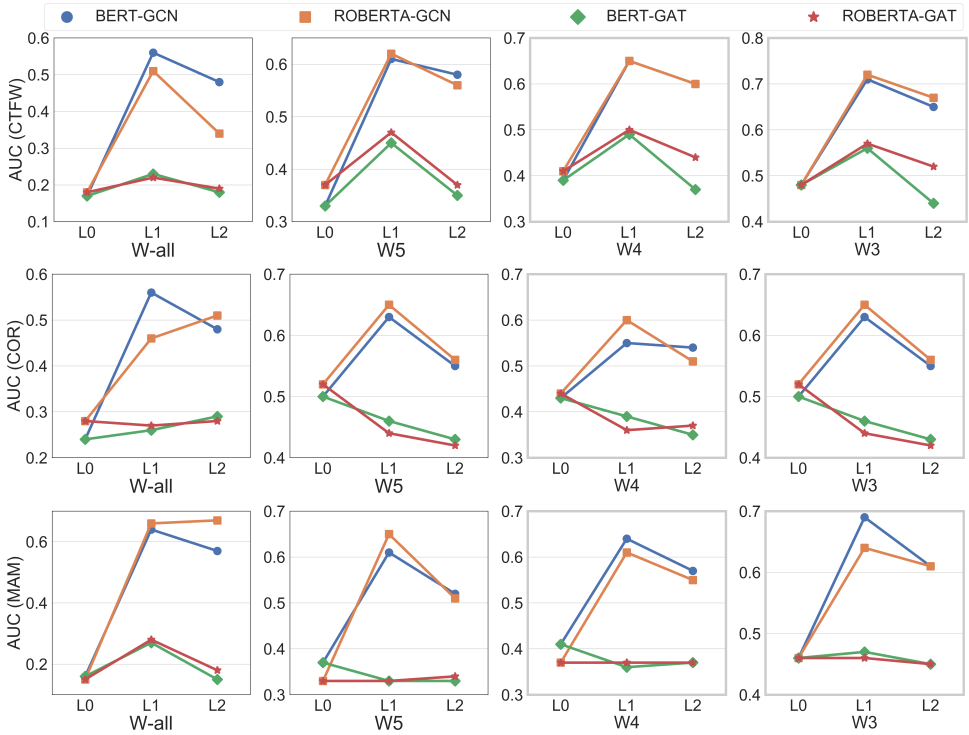

Figure 3: Effect of GNN Layers (L_0, L_1, L_2) on performance (PRAUC) of the models for W_{all}, W_5, W_4, W_3 settings on the three datasets

Models: We compare four variants of our LM-GNN models both with baseline and among each other in Table 2. The scores are overall best scores across single and double layers GNN (GCN/GAT) and LM (BERT/RoBERTa) after experiments with both base and large version, trained with pre-trained and randomly initialized weights.

We see that the best LM-GCN models outperform the best baseline model by 0.12, 0.07, 0.06 in PRAUC for CTFW, COR, and MAM datasets, respectively. However, the best LM-GAT scores falls short of the baselines indicating that the graph attentions on LM *sentence representations* cannot learn robust representation to perform this edge prediction task. Another thing to notice here is that, the best BERT-GCN models perform better than RoBERTa-GCN for COR and MAM datasets while performs poorly in the CTFW dataset. We hypothesize that this is because, the CTFW dataset has ten times more data than COR and six times more than MAM, which helps the RoBERTa model correctly predict the edges.

6.3 Analysis

Effect of Graph Connection Type: Table 4 shows how the models behave with semi-complete (SC) and linear (L) graph connection. For each dataset, we compare the PRAUC results for each window to draw more granular insight on the effect of neighbor aware representation learning. When we restrict graph learning by creating small windows (W_3, W_4, W_5), the linear model works better because of its selective learning conditioned on its predecessor. On the other hand, the semi-complete connection helps to learn a global awareness and works best in the W_{all} setting. It is important to note that each model performs better than the average PRAUC performance, which is the percentage of edges in the data indicating that the model is

	W_3	W_4	W_5	W_{all}
CTFW-SC	0.6630	0.5985	0.5733	**0.5590**
CTFW-L	**0.7221**	**0.6520**	**0.6150**	0.3962
CTFW-EP	0.3700	0.2900	0.2400	0.0700
COR-SC	0.5639	0.5129	0.4731	**0.5580**
COR-L	**0.6456**	**0.6012**	**0.5274**	0.4034
COR-EP	0.3700	0.3100	0.2600	0.1700
MAM-SC	0.6528	0.6219	0.6091	**0.6718**
MAM-L	**0.6888**	**0.6362**	**0.6137**	0.4161
MAM-EP	0.4500	0.3700	0.3200	0.1500

Table 4: Effect of Semi-Complete(SC) and Linear(L) Graph Connection on 3 datasets in Area under Precision-Recall Curve (PRAUC). We also keep edge-percent (EP) in four window settings for comparison.

able to learn using the graph connections.

Effect of Graph Layers: We study how the depth of the GNNs affects the performance. We compare PRAUC across all four variations of the model in No-Window (W_{all}), W_5, W_4, W_3 settings in Figure 3. We experimented with no (L_0), single (L_1) and double (L_2) GNN layers. In all three datasets, we find the performance improves when we use a single layer and degrades beyond that for each of the windows with GCN based models. We do not go beyond two layers because of this observation and the graph connection types we use. We believe the reason for this drop (0.03-0.08 PRAUC) is that information from 2-hop neighbors might hinder the learning of the current node and confuse the model to predict wrongly. The GAT-based models mostly remain unaffected with the graph layers for both COR and MAM while showing some improvement in CTFW for one layer setting.

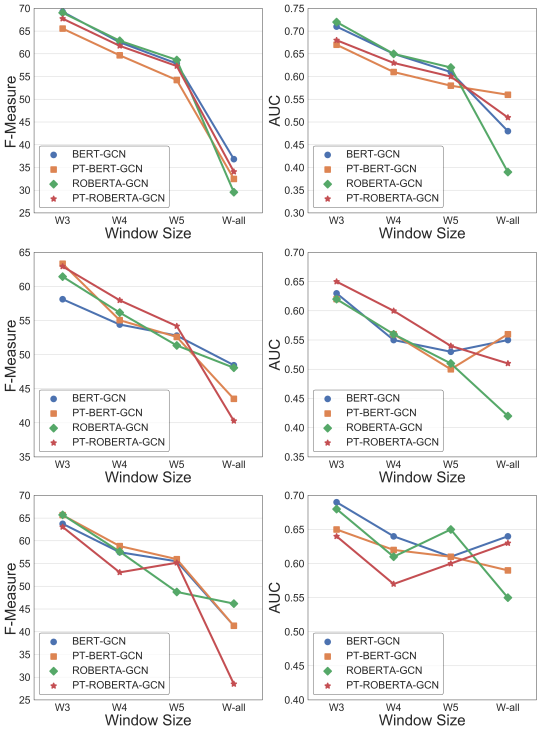

Figure 4: Performance for CTFW, COR, MAM trained from **scratch** and fine-tuned with **pre-trained weights**.

Effect of Pre-trained LM Weights: We study the impact of pre-trained weights of BERT and RoBERTa on the performance in Figure 4. We notice, for the three datasets, the performance slightly decreases when the pre-trained model weights are used. This observation may be because the texts' nature is quite different from the type of texts these LMs have been pre-trained on. The CTFW data often contains code fragments embedded in sentences, emoticons, or common conversational languages used in public forums.

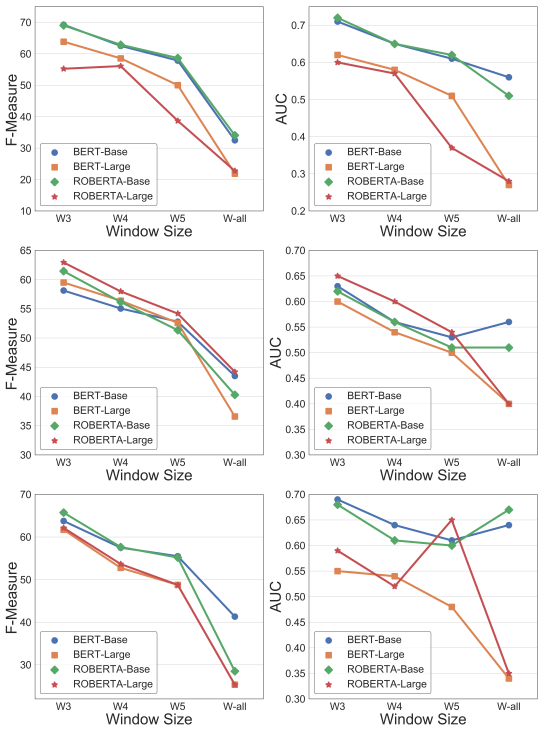

Figure 5: Performance on CTFW, COR, MAM trained with **base** and **large** version of the model.

Effect of LM Size: We also experimented with the size of sentence embeddings to see if that makes any difference to the performance. We use base and large version of BERT and RoBERTa for the experiments across three datasets. We present the impact on F1 and PRAUC in Figure 5. The performance of the larger versions of the models drop in all three datasets. This drop, we believe, is because the sentences in these texts are relatively short and help the smaller versions of the models with lesser parameters to learn better.

Other experiments: We also experimented with modifications of other parts of the models like changing the number of projection layers, projection layer sizes, the number of attention heads in the GAT model, or dropout percent in selected layers and modes of message aggregation (add, max, mean). We do not report them since they do not significantly change PRAUC values.

7 Related Work

Procedural knowledge extraction: There are attempts to extract structured knowledge from cooking instructions in the form of named entities (Mal-

maud et al., 2014), their sentence-level dependencies (Mori et al., 2014; Maeta et al., 2015; Xu et al., 2020), and action-verb argument flow across sentences (Jermsurawong and Habash, 2015; Kiddon et al., 2015; Pan et al., 2020). In other domains, extraction of clinical steps from medline abstracts (Song et al., 2011), extraction of material synthesis operations and its arguments in material science (Mysore et al., 2019), providing structures to how-to procedures (Park and Motahari Nezhad, 2018), and action-argument retrieval from web design tutorials (Yang et al., 2019) mostly focus on fine-grained entity extractions rather than action or information traces. The goal of our paper is constructing flow graphs from free-form, natural-language procedural texts without diverse domain knowledge. Hence, we refrain from training specialized named-entity recognizers for each domain to find specific entities. Our work is related to event or process discovery in process modeling tasks (Epure et al., 2015; Honkisz et al., 2018; Qian et al., 2020; Hanga et al., 2020), but our goal is not finding specific events or actions from procedural texts. In addition, the recent research proposed a method to create the forum structures from an unstructured forum based on the contents of each post using BERT's Next Sentence Prediction (Kashihara et al., 2020). However, we focus on building flow graphs for procedural texts using GNNs.

Graph Neural Networks: GNNs are important in reasoning with graph-structured data in three major tasks, node classification (Kipf and Welling, 2016; Hamilton et al., 2017), link prediction (Schlichtkrull et al., 2018), and graph classification (Ying et al., 2018; Pan et al., 2015, 2016; Zhang et al., 2018). GNNs help learn better node representations in each task using neural message passing (Gilmer et al., 2017) among connected neighbors. We consider two widely used GNNs, GCN (Graph Convolutional Network) (Kipf and Welling, 2016) and GAT (Graph Attention Networks) (Veličković et al., 2017) to learn sentence representation to provide a better edge prediction.

Edge Prediction Task: Edge or link prediction tasks (Li et al., 2018; Zhang and Chen, 2018; Pandey et al., 2019; Haonan et al., 2019; Bacciu et al., 2019) work mainly on pre-existing networks or social graphs as inputs and predict the existence of future edges between nodes by extracting graph-specific features. Different from existing work, we modeled the task of generating a graph-structure from a given natural-language text as an edge prediction task in a graph and learning representations of sentences considered as nodes.

Combinations of BERT and GCN: Recent works have used concatenation of BERT and GCN representations of texts or entities to improve performance of tasks like commonsense knowledge-base completion (Malaviya et al., 2019), text classification (Ye et al.; Lu et al., 2020), multi-hop reasoning (Xiao et al., 2019), citation recommendation (Jeong et al., 2019), medication recommendation (Shang et al., 2019), relation extraction (Zhao et al., 2019). Graph-BERT (Zhang et al., 2020) solely depends on attention layers of BERT without using any message aggregation techniques. However, we differ from each of the previous methods in terms of model architecture, where we use BERT to learn initial sentence representations and GCN or GAT to improve them by learning representations from its neighboring connected sentences. BERT-GAT for MRC (Zheng et al., 2020) created the graph structure from the well-structured wikipedia data whereas we explore two predefined natures of graph structures because of the free-formed text nature without such well-defined text-sections, presence of code-fragments, emoticons, and unrelated-token.

8 Conclusion and Future Work

We introduce a new procedural sentence flow extraction task from natural-language texts. This task is important for procedural texts in every domain. We create a sufficiently large procedural text dataset in the cybersecurity domain (CTFW) and construct structures from the natural form. We empirically show that this task can be generalized across multiple domains with different natures and styles of texts. In this paper, we only focus on English security write-ups. As part of future work, we plan to build automated agents in the cybersecurity domain to help and guide novices in performing software vulnerability analysis. We also plan to include non-English write-ups. We hope the CTFW dataset will facilitate other works in this research area.

Acknowledgement

The authors acknowledge support from the Defense Advanced Research Projects Agency (DARPA) grant number FA875019C0003 for this project.

Impact Statement

The dataset introduced here consists of write-ups written in public forums by students or security professionals from their personal experiences in the CTF challenges. The aggregated knowledge of such experiences is immense. This in-depth knowledge of the analysis tools and the approach to a problem is ideal for students working in software vulnerability analysis to learn from. Automated tutors built using such knowledge can reduce the efforts and time wasted in manually reading through a series of lengthy write-up documents.

CTFTime website states that the write-ups are copyrighted by the authors who posted them and it was practically impossible to contact each authors. It is also allowed to use the data for research purposes (usc; euc) Thus, we follow the previous work (Švábenskỳ et al., 2021) using data from CTFTime and share only the urls of those write-ups from the CTFTime website which we use. We do not provide the scraper script since it would create a local copy of the write-up files unauthorized by the users. Interested readers can replicate the simple scraper script from the instructions in Appendix A and use it after reviewing the conditions under which it is permissible to use. We, however, share our annotations for those write-ups files.

Part of the annotations were provided as an optional, extra-credit assignment for the Information Assurance course. These CTF write-ups were directly related to the course-content, where students were required to read existing CTF write-ups and write write-ups for other security challenges they worked on during the course. Then students were given the option of voluntarily annotating CTF write-ups they read for extra credits in the course. For this task, we followed all the existing annotation guidelines and practices. We also ensured that

- The volunteers were aware of the fact that their annotations would be used for a research project.
- They were aware that no PII was involved or would be used in the research project.
- They were aware that extra credits were entirely optional, and they could refrain from submitting at any point of time without any consequences.
- Each volunteer was assigned only 10-15 write-ups based on a pilot study we did ahead of time, annotating an average-length CTF write-up took about two minutes (maximum ten mins).

Remaining annotations were performed by the Teaching Assistants (TA) of the course. These annotations were done as part of the course preparation process, which was part of their work contract. All the TAs were paid bi-weekly compensation by the university or by research funding. It was also ensured that the TAs knew these annotations would be used for a research project, their PII was not involved and annotations were to be anonymized before using.

References

European union, copyright in the eu, 2020. https://europa.eu/youreurope/business/running-business/intellectual-property/copyright/index_en.htm. Accessed: 2021-05-01.

Us copyright office, copyright law of the united states, 2016. https://www.copyright.gov/title17/92chap1.html#107. Accessed: 2021-05-01.

Davide Bacciu, Alessio Micheli, and Marco Podda. 2019. Graph generation by sequential edge prediction. In *ESANN*.

CTFTime. CTFTime. https://ctftime.org. Accessed: 2021-05-01.

Estelle Delpech and Patrick Saint-Dizier. 2008. Investigating the structure of procedural texts for answering how-to questions. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco. European Language Resources Association (ELRA).

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

Elena Viorica Epure, Patricia Martín-Rodilla, Charlotte Hug, Rebecca Deneckère, and Camille Salinesi. 2015. Automatic process model discovery from textual methodologies. In *2015 IEEE 9th International Conference on Research Challenges in Information Science (RCIS)*, pages 19–30. IEEE.

Matthias Fey and Jan Eric Lenssen. 2019. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*.

Lionel Fontan and Patrick Saint-Dizier. 2008. Analyzing the explanation structure of procedural texts: Dealing with advice and warnings. In *Semantics in Text Processing. STEP 2008 Conference Proceedings*, pages 115–127.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*.

Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034.

Khadijah Muzzammil Hanga, Yevgeniya Kovalchuk, and Mohamed Medhat Gaber. 2020. A graph-based approach to interpreting recurrent neural networks in process mining. *IEEE Access*, 8:172923–172938.

Lu Haonan, Seth H Huang, Tian Ye, and Guo Xiuyan. 2019. Graph star net for generalized multi-task learning. *arXiv preprint arXiv:1906.12330*.

Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*.

Krzysztof Honkisz, Krzysztof Kluza, and Piotr Wiśniewski. 2018. A concept for generating business process models from natural language description. In *International Conference on Knowledge Science, Engineering and Management*, pages 91–103. Springer.

Chanwoo Jeong, Sion Jang, Hyuna Shin, Eunjeong Park, and Sungchul Choi. 2019. A context-aware citation recommendation model with bert and graph convolutional networks. *arXiv preprint arXiv:1903.06464*.

Jermsak Jermsurawong and Nizar Habash. 2015. Predicting the structure of cooking recipes. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 781–786.

Kazuaki Kashihara, Jana Shakarian, and Chitta Baral. 2020. Social structure construction from the forums using interaction coherence. In *Proceedings of the Future Technologies Conference*, pages 830–843.

Chloé Kiddon, Ganesa Thandavam Ponnuraj, Luke Zettlemoyer, and Yejin Choi. 2015. Mise en place: Unsupervised interpretation of instructional recipes. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 982–992.

Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Ji-chao Li, Dan-ling Zhao, Bing-Feng Ge, Ke-Wei Yang, and Ying-Wu Chen. 2018. A link prediction method for heterogeneous networks based on bp neural network. *Physica A: Statistical Mechanics and its Applications*, 495:1–17.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Zhibin Lu, Pan Du, and Jian-Yun Nie. 2020. Vgcn-bert: Augmenting bert with graph embedding for text classification. In *European Conference on Information Retrieval*, pages 369–382. Springer.

Hirokuni Maeta, Tetsuro Sasada, and Shinsuke Mori. 2015. A framework for procedural text understanding. In *Proceedings of the 14th International Conference on Parsing Technologies*, pages 50–60.

Chaitanya Malaviya, Chandra Bhagavatula, Antoine Bosselut, and Yejin Choi. 2019. Exploiting structural and semantic context for commonsense knowledge base completion. *arXiv preprint arXiv:1910.02915*.

Jonathan Malmaud, Earl Wagner, Nancy Chang, and Kevin Murphy. 2014. Cooking with semantics. In *Proceedings of the ACL 2014 Workshop on Semantic Parsing*, pages 33–38.

Shinsuke Mori, Hirokuni Maeta, Yoko Yamakata, and Tetsuro Sasada. 2014. Flow graph corpus from recipe texts. In *LREC*, pages 2370–2377.

Sheshera Mysore, Zach Jensen, Edward Kim, Kevin Huang, Haw-Shiuan Chang, Emma Strubell, Jeffrey Flanigan, Andrew McCallum, and Elsa Olivetti. 2019. The materials science procedural text corpus: Annotating materials synthesis procedures with shallow semantic structures. *arXiv preprint arXiv:1905.06939*.

Liang-Ming Pan, Jingjing Chen, Jianlong Wu, Shaoteng Liu, Chong-Wah Ngo, Min-Yen Kan, Yugang Jiang, and Tat-Seng Chua. 2020. Multi-modal cooking workflow construction for food recipes. In *Proceedings of the 28th ACM International Conference on Multimedia*, pages 1132–1141.

Shirui Pan, Jia Wu, Xingquan Zhu, Guodong Long, and Chengqi Zhang. 2016. Task sensitive feature exploration and learning for multitask graph classification. *IEEE transactions on cybernetics*, 47(3):744–758.

Shirui Pan, Jia Wu, Xingquan Zhu, Chengqi Zhang, and S Yu Philip. 2015. Joint structure feature exploration and regularization for multi-task graph classification. *IEEE Transactions on Knowledge and Data Engineering*, 28(3):715–728.

Babita Pandey, Praveen Kumar Bhanodia, Aditya Khamparia, and Devendra Kumar Pandey. 2019. A comprehensive survey of edge prediction in social networks: Techniques, parameters and challenges. *Expert Systems with Applications*, 124:164–181.

Hogun Park and Hamid Reza Motahari Nezhad. 2018. Learning procedures from text: Codifying how-to procedures in deep neural networks. In *Companion Proceedings of the The Web Conference 2018*, pages 351–358.

Chen Qian, Lijie Wen, Akhil Kumar, Leilei Lin, Li Lin, Zan Zong, Jianmin Wang, et al. 2020. An approach for process model extraction by multi-grained text classification. In *International Conference on Advanced Information Systems Engineering*, pages 268–282. Springer.

Kenneth Reitz. Requests: Http for humans. https://requests.readthedocs.io/en/master/. Accessed: 2020-10-23.

Leonard Richardson. 2007. Beautiful soup documentation. *April*.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pages 593–607. Springer.

Junyuan Shang, Tengfei Ma, Cao Xiao, and Jimeng Sun. 2019. Pre-training of graph augmented transformers for medication recommendation. *arXiv preprint arXiv:1906.00346*.

Sa-kwang Song, Heung-seon Oh, Sung Hyon Myaeng, Sung-Pil Choi, Hong-Woo Chun, Yun-Soo Choi, and Chang-Hoo Jeong. 2011. Procedural knowledge extraction on medline abstracts. In *International Conference on Active Media Technology*, pages 345–354. Springer.

spaCy. 2017. spacy v2.0. https://spacy.io/models/en#en_core_web_md.

Valdemar Švábenskỳ, Pavel Čeleda, Jan Vykopal, and Silvia Brišáková. 2021. Cybersecurity knowledge and skills taught in capture the flag challenges. *Computers & Security*, 102:102154.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Thomas Wolf, Julien Chaumond, Lysandre Debut, Victor Sanh, Clement Delangue, Anthony Moi, Pierric Cistac, Morgan Funtowicz, Joe Davison, Sam Shleifer, et al. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45.

Yunxuan Xiao, Yanru Qu, Lin Qiu, Hao Zhou, Lei Li, Weinan Zhang, and Yong Yu. 2019. Dynamically fused graph network for multi-hop reasoning. *arXiv preprint arXiv:1905.06933*.

Frank F Xu, Lei Ji, Botian Shi, Junyi Du, Graham Neubig, Yonatan Bisk, and Nan Duan. 2020. A benchmark for structured procedural knowledge extraction from cooking videos. *arXiv preprint arXiv:2005.00706*.

Yoko Yamakata, Shinsuke Mori, and John A Carroll. 2020. English recipe flow graph corpus. In *Proceedings of The 12th Language Resources and Evaluation Conference*, pages 5187–5194.

Longqi Yang, Chen Fang, Hailin Jin, Walter Chang, and Deborah Estrin. 2019. Creative procedural-knowledge extraction from web design tutorials. *arXiv preprint arXiv:1904.08587*.

Zhihao Ye, Gongyao Jiang, Ye Liu, Zhiyong Li, and Jin Yuan. Document and word representations generated by graph convolutional network and bert for short text classification.

Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. 2018. Hierarchical graph representation learning with differentiable pooling. In *Advances in neural information processing systems*, pages 4800–4810.

Jiawei Zhang, Haopeng Zhang, Li Sun, and Congying Xia. 2020. Graph-bert: Only attention is needed for learning graph representations. *arXiv preprint arXiv:2001.05140*.

Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*, pages 5165–5175.

Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An end-to-end deep learning architecture for graph classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Yi Zhao, Huaiyu Wan, Jianwei Gao, and Youfang Lin. 2019. Improving relation classification by entity pair graph. In *Asian Conference on Machine Learning*, pages 1156–1171.

Bo Zheng, Haoyang Wen, Yaobo Liang, Nan Duan, Wanxiang Che, Daxin Jiang, Ming Zhou, and Ting Liu. 2020. Document modeling with graph attention networks for multi-grained machine reading comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6708–6718, Online. Association for Computational Linguistics.

A Extraction and Processing of Write-ups:

The extraction of CTF Write-up involved the following three phases.

Writeup URL extraction : We loop through all the write-up pages on ctftime website from page numbers 1 to 25500). We use a simple python scraper to scrape the content of each page using python requests (Reitz) library. We look for keyword "*Original write-ups*" and extracted the href component if present. These URLs are stored for each writeup indexed with the page numbers.

Write-up Content extraction : We use these URLs and extract the contents of the write-ups using python libraries requests and BeautifulSoup (Richardson, 2007). We extract all the text lines ignoring contents in html tags like style, scripts, head, title. The contents are stored in a text file named with the same page ids of the URLs.

Processing of Write-up : We processed and filter out sentences which do not have any verb forms using spacy (spaCy, 2017) POS-Tagger. We cleaned and removed unnecessary spaces and split them into sentences. The processing script is available in the github.

B CTFW Data Statistics

In CTFW, there are write-ups for 2236 unique tasks. Only four out of those having more than 5 write-ups each. 72% of the tasks have single write-up. The write-ups are from 311 unique competitions, ranging from years 2012-2019. A task having multiple write-ups vary in contents. In CTFW, only 3% of the tasks have more than three write-ups, and 9% have more than two.

C Training Details:

The correct set of hyperparameters are found by running three trials. We run for {50, 100} epochs and store the model with the best PRAUC score. Each training with evaluation takes around 1-3 hours for base version of models and around 6 hours for larger versions depending upon the dataset used. The model parameters are directly proportional to the model parameters of language models, since the GNN only allow few more parameters as compared to the LMs.

D Baseline NS with weighted cross-entropy

Table 5 shows the PRAUC values when we use weighted cross-entropy with base version of BERT on unbalanced data during training. The results are not much different than the Next-Sentence baseline shown previously.

Dataset	W_3	W_4	W_5	W_{all}
CTFW	0.4613	0.4397	0.2546	0.3681
COR	0.4724	0.4748	0.4837	0.4761
MAM	0.5318	0.2318	0.2297	0.4724

Table 5: BERT-base-uncased performance with NS prediction when Weighted Cross-Entropy used with Unbalanced Training Data

E Number of comparisons Reduction using Windows

We can control the total number of comparisons required to predict the edges in a graph by using the windows (W_N where $N = 3, 4, 5, all$). The number of comparisons for each window is given by the equation 7. We can reduce the number of comparisons considerably for large documents using shorter windows of 3, 4, 5 sentences. The number of comparison $\mathcal{C}$ is defined by

$$\mathcal{C} = \begin{cases} max\{(n-s), 0\}s + \frac{s(s-1)}{2} & n = 3, 4, 5 \\ \binom{n}{2} & n = all \end{cases} \quad (7)$$

F CTFW STC Label statistics

Table 6 shows the label distributions of Sentence Type Classification data.

Label	Train	Val	Test
A	11143	1499	3321
I	23279	3075	6882
A/I	2931	380	826
C	1386	185	338
NONE	82012	12192	22896

Table 6: CTFW Sentence Type Classification