EXPLOITING SEMANTIC STRUCTURES TOWARD PROCEDURAL REASONING

By

Hossein Rajaby Faghihi

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Computer Science—Doctor of Philosophy

2024

ABSTRACT

Reasoning over procedural text, which encompasses texts such as recipes, manuals, and 'how to' tutorials, presents formidable challenges due to the dynamic nature of the world it describes. Reasoning over procedural text has many challenges. These challenges are embodied in tasks such as 1) tracking entities and their status changes (entity tracking) and 2) summarization of the process (procedural summarization). This thesis aims to enhance the representation and reasoning over textual procedures by harnessing semantic structures in the input text and imposing constraints on the models' output. It delves into using semantic structures derived from the text, including relationships between actions and objects, semantic parsing of instructions, and the sequential structure of actions. Additionally, the thesis investigates the integration of structural and semantic constraints within neural models, resulting in coherent and consistent outputs that align with external knowledge. The thesis contributes significantly to three main areas: **Entity tracking**, **Procedural Abstraction**, and the **Integration of constraints in deep learning**.

In the entity tracking task, we have made four primary contributions: 1) Developed a novel architecture for encoding event flow in pretrained language models. 2) Enabled seamless transfer learning from diverse corpora through task reformulation. 3) Enhanced language models by incorporating knowledge from semantic parsers and leveraging ontological abstraction of actions. 4) Created a new evaluation scheme considering fine-grained semantics in tracking entities.

Regarding procedural summarization, the thesis proposes a model for an explicit latent space for the procedure that is indirectly supervised to ensure the summary's action order corresponds to the order of events in the multi-modal instructions.s

In the realm of integrating domain knowledge with deep neural networks, the thesis makes two significant contributions, 1) it contributes to the development of a generic framework that facilitates the incorporation of first-order logical constraints in neural models, and 2) it creates a new benchmark for evaluating constraint integration methods across five categories of tasks. This benchmark introduces novel evaluation criteria and offers valuable insights into the effectiveness of constraint integration methods across various tasks.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

MSU	Michigan State University
NLP	Natural Language Understanding
AI	Artificial Intelligence
NER	Named Entity Recognition
PLM	Pre-trained Language Models
LLM	Large Language Models
TSLM	Time-Stamped Language Models

CHAPTER 1

INTRODUCTION

1.1 Motivation

Natural language understanding concerns developing algorithms and models that can interpret and analyze human language. The ability to understand natural language has numerous practical applications, such as improving search engines [1], automatic customer service [30], and creating intelligent personal assistants [79]. However, natural language understanding remains a complex and challenging problem due to the complexity and variability of human language. In recent years, there has been significant progress in developing machine learning models and techniques to tackle this problem. In this manuscript, we focus on natural language understanding in the domain of procedural text.

Procedural texts, such as recipes [16, 184], manuals and tutorials [170, 180], and stories [160], provide step-by-step instructions for various tasks. Understanding the procedural text is important in natural language processing [171] and robotics [164]. In addition to the general challenges of natural language understanding, comprehending procedural text is difficult due to the dynamic nature of the environment described in the process. This results in new challenges specific to this domain. We pinpoint some of these as follows:

- **Reasoning over the temporal relationships:** Procedural texts describe a sequence of steps that must be performed in a specific order. For example, if the instruction is "add the flour after mixing the wet ingredients", the agent should understand that the sequence of action is "mix the wet ingredients" and then "add the flour".

- **Reasoning over the global context:** This challenge arises when understanding a particular step requires information from previous or future steps. For instance, a recipe may instruct to "stir the vegetables" followed by "take them out of the pan". Without looking at the subsequent steps, it may not be clear where the vegetables should be stirred in.

- **Grounding actions:** Procedural texts often use synonyms or multiple meanings of a word to describe the same physical action. For example, a recipe might use the terms "chop," "dice," and "mince" to refer to the action of cutting ingredients into small pieces.

- **Grounding entities:** Procedural texts may have complex entity grounding, making it difficult to determine which entity is being referred to in a particular step. For example, grounding the vegetables in the "stir the vegetables" instruction is difficult when the recipe contains multiple ingredients, such as {parsely, carrot, meat, and sugar}. One requires additional knowledge about the type of ingredients to understand this instruction fully.

- **Reasoning over action sequences:** Procedural texts require a deep understanding of the actions being performed, including the sequence and dependencies between them. For instance, if a process describes entity ‘animal’ to ‘die’ (destroy) at step i, the ‘animal‘ cannot be moved at step i 1.

- **Reasoning over causal relations**: Procedural texts require the ability to reason about causality, to understand how actions relate to each other, and what the consequences of a specific action might be. For instance, in the sequence of information "1. Wind creates waves 2. Waves hit rocks on the beach", it can be inferred that calm weather would result in smaller waves and, hence, less damage to the rocks on the beach.

- **Reasoning over actions and their consequences:** There are complex dependencies between the actions and the changes in the world that should be understood. For instance, if entity ‘water‘ is detected to be moving in the sentence ‘water from the soil is absorbed‘, it should be inferred that the new location of ‘water‘ is different than ‘soil‘.

These challenges have been embodied in multiple tasks, such as entity tracking [32, 16], procedural abstraction/summarization [184], and causal reasoning [169]. This thesis focuses on entity tracking and procedural abstraction, delving deeper into each task’s challenges in their corresponding sections in the subsequent chapters. In addition to the intrinsic challenges of understanding procedural text due to its dynamic nature, the available benchmarks for this task

		Participants				
Paragraph	State number	Water	Light	CO2	Mixture	Sugar
(Before the process starts)	State 0	Soil	Sun	?	-	-
Roots absorb water from soil	State 1	Root	Sun	?	-	-
The water flows to the leaf	State 2	Leaf	Sun	?	-	-
Light from the sun and CO2 enter the leaf	State 3	Leaf	Leaf	Leaf	-	-
Water, light, and CO2 combine into mixture	State 4	-	-	-	Leaf	-
Mixture forms sugar	State 5	-	-	-	-	Leaf

Table 1.1 An example of procedural text and its annotations from the Propara dataset [32]. "-" means the entity does not exist. "?" means the location of the entity is unknown.

are mostly small and provide **smaller amounts of annotated data** compared to many other NLP tasks. This is due to the costly and complex human process of providing fine-grained annotations of world states during the process. Training neural models to learn from a smaller set of annotated data can result in challenges in generalization and may cause overfitting.

In order to tackle the challenges discussed earlier, our objective is to leverage the semantic structures that can be derived from procedural text or the interdependencies among various decisions made by neural models, also known as structured prediction, especially based on models designed for procedural reasoning. Prior to outlining our contributions, We discuss the scenarios of procedural understanding that are the focus of our investigation in this thesis.

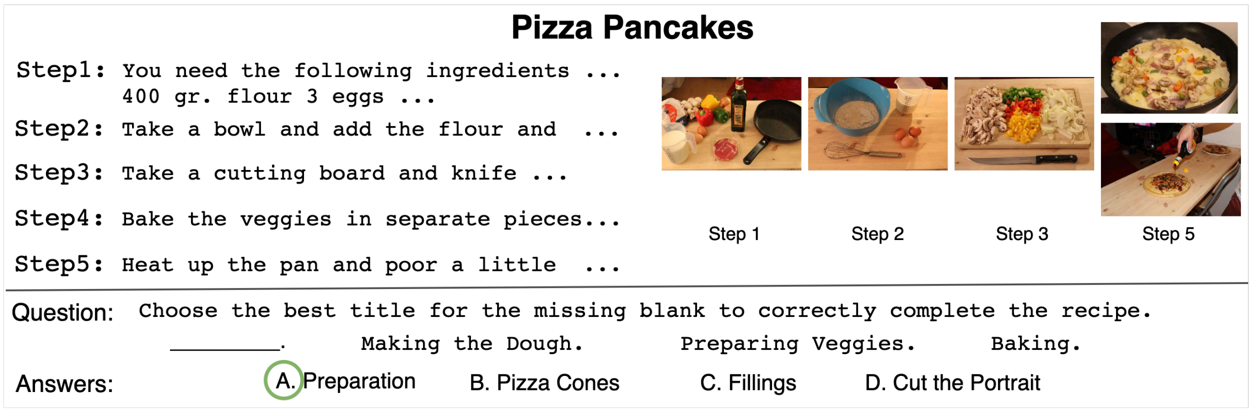
1.2 Procedural Reasoning Tasks

Here, we briefly describe the main procedural reasoning problems that we focus on, that is, entity tracking and procedural abstraction.

Entity tracking involves monitoring entities' states as they undergo a series of actions, while procedural abstractions aim to generate a condensed summary of the steps involved in a process. Both tasks are essential for a comprehensive understanding of the process, as they capture different aspects of the task.

Entity Tracking Table 1.1 presents an illustrative example of the entity tracking task. This task revolves around processing step-by-step instructions along with the entities involved in a given process, with the aim of generating comprehensive annotations that trace the evolution of these entities' properties over the course of the instructions. The property of interest shown in Table

1.1 is the location of entities. To expound further, consider the provided sentence, "Roots absorb water from soil," and focus on the entity of interest, namely, "water." In this context, the primary objective of the entity tracking task is to discern and highlight the relocation of the "water" entity as it transitions from its original location in the "soil" to its eventual destination in the "root."


Pizza Pancakes

```
Step1: You need the following ingredients ...
       400 gr. flour 3 eggs ...
Step2: Take a bowl and add the flour and  ...
Step3: Take a cutting board and knife ...
Step4: Bake the veggies in separate pieces...
Step5: Heat up the pan and poor a little  ...
```

Question: `Choose the best title for the missing blank to correctly complete the recipe.`

`_________.   Making the Dough.   Preparing Veggies.   Baking.`

Answers: [✓ A.] Preparation B. Pizza Cones C. Fillings D. Cut the Portrait

Figure 1.1 A sample input for the task of procedural summarization in a simplified setting. This task is known as Textual Cloze.

Procedural Abstraction The procedural abstraction task refers to the process of condensing a series of actions and their associated outcomes in order to provide a concise and short representation that accurately captures the original sequence of events.

To facilitate the examination of this task, a simplified variant can be explored, wherein the abstract consists of only four action phrases that effectively convey the chronological progression of the process. An illustrative example of this task is presented in Figure 1.1. In this example, the sequence denoted by "1. Preparation, 2. Making the Dough, 3. Preparing Veggies, and 4. Baking" serves as an abstract and succinct action-oriented interpretation of the procedure of baking "Pizza Pancakes". Compared to the full summarization task, it is worth noting that this simplified version deliberately overlooks the element of "comprehensiveness" and relaxes the requirements for "coherence".

1.3 Contributions

In this thesis, our primary aim is to exploit the inherent semantic structures in language and constraints in structured prediction tasks to advance Procedural Reasoning over text.

Firstly, we focus on utilizing the input semantic structures derived from the text, such as the interrelationships between actions and objects, the semantic parse of the instructions, and the relative ordering of actions. By leveraging these semantic structures, we can enhance the understanding and representation of procedural knowledge.

Secondly, we aim to harness the power of structural and semantic constraints over the neural model's decision and latent variables. This line of inquiry falls within the domain of structured output prediction, where the model outputs are correlated within a certain structure. For instance, in predicting actions associated with a particular entity at each step, there exists a sequential constraint: if the action at step i is "create," it is impossible for the action at step i 1 to also be "create." By incorporating these structural constraints, we can guide the neural models to generate more coherent and realistic outputs, aligning them with the inherent constraints of procedural knowledge

In pursuit of our objective, we put forth a range of innovative neural architectures that leverage semantic enhancements applied to the instructions, as well as novel learning objectives devised based on the interconnections among latent and output variables of neural models. Our primary focus is on addressing the challenges inherent in procedural understanding, which we tackle through formulating two distinct tasks: entity tracking and procedural abstraction.

To harness the semantic dependencies present in the latent and output variables of neural models, we delve into the broader domain of integrating domain knowledge with deep neural networks (DNN). Within this framework, we specifically concentrate on incorporating knowledge that can be expressed as first-order logical constraints. By integrating domain knowledge, we strive to enhance the reasoning capabilities and overall performance of the neural models, especially in capturing the intricacies of procedural reasoning.

In summary, our main contributions toward the task of **entity tracking** are:

- We design a novel mechanism to **encode the relative time of events** and capture the **flow**

of events within language models. This enables us to reason over the whole context of the process while being able to shift focus from one step to another. Our method significantly enhances language models' ability to reason over action sequences and both local and global contexts. We evaluate this mechanism in the task of entity tracking and achieve state-of-the-art results.

- We provide **reformulations** for the entity tracking task, enabling effective **transfer learning** from generic domain question answering benchmarks to the procedural domain. This reformulation helps address the challenges arising from limited available annotations for the entity tracking task.

- We provide a novel approach to **utilize symbolic and neural semantic parsers** alongside **ontological hierarchies of actions** in understanding procedural texts. This approach combines semantic parses represented as graphs and encoded via graph attention networks with pretrained language models. Our proposal improves the ability of baseline models to address challenges in grounding actions, reasoning over action sequences, and understanding actions and their consequences. This approach also enhances the explainability of neural decision-making processes and improves model generalizability through proper abstractions over textual information. We evaluate this approach in the entity tracking task and achieve further improvements over the state-of-the-art results.

- We provide **new evaluation metrics** to better understand the challenges of the entity tracking task based on the difficulty of required reasoning steps for tracking the entities.

- We are the pioneer in enforcing global consistency in the entity tracking task, which ensures decision consistency while considering all the model decisions in parallel rather than sequential, resulting in a better performance compared to sequential inference.

Our contributions toward the task of **procedural abstraction** are listed below:

- We proposed a new training objective that utilizes distant supervision signals based on constraints on generating summaries of recipes. These supervision signals are applied in a latent matching space between summary points and the instruction steps, and the model is simply enforced to match earlier summary points to earlier steps of the process.
- We proposed and showcased the benefits of using a multi-modal architecture over a single textual modality model in generating process abstractions.
- We analyzed and provided insights and reasons on cases where the multi-modal information is helpful or hurtful in processing the process instructions.

Finally, our main contributions toward the **integration of domain knowledge with DNNs** are:

- We develop a generic framework for integrating constraints with deep neural models (DomiKnowS). This framework provides a declarative interface to define knowledge and computational units while providing a senseless transition between multiple integration techniques.
- We create a new evaluation benchmark (GLUECons), which facilitates research on integrating explicit knowledge of the task with learning from data. We further showcase the advantage of integrating domain knowledge with deep neural models in multiple tasks and under various configurations over data size and network parameters.
- We propose a novel framework for mapping decisions originating from various models into comparable values in a unified framework for enforcing consistency in the heterogeneous decision-making process.

1.4 Manuscript Outline

In this section, we provide a short overview of the chapters of this document and discuss the structure of the remaining chapters.

Chapter 3 focuses on the entity tracking task. It begins with a formal definition of the task in Section 3.1, followed by a detailed discussion of the specific challenges associated with entity tracking in Section 3.2. Section 3.3 explores existing evaluation metrics for this task, while Section 3.4 provides an overview of related research. Next, Section 3.5 will cover our approach for reformulating the tasks of entity tracking as question-answering and a novel method to encode the relative time frame of events in pretrained language models. Our contributions discussed in this section are published as part of the paper "Time-Stamped Language Model: Teaching Language Models to Understand the Flow of Events" [48] in NAACL 2021. Section 3.6 will cover our main contributions toward utilizing semantic parsers toward understanding the procedural text in the form of entity tracking. The results of this effort are part of the paper "The Role of Semantic Parsing in Understanding Procedural Text" [50] in EACL 2023.

Chapter 4 delves into the task of procedural abstraction. It examines the simplified action-based abstraction task. The approach for mapping the temporal order of actions in abstract and the process is described in Section 4.1. The results of this work were published in ALVR2021 under the title "Latent alignment of procedural concepts in multimodal recipes" [131].

In Chapter 5, we explain our general approach for integrating domain knowledge with neural learning. Section 5.1 describes our declarative learning-based framework, DomiKnowS, for knowledge integration. Section 5.2 introduces GLUECons, a new benchmark for evaluating constraint integration methods. Our efforts in designing DomiKnowS are presented in EMNLP2021 under the title of "DomiKnowS: A Library for Integration of Symbolic Domain Knowledge in Deep Learning" [47]. GLUECons is also presented in AAAI2023 under the name of "GLUECons: A Generic Benchmark for Learning Under Constraints"[51].

In Chapter 6, we combine two key aspects of our research: procedural reasoning and the integration of domain knowledge with neural learning. To fully harness the potential of our proposed approach for integrating knowledge, we extend existing methods to ensure consistency during inference. These extensions are tailored to handle scenarios where we need to combine decisions with different characteristics, such as size and accuracy. In Section 6.3, we explain our new technical framework for these extensions. Furthermore, in Section 6.2, we propose a novel technique for procedural reasoning within a generative language model. Our approach aims at enhancing the consistency of reasoning within the language model through data augmentation and knowledge integration.

Finally, in Chapter 7, we summarize our contributions and propose future directions and possible extensions of our work.

CHAPTER 2

BACKGROUND

In this chapter, we will review the basic definitions and related works which are essential to understand the contributions of this work. We describe the general research relevant to each contribution in the subsequent chapters.

2.1 Language Models

Throughout this document, we refer to language models in terms of transformer-based neural architectures, which are mostly pre-trained on the masked/next token prediction task. Transformer-based language models are a type of deep learning model used for natural language processing tasks such as language generation [145], machine translation [173], and text classification [119]. They are based on the Transformer architecture introduced by [173] in 2017.

2.1.1 Transformers

The Transformer architecture [173] is designed to address the limitations of traditional recurrent neural networks (RNNs) [107] and convolutional neural networks (CNNs)[72] in processing long-range dependencies in sequential data such as text. It does this by introducing self-attention mechanisms that allow the model to selectively attend to different parts of the input sequence, regardless of their position in the sequence.

Masked Token Prediction Masked token prediction is a task where the model is given a sequence of tokens with some tokens randomly masked, and the model is asked to predict the masked tokens based on the context of the surrounding tokens. Formally, given a sequence of tokens $X = \{x_1, x_2, \ldots, x_n\}$, a subset of the tokens are randomly selected and replaced with a special $MASK$ token to obtain a masked sequence $Y = \{y_1, y_2, \ldots, y_n\}$, where y_j is either x_j or $MASK$. The goal of the model is to predict the original tokens of the masked positions in Y, i.e., to find the sequence $Z = \{z_1, z_2, \ldots, z_n\}$, where $z_{i,j} = x_i$ if $y_j = x_i$ and $z_{i,j}$ is the predicted token if $y_j = MASK$.

The objective function used to train the model is typically the cross-entropy loss between the predicted probability distribution over the vocabulary and the true distribution of the original tokens. Let $\hat{y}_j$ be the predicted distribution over the vocabulary for the j-th masked position, and let px_i be the true probability distribution of the original token x_i. The loss function can be defined as:

$$\mathcal{L}_{\text{mask}} = - {}_{j=1}^{n} {}_{i=1}^{|\mathcal{V}|} px_i \log \hat{y}_{j,i}$$

where $\mathcal{V}$ is the vocabulary of the language.

Next Token Prediction Next token prediction is a task where the model is given a sequence of tokens and is asked to predict the next token in the sequence. Formally, given a sequence of tokens $X = \{x_1, x_2, \ldots, x_n\}$, the model is asked to predict the probability distribution over the vocabulary of the next token x_{n1}. The objective function used to train the model is typically the cross-entropy loss between the predicted probability distribution over the vocabulary and the true distribution of the next token x_{n1}.

The model is trained in an iterative approach, where the true next token is provided as input to the model at each time step during training. During inference, the model can generate text by recursively predicting the next token given the previously generated tokens. Let $\hat{y}$ be the predicted distribution over the vocabulary for the next token, and let px_{n1} be the true probability distribution of the next token. The loss function can be defined as:

$$\mathcal{L}_{\text{next}} = - {}_{i=1}^{|\mathcal{V}|} px_{n1} \log \hat{y}_i$$

where $\mathcal{V}$ is the vocabulary of the language.

During inference, the model can generate text by recursively predicting the next token given the previously generated tokens. Let $X^t = \{x_1, x_2, \ldots, x_t\}$ be the first t tokens of the sequence, and let $\hat{y}^{t1}$ be the predicted distribution over the vocabulary for the next token given X^t. The model can then sample a token from the predicted distribution and add it to the sequence to obtain X^{t1}. This process can be repeated until a stopping criterion is met, such as reaching a maximum sequence length or generating an end-of-sentence token.

Next token prediction is a fundamental task in language modeling, and it is often used as a pretraining task for more complex natural language processing tasks, such as machine translation and question answering. The success of the next token prediction relies on the ability of the model to capture the dependencies between the tokens in the sequence and to learn a representation of the language that can generalize to new tasks.

2.1.2 Encoder-only Models

Encoder-only models are a type of transformer-based language model that consists of an encoder component only, without a decoder. This means that the model is trained to generate representations of input sequences that capture the contextual relationships between the tokens in the sequence, but it does not generate output sequences. Two popular examples of encoder-only models are BERT (Bidirectional Encoder Representations from Transformers) [42] and RoBERTa (Robustly Optimized BERT Pretraining Approach) [95].

Formally, the input to the encoder-only model is a sequence of tokens $X = x_1, x_2, \ldots, x_n$, which is first tokenized and then fed into the model. The model generates a sequence of hidden representations $H = h_1, h_2, \ldots, h_n$, where h_i is the hidden representation of the i-th token in the input sequence.

The architecture of the encoder-only model is based on the transformer architecture, which consists of a stack of L identical layers. Each layer has two sub-layers: a self-attention mechanism and a position-wise fully connected feed-forward network. The self-attention mechanism allows the model to attend to different positions of the input sequence to generate the hidden representation of each token. The position-wise feed-forward network allows the model to capture complex interactions between the tokens.

The objective function used to train the encoder-only model is typically the masked language modeling task.

BERT BERT (Bidirectional Encoder Representations from Transformers) [42] is an encoder-only transformer-based language model developed by Google AI Language. It was introduced in 2018 and has achieved state-of-the-art results on various natural language processing tasks, including question answering, sentiment analysis, and named entity recognition.

The architecture of BERT is based on a transformer encoder that is pre-trained on large amounts of text data using the masked language modeling task and the next sentence prediction task. During training, a subset of the tokens in the input sequence is randomly masked, and the model is trained to predict the original tokens based on the context of the surrounding tokens. The next sentence prediction task involves predicting whether two sentences are consecutive in a document or whether they are randomly sampled from the corpus.

After pre-training, the weights of the BERT model can be fine-tuned on a downstream task with a smaller labeled dataset, such as sentiment analysis or named entity recognition. Fine-tuning involves adding a task-specific output layer on top of the pre-trained encoder and training the model to minimize the task-specific loss function.

One of the key features of BERT is its bidirectional nature, which allows it to capture the contextual relationships between the tokens in both directions. This is achieved through the use of the masked language modeling task, which requires the model to predict the original tokens based on the context of both the preceding and following tokens.

RoBERTa RoBERTa (Robustly Optimized BERT Pretraining Approach) [95] is an encoder-only transformer-based language model developed by Facebook AI Research. It was introduced in 2019 and is an extension of the BERT model, with improvements to the pre-training objectives and hyperparameters.

The architecture of RoBERTa is similar to that of BERT, with a transformer encoder that is pre-trained on large amounts of text data using a variety of pre-training objectives, including a new task called dynamic masking. Dynamic masking involves randomly masking different spans of tokens in the input sequence, as opposed to contiguous blocks, as in the original masked language modeling task. The masked token would change from one iteration to another.

RoBERTa also makes several hyperparameter optimizations to the BERT model, such as increasing the batch size, training for longer, and using a larger number of training steps. These optimizations result in a more robust pre-trained model that is less sensitive to hyperparameter choices and can be fine-tuned on a wider range of downstream tasks.

In addition, RoBERTa uses a technique called byte-pair encoding (BPE) to handle out-of-vocabulary (OOV) words, which involves breaking down rare or unknown words into subword units. This allows the model to handle rare or unseen words better and improves its generalization ability.

2.1.3 Encoder-Decoder Models

Encoder-decoder models are a class of neural network architectures commonly used in natural language processing tasks such as machine translation and text summarization. They consist of an encoder network that processes the input sequence and generates a fixed-size representation of it and a decoder network that generates the output sequence based on the encoder representation.

To adapt encoder-only models to the encoder-decoder architecture, a decoder network is added on top of the pre-trained encoder. The encoder network first processes the input sequence to generate a fixed-size representation $\mathbf{h}$, which is then passed to the decoder network. The decoder generates the output sequence $\mathbf{y} = y_1, \ldots, y_m$ one token at a time, where y_i represents the i-th token in the output sequence and m is the length of the output sequence.

During decoding, the model uses an attention mechanism to dynamically weigh the importance of each encoder representation based on the decoder's current state and the previous output tokens. This allows the model to capture long-range dependencies between the input and output sequences and generate fluent and coherent output.

Encoder-decoder models can be trained end-to-end using a variety of loss functions, such as cross-entropy loss or reinforcement learning. During training, the model learns to generate output sequences that are close to the target sequences based on the chosen loss function.

Overall, encoder-decoder models have proven to be a powerful tool in natural language processing and have achieved impressive results on tasks such as machine translation and text summarization. The transformer-based architecture, in particular, has become a popular choice for encoder-decoder models due to its strong performance and ability to capture long-range dependencies in the input and output sequences.

T5 The T5 (Text-to-Text Transfer Transformer) [129] model is a transformer-based encoder-decoder architecture that was introduced in 2019. T5 is trained in a text-to-text format, where both the input and output are text sequences. This allows T5 to perform a wide range of tasks, including language modeling, text classification, and machine translation, among others.

T5 is further fined-tuned on predicting the output sequence from the input sequence, given a task-specific prefix that is added to the input sequence. During fine-tuning, the model presents a range of input-output pairs, each with a unique prefix, and learns to generate the corresponding output sequence. Using a prefix ensures that the model knows which task it is performing, allowing it to generalize to new tasks by simply adding a new prefix.

T5 uses a variant of the transformer architecture called the T5-Base, which consists of 12 encoder and decoder layers with 768 hidden units and 12 attention heads. In addition to the base model, T5 also includes larger models such as T5-Large, T5-3B, and T5-11B, with up to 11 billion parameters.

BART The BART (Bidirectional and Auto-Regressive Transformer) [85] model is another transformer-based encoder-decoder architecture introduced by Facebook AI Research in 2019. Like other encoder-decoder models, BART consists of an encoder network that generates a fixed-size representation of the input sequence and a decoder network that generates the output sequence based on the encoder representation. However, BART differs from other models in its pre-training and fine-tuning procedures.

BART is pre-trained using a denoising autoencoder objective, where the model is trained to reconstruct a noisy version of a text sequence. This involves randomly masking some tokens in the input sequence and training the model to predict the original sequence from the masked version. Additionally, BART is trained using both bidirectional and left-to-right language modeling objectives, allowing it to capture both global and local context information.

BART also uses a different fine-tuning procedure than other models. Instead of fine-tuning the entire model on a specific task, BART fine-tunes only the decoder network while keeping the encoder fixed. This is done by adding a task-specific output layer on top of the decoder network and fine-tuning it on the task-specific data. This approach is known as "discriminative fine-tuning" and has improved performance on downstream tasks while reducing the risk of overfitting.

BART uses a transformer architecture similar to that of the T5 model, with both base and large versions available. The base version consists of 12 encoder and decoder layers with 768 hidden units and 12 attention heads, while the large version consists of 12 encoder and decoder layers with 1024 hidden units and 16 attention heads.

2.1.4 Decoder-only Models

In contrast to encoder-only and encoder-decoder models, decoder-only models are designed to generate text without using an encoder network. Instead, these models use a large decoder network to generate text directly based on a given prompt or conditioning information.

Decoder-only models are trained using a variant of the autoregressive language modeling objective. The model is trained to predict the next token in a text sequence given the preceding tokens. However, unlike encoder-decoder models, decoder-only models do not require an encoder to generate the conditioning information for the decoder. Instead, the conditioning information is typically provided as a fixed-size vector or a sequence of tokens.

Decoder-only models have the advantage of being computationally efficient since they do not require an encoder network. However, they may struggle with tasks that require the model to generate text based on complex input sequences or long-term dependencies.

GPT One popular type of decoder-only model is GPT (Generative Pre-trained Transformer) [128], introduced by OpenAI in 2018. GPT uses a transformer architecture similar to that of the encoder-decoder models but with only a decoder network. The model is pre-trained on a large corpus of text using the autoregressive language modeling objective and can then be fine-tuned on various natural language processing tasks.

GPT is highly effective at various natural language generation tasks, including text completion, text generation, and dialog generation. GPT-3 contains 175 billion parameters, making it the largest and most powerful language model to date.

2.2 Learning Objectives

Cross-Entropy Cross-entropy is a commonly used loss function in machine learning, particularly for classification tasks. Given a set of predicted probabilities $\hat{y}_i$ and a set of true probabilities y_i, the cross-entropy between these two probability distributions is defined as:

$$Hy, \hat{y} = - \mathop{}_{i=1}^{n} y_i \log \hat{y}_i \tag{2.1}$$

, where n is the number of classes in the classification problem. The cross-entropy measures the difference between the predicted probabilities and the true probabilities, with a lower value indicating better agreement between the two distributions.

In the context of neural network training, cross-entropy is used as a loss function to optimize the model's parameters. For example, in a classification problem with k classes, the cross-entropy loss for a single training example is defined as:

$$Ly, \hat{y} = - \mathop{}_{i=1}^{k} y_i \log \hat{y}_i \tag{2.2}$$

, where y_i is a one-hot vector representing the true class label, and $\hat{y}_i$ is a vector of predicted class probabilities output by the model. The overall cross-entropy loss for a batch of training examples is typically calculated as the mean of the individual losses:

$$\mathcal{L}\theta = \frac{1}{m} \mathop{}_{i=1}^{m} Ly^i, \hat{y}^i \tag{2.3}$$

where θ represents the model's parameters, and m is the batch size.

In practice, cross-entropy is a widely used loss function in deep learning. It is often used in combination with other techniques, such as gradient descent optimization and regularization, to train neural networks on a variety of tasks.

Triplet Loss Triplet loss is a type of loss function used in metric learning, which aims to learn a mapping from input data to a metric space such that the distance between points in the metric space corresponds to their similarity in the input space. Triplet loss is commonly used in tasks such as face recognition, image retrieval, and person re-identification.

In triplet loss, the model is trained using triplets of samples: an anchor sample, a positive sample, and a negative sample. The goal of the model is to learn embeddings such that the distance between the anchor and the positive sample is minimized while the distance between the anchor and the negative sample is maximized. The triplet loss is defined as follows:

$$L_{triplet} = \max 0, da, p - da, n \ \ \alpha \tag{2.4}$$

, where a, p, and n are the embeddings of the anchor, positive, and negative samples, respectively, $d\cdot, \cdot$ is a distance function such as Euclidean distance or cosine similarity, and α is a margin that defines the minimum distance that should be maintained between the anchor and negative samples.

In practice, triplet loss can be challenging to train, as it requires a careful selection of triplets to ensure that the loss is informative and not trivially satisfied. One common approach is to use hard negative mining, where the negative sample is selected to be the most difficult sample that violates the triplet constraint. Another approach is to use semi-hard negative mining, where the negative sample is selected to be close to the anchor but still violates the triplet constraint.

2.3 NLP Tasks

Text Classification Text classification is a fundamental task in natural language processing (NLP) that involves assigning predefined categories or labels to a given input text. Text classification has many real-world applications, such as spam filtering [94], sentiment analysis [119], and topic classification [123].

In text classification, the input text is usually represented as a sequence of tokens, such as words or subwords, and the task is to predict the most appropriate label or category for the input text from a predefined set of labels. The labels can be binary, such as positive or negative sentiment, or multiclass, such as different types of news articles.

To train a text classification model, we need a labeled dataset that includes a set of input texts and their corresponding labels. The dataset is usually divided into three sets: a training set, a validation set, and a test set. The training set is used to optimize the model parameters, the validation set is used to tune the hyperparameters of the model and prevent overfitting, and the test set is used to evaluate the final performance of the model on unseen data.

Question-Answering Question answering (QA) is a task in natural language processing (NLP) that involves answering a given question based on a given context or document.

In QA, the input consists of a question and a context, which is usually a paragraph or a document that contains the information necessary to answer the question. The task is to extract the answer from the context. QA can be categorized into two types: extractive and abstractive. In extractive QA, the answer is a span of text from the context, while in abstractive QA, the answer is free-form and is generated based on the context and the question. QA can be formulated as a machine learning task, where the model is trained on a dataset of question-answer pairs.

Semantic Role Labeling Semantic role labeling (SRL) [105] is a task in natural language processing (NLP) that involves identifying the semantic roles played by different entities and events in a sentence. The task of SRL is to assign a semantic label to each constituent of a sentence, such as the subject, object, and predicate. These labels indicate the role of each constituent in the underlying event or action described by the sentence.

SRL is typically formulated as a supervised machine learning task, where the model is trained on a dataset of annotated sentences. The input to an SRL model is a sentence, and the output is a set of semantic labels that correspond to the roles of the constituents in the sentence. For example, in the sentence "John ate the pizza with a fork", the subject "John" is assigned the semantic role of agent, the object "the pizza" is assigned the semantic role of patient, and the prepositional phrase "with a fork" is assigned the semantic role of the instrument.

2.4 Benchmarks

2.4.1 Question-Answering

There are several benchmarks for QA in NLP, which are used to evaluate the performance of different QA models.

Stanford Question Answering Dataset (SQuAD) [132]: In our research, we use SQuAD as part of our neural learning process. The Stanford Question Answering Dataset (SQuAD) is a benchmark for QA in NLP that consists of a set of questions and their corresponding context passages. The dataset contains over 100,000 question-answer pairs, and the questions are designed to test the model's ability to understand the context and extract the relevant information to answer the question.

2.4.2 Entity Tracking

Propara [32] This dataset was created as a benchmark for procedural text understanding to track entities at each step of a process. Propara contains 488 paragraphs and 3,300 sentences with annotations that are provided by crowd-workers. The annotations (81,000) are the location of entities at each step of the process. The location can be either the name of the location, an unknown location, or specified as non-existence.

NPN-Cooking [16] This is a benchmark containing textual cooking instructions. Annotators have specified the ingredients of each recipe alongside the changes happening on each ingredient during the steps of the instructions. These changes are reported in categories such as location, temperature, cleanliness, and shape. We evaluate our model on the location prediction task of this benchmark, which is the hardest task due to having more than 260 candidate answers. We do not use the candidates to find the locations in our setting; Instead, we find a span of the text as the final location answer. This is a relatively harder setting but more flexible and generalizable than the classification setting.

2.4.3 Procedural Summarization

RecipeQA RecipeQA [184] is a dataset designed for multimodal comprehension of cooking recipes. It contains more than 36,000 question-answer pairs that are automatically generated from around 20,000 unique recipes with step-by-step instructions and images. The questions in RecipeQA require a joint understanding of multiple modalities, such as titles, descriptions, or images, and solving them involves capturing the temporal flow of events and making sense of procedural knowledge. The dataset provides a unique opportunity to study the ability of models to reason about procedural knowledge and to perform complex reasoning tasks that require multimodal comprehension.

CHAPTER 3

ENTITY TRACKING

In this chapter, we delve deeper into the task of entity tracking. Entity tracking aims to understand the evolution of entities inside a process. This can be studied by following the important properties of each entity after each action in the process. We provide a formal definition of the task in Section 3.1.

This task poses unique challenges compared to the conventional reading comprehension task due to the dynamic nature of the world described in the text. We will discuss some of the main challenges of this task in Section 3.2.

Our study on the entity tracking task relies on the two benchmarks of Propara [32] and NPN-Cooking [16], both of which provide fine-grained annotation for the properties of entities after each step of the process. The evaluation of this task goes beyond the simple F1 accuracy measures for directly comparing the model decisions and ground truth and proposes the evaluation of high-level abstraction of the process, such as its conversions, inputs, and outputs. We discuss this in more detail in Section 3.3.

As the main goal of this thesis is to investigate approaches to benefit from the available semantic structures for the task, we propose two approaches to enhance the pre-trained language models with knowledge about the relative time frame of events and semantic relations between entities and actions. We investigate the former in Section 3.5 and the latter in Section 3.6.

3.1 Task Definition

An example of a procedural text is shown in Figure 3.1. The example is taken from the Propara [32] dataset and shows the process of oil creation. At each row, the first column is the list of the sentences, each of which forms one step of the procedure. The second column contains the number of steps in the process, and the rest are the entities interacting in the process and their location at each step. The location of entities at step 0 is their initial location, which is not affected by this process. If an entity has a known or unknown location (specified by "?") at step 0, we call it an input.

Process	Participants			
Sentences	**plant**	**animal**	**bone**	**oil**
Before the process begins	?	?	-	-
1.Plants and animals die in a watery environment	watery environment	watery environment	-	-
2.Over time, sediments build over	sediment	sediment	-	-
3.the body decomposes	sediment	-	sediment	-
4.Gradually buried material becomes oil	-	-	-	sediment

Figure 3.1 An example of procedural text and its annotation (location of objects). '-' means the entity does not exist; '?' means the entity's location is unknown.

The procedural reasoning task can be formally defined by a procedural text including m steps, $S = \{s^1, s^2, ..., s^m\}$, a set of entities $E = \{e_1, e_2, ..., e_n\}$, where n is the number of entities and a set of properties. Specifically, in the Propara dataset, the property of interest is only the location of the entities $P_L = \{{p_L}_1^0, {p_L}_1^1, ..., {p_L}_n^m\}$, where ${p_L}_i^t$ denotes the jth entity at step t. In Propara, the location prediction starts at step 0, which indicates the entity's location before the process begins. The location of an entity can either be known (represented by a string) or unknown (represented by "?").

Similar to prior research [170], the location property is used to infer a set of actions $A = \{a_1^1, a_2^1, .., a_n^m\}$, where a_t^j denotes the action type applied to entity j at step t.

3.2 Challenges

Inferring actions and their impact on entities involved in a procedural text can be challenging in various aspects. **First**, there are dependencies between steps to be considered in predicting a plausible action set. For instance, an entity destroyed at step t of the process cannot be moved again at step t 1. **Second**, some sentences contain ambiguous local signals by including multiple action verbs. For example, "The oxygen is consumed in the process of forming carbon dioxide.",

Step	Entity	Action	Before	After
1	Water	Move	Root	Leaf
2	Water	Destroy	Leaf	-
1	Sugar	Create	-	Leaf
2	Sugar	None	Leaf	Leaf

Table 3.1 A sample annotation available in the Propara dataset, which is directly used for the document-level evaluation.

where the oxygen is being destroyed, and the carbon dioxide is being created. **Third**, the sentences are incomplete in some steps. For instance, a step of the process might only indicate "is buried in mud", which cannot be understood without context. **Fourth**, finding the properties of some entities may require reasoning over both the global context and local relations. For instance, in the sentences “1. Magma rises to the surface. 2. Magma cools to form lava”, the location of ‘Lava’ after step 2 should be inferred from the prior location of Magma, which is indicated in its previous step. **Fifth**, common sense is required to understand some consequences. For example, in Figure 3.1, step 3, one should use common sense to realize that ‘decomposing body’ would expose the ‘bones’, which will be left behind in the ‘sediment’. **Sixth**, understanding some relations requires an advanced co-reference resolution. In Figure 3.1, step 4, a complex co-reference resolution is required to understand that the ‘buried material’ refers to both ‘plants and animals bones’ and that they are transforming into the ‘oil’.

3.3 Task Evaluation

Sentence-level evaluation is introduced in [32] for Propara dataset. This evaluation focuses on the following three categories.

- **Cat1** Is e created (destroyed/moved) during the process?
- **Cat2** When is e created (destroyed/moved) during the process?
- **Cat3** Where is e created (destroyed/moved from or to) during the process?

Document-level evaluation is a more comprehensive evaluation process and introduced later in [167] for the Propara benchmark. Currently, this is the default evaluation in the Propara leaderboard containing four criteria:

- **What are the Inputs?** Which entities existed before the process began and do not exist after the process ends.
- **What are the Outputs?** Which entities got created during the process?
- **What are the Conversions?** Which entities got converted to other entities?
- **What are the Moves?** Which entities moved from one location to another?

The document-level evaluation requires models to reformat their predictions in a tabular format as shown in Table 3.1. At each row of this table, for each entity at a specific step, we can see the action applied to that entity, the location of that entity before that step, and the location of the entity after that step. The action takes values from a predefined set including, “None”, “Create”, “Move”, and “Destroy”. The exact action can be specified based on the before and after locations.

3.4 Related Research

ScoNe [97], NPN-Cooking [16], bAbI [175], ProcessBank [13], and Propara [32] are benchmarks proposed to evaluate models on procedural text understanding. Processbank [13] contains procedural paragraphs mainly concentrated on extracting arguments and relations for the events rather than tracking the states of entities. ScoNe [97] aims to handle co-reference in a procedural text expressed about a simulated environment. bAbI [175] is a simpler machine-generated textual dataset containing multiple procedural tasks such as motion tracking, which has encouraged the community to develop neural network models supporting explicit modeling of memories [162, 146] and gated recurrent models [25, 64]. NPN-Cooking [16] contains recipes annotated with the state changes of ingredients on criteria such as location, temperature, and composition. Propara [32] provides procedural paragraphs and detailed annotations of entity locations and the status of their existence at each step of a process.

Some earlier models on procedural text understanding, inspired by bAbI dataset, are Memory Network architecture [162], Recurrent Relational network (RRN) [146], and gated recurrent models such as GRU [25] and Recurrent Entity Networks(EntNet) [64]. EntNet uses dynamic memory for the world’s hidden state, which will be updated based on a gated mechanism at each step. RRN augments neural networks with the capacity to do multi-step relational reasoning while keeping a memory of hidden states (separate memory per entity) at each step and updating that based on the memory representations and newly retrieved representations at each step.

Inspired by Propara and NPN-Cooking benchmarks, recent research has focused on tracking entities in a procedural text. Datasets such as Procedural Cyber-Security text [122] and OpenPI [170] are further designed with a similar task in mind.

To address the challenges in this type of reasoning over procedural text, various models are proposed. Query Reduction Networks (QRN) [151] performs gated propagation of a hidden state vector at each step. Neural Process Network (NPN) [16] computes the state changes at each step by looking at the predicted actions and involved entities. Prolocal [32] predicts locations and status changes locally based on each sentence and then globally propagates the predictions using a persistence rule. Proglobal [32] predicts the status changes and locations over the whole paragraph using distance values at each step and predicts current status based on current representation and the predictions of the previous step. ProStruct [167] aims to integrate manually extracted rules or knowledge-base information on VerbNet [150] as constraints to inject common-sense into the model. KG-MRC [35] uses a dynamic knowledge graph of entities over time and predicts locations with spans of the text by utilizing reading comprehension models. Ncet [61] updates entities representation based on each sentence and connects sentences together with an LSTM. To ensure the consistency of predictions, Ncet uses a neural CRF over the changing entity representations. XPAD [34] is also proposed to make dependency graphs on the Propara dataset to explain the dependencies of events over time. Most recently, DynaPro [6] feeds an incremental input to pre-trained LMs' question answering architecture to predict entity status and transitions jointly. Recent models have further investigated the integrating of common-sense (KOALA) [192], utilizing large generative language models (LEMON) [153], or using both the question answering setting and sequential structural constraints at the same time (CGLI) [99] to address this task.

Procedural reasoning has also been pursued within the multi-modality domain [184, 131, 5] which has additional challenges of aligning the representation spaces of different modalities.

Additionally, recent research has alsop investigated the causality of past events in enabling future events, thorough different datasets such as WIQA [169] and Trip [160] for story understanding.

3.5 Encoding Temporal information

The current language models convey rich linguistic knowledge and can serve as a strong basis for solving various NLP tasks [95, 43, 187]. That is why most of the state-of-the-art models on procedural reasoning are also built based on current language models [6, 61]. However, they do not contain a dedicated representation to understand the relative time of actions when encoding a procedural text. Here, we propose a new approach for feeding the procedural information into LMs in a way that the LM-based QA models are aware of the taken steps and can answer the questions related to each specific time-frame in the procedure.

We propose the Time-Stamped Language model (TSLM model), which uses timestamp embedding to encode past, current, and future time of events as a part of the input to the model. TSLM utilizes timestamp embedding to answer differently to the same question and context based on different steps of the process. As we do not change the portion of the input manually, our approach enables us to benefit from the pre-trained LMs on other QA benchmarks by using their parameters to initialize our model and adapt their architecture by introducing a new embedding type. Here, we use RoBERTa [95] as our baseline language model.

We evaluate our model on two benchmarks, Propara [32] and NPN-Cooking [16]. Propara contains procedural paragraphs describing a series of events with detailed annotations of the entities along with their status and location. NPN-Cooking contains cooking recipes annotated with their ingredients and their changes after each step in criteria such as location, cleanliness, and temperature.

TSLM differs from previous research as its primary focus is on using pre-trained QA models and integrating the flow of events in the global representation of the text rather than manually changing the part of the input fed to the model at each step. In contrast to DynaPro [6], we explicitly inject past, current, and future timestamps into the language models input and implicitly train the model to understand the events' flow rather than manually feeding different portions of the context at each step.

TSLM outperforms the state-of-the-art models in nearly all metrics of two different evaluations defined on the Propara dataset. Results show a 3.1% F1 score improvement and a 10.4% improvement in recall. TSLM also achieves the state-of-the-art result on the location accuracy on the NPN-Cooking location change prediction task by a margin of 1.55%.

In summary, our contribution is as follows:

- We propose Time-Stamped Language Model (TSLM model) to encode the meaning of past, present, and future steps in processing a procedural text in language models.
- Our proposal enables procedural text understanding models to benefit from pre-trained LM-based QA models on general-domain QA benchmarks.
- TSLM outperforms the state-of-the-art models on the Propara benchmark on both document-level and sentence-level evaluations. TSLM improves the performance state-of-the-art models on the location prediction task of the NPN-Cooking [16] benchmark.
- Improving over two different procedural text understanding benchmarks suggests that our approach is effective, in general, for solving the problems that require the integration of the flow of events in a process.

3.5.1 Model Components

QA Adaptation To predict the status and the location of entities at each step, we model F with a question-answering setting. For each entity e, we form the input Q_e as follows:

$$\begin{aligned} Q_e =&\text{[CLS] Where is } e\text{? [SEP]} \\ & s_1 \text{ [SEP] } s_2 \text{ [SEP] ..., } s_n \text{ [SEP]} \end{aligned} \tag{3.1}$$

Although Q_e is not a step-dependent representation and does not incorporate any different information for each step, our mapping function needs to generate different answers for the question "Where is entity e?" based on each step of the procedure.

		Participants				
Paragraph	State number	Water	Light	CO2	Mixture	Sugar
(Before the process starts)	State 0	Soil	Sun	?	-	-
Roots absorb water from soil	State 1	Root	Sun	?	-	-
The water flows to the leaf	State 2	Leaf	Sun	?	-	-
Light from the sun and CO2 enter the leaf	State 3	Leaf	Leaf	Leaf	-	-
Water, light, and CO2 combine into mixture	State 4	-	-	-	Leaf	-
Mixture forms sugar	State 5	-	-	-	-	Leaf

Table 3.2 An example of procedural text and its annotations from the Propara dataset [32]. "-" means the entity does not exist. "?" means the location of the entity is unknown.

For instance, consider the example in Table 3.2 and the question "Where is water?"; Our model should generate different answers at four different steps. The answer will be “root”, “leaf”, “leaf”, “non-existence” for steps 1 to 4, respectively.

To model this, we create pairs of Q_e, t_i for each $i \in \{0, 1, ..., n\}$. For each pair, Q_e is timestamped according to t_i using $Timestamp.$ function described in Sec. 3.5.1 and mapped to an updated step-dependent representation, $Q_e^{t_i} = Timestamp Q_e, t_i$.

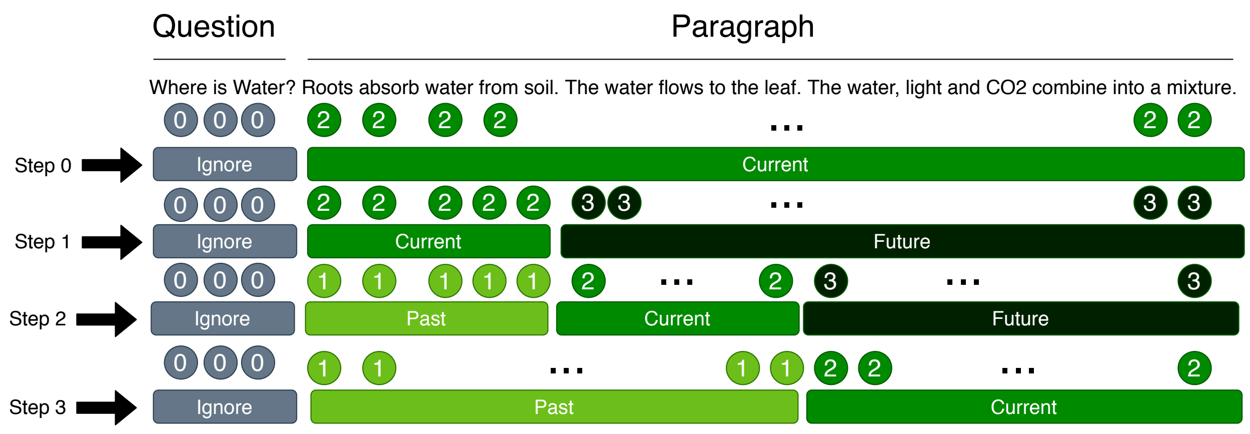
The updated input representation is fed to a language model (here ROBERTA) to obtain the step-dependent entity representation, $R_e^{t_i}$, as shown in Equation 3.2. We discuss the special case of $i = 0$ in more details in Sec. 3.5.1.

$$R_e^{t_i} = RoBERTa Q_e^{t_i} \tag{3.2}$$

We use the step-dependent entity representation, $R_e^{t_i}$, and forward it to another mapping function $g.$ to obtain the location and status of the entity e in the output. In particular, the output includes the following three vectors, a vector representing the predictions of entity status S, another vector for each token’s probability of being the start of the location span L, and a third vector carrying the probability of each word being the last token of the location span. The outputs of the model are computed according to Equation 3.3.

$$status, Start\_prob, End\_prob = g R_e^{t_i} \tag{3.3}$$

where R_e is the tokens’ representations output of RoBERTa [95], and $g.$ is a function we apply to the token representations to get the final predictions. We will discuss each part of the model



Question
Paragraph
Where is Water? Roots absorb water from soil. The water flows to the leaf. The water, light and CO2 combine into a mixture.
Step 0
Ignore
Current
Step 1
Ignore
Current
Future
Step 2
Ignore
Past
Current
Future
Step 3
Ignore
Past
Current

Figure 3.2 An example of timestamp embedding in a procedural text. The question is always ignored with the value "0". At each step i, the tokens from that step are paired with the “current” value, tokens from steps 0 to i are paired with the “past” value, and the tokens from step i to the last step are paired with the “future” value.

separately in the following sections.

Timestamp Embedding The timestamp embedding adds the step information to the input Q_e to be considered in the attention mechanism. The step attention is designed to distinguish between current (what is happening now), past (what has happened before), and future (what has not yet happened) information.

We use the mapping function $Timestamp.$ from the pair Q_e, t_i to add a number along with each token in Q_e and retrieve the step-dependent input $Q_e^{t_i}$ as shown in Figure 3.2. The Mapping function $Timestamp.$ integrates past, current, and future representations to all of the tokens related to each part. $Timestamp.$ function assigns the number 1 for the past, 2 for the current, and 3 for future tokens in the paragraph by considering one step of the process as the current event. These values are used to compute an embedding vector for each token, which will be added to its initial representation as shown in Figure 3.3. The special number 0 is assigned to the question tokens, which are not part of the process timeline. For predicting State 0 (The process inputs), we set all the paragraph information as the current step.

Status classification To predict the entities' status, we apply a linear classification module on top of the CLS token representation in R_e as shown in Equation 3.4.

$$Attribute = Softmax W^T Re_C \tag{3.4}$$

where Re_C is the representation of the CLS token which is the first token in R_e.

Span prediction We predict a location span for each entity for each step of the process as shown in Equation 3.5, we follow the popular approach of selecting start/end tokens to detect a span of the text as the final answer. We compute the probability of each token being the start or the end of the answer span. If the index with the highest probability to be the start token is $token_{start}$ and for the end token is $token_{end}$, the answer location will be $Location = Ptoken_{start} : token_{end}$.

$$\begin{aligned} \text{Start\_prob} &= \text{Softmax} W_{\text{start}}^T R_e^{t_i} \\ \text{End\_prob} &= \text{Softmax} W_{\text{end}}^T R_e^{t_i} \\ \text{token}_{\text{start}} &= \arg\max_i \text{Start\_prob} \\ \text{token}_{\text{end}} &= \arg\max_i \text{End\_prob} \end{aligned} \tag{3.5}$$

3.5.2 Training & Inference

Training We use the cross-entropy loss function to train the model. At each prediction for entity e at timestamp t_i, we compute one loss value $loss_{attribute}$ regarding the status prediction and one loss value $loss_{location}$ for the span selection. The variable $loss_{location}$ is the summation of the losses of the start token and the end token prediction, $loss_{location} = loss_{location_{start}} \; loss_{location_{end}}$. The final loss of entity e at time t_i is computed as in Equation 3.6.

$$Loss_i^e = loss_{i,attribute}^e \; loss_{i,location}^e \tag{3.6}$$

Inference At inference time, we apply two different post-processing rules on the outputs of the model. First, we impose that the final selected location answer should be a noun phrase in the

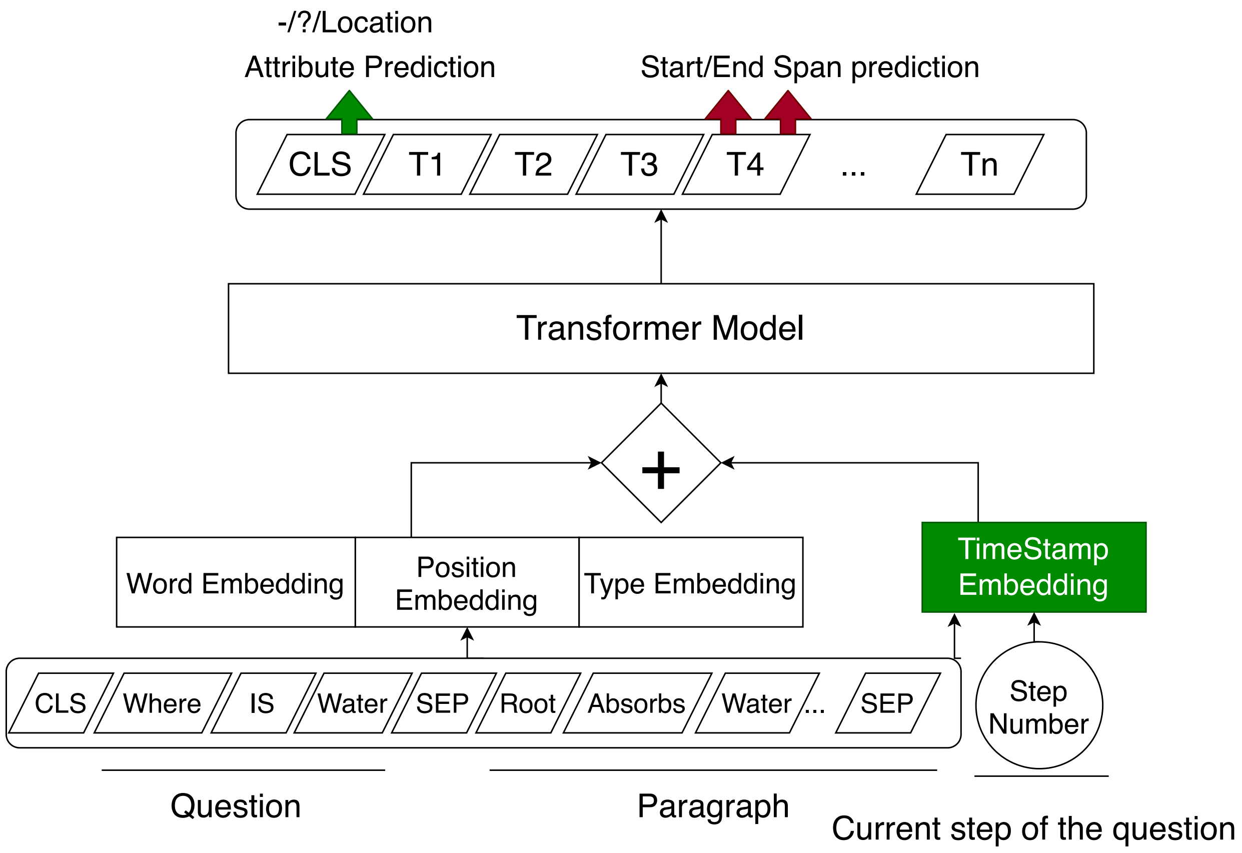


Figure 3.3 An overview of the proposed model. The “Timestamp Embedding” module is introduced in this work and the rest are taken from basic language model architecture.

original procedure. Considering that a location span is a noun phrase, we limit the model to do a $softmax$ over tokens of noun phrases in the paragraph to select the start and end tokens. Second, we apply consistency rules to make sure that our predicted status of entities is consistent. We define the two following rules:

- An entity can not be created if it has been already destroyed: *if $S_e^{t_i}$ is "non-existence" and $S_e^{t_{i1}}$ is unknown or known location, then for every step j, if $S_e^{t_j}$ is unknown or known location and $S_e^{t_{j1}}$ is "non-existence", then $i < j$.*
- An entity cannot be created/destroyed twice in a process: *if $S_e^{t_j}$ and $S_e^{t_i}$ are both "-", $S_e^{t_{j1}}$ and $S^{t_{i1}}$ are both either known or unknown location, then $i = j$.*

$S_e^{t_i}$ is the status of entity e at step t_i of the process.

	Sentence-level					Document-level		
Model	Cat1	Cat2	Cat3	Macro^{Avg}	Micro^{Avg}	P	R	F1
ProLocal [32]	62.7	30.5	10.4	34.5	34.0	**77.4**	22.9	35.3
ProGlobal [32]	63.0	36.4	35.9	45.1	45.4	46.7	52.4	49.4
EntNet [64]	51.6	18.8	7.8	26.1	26.0	50.2	33.5	40.2
QRN [151]	52.4	15.5	10.9	26.3	26.5	55.5	31.3	40.0
KG-MRC [35]	62.9	40.0	38.2	47.0	46.6	64.5	50.7	56.8
NCET [61]	73.7	47.1	41.0	53.9	54.0	67.1	58.5	62.5
XPAD [34]	-	-	-	-	-	70.5	45.3	55.2
ProStruct [167]	-	-	-	-	-	74.3	43.0	54.5
DYNAPRO [6]	72.4	49.3	**44.5**	55.4	55.5	75.2	58.0	65.5
TSLM (Our Model)	**78.81**	**56.8**	40.9	**58.83**	**58.37**	68.4	**68.9**	**68.6**

Table 3.3 Results from the sentence-level and document-level evaluation on Propara. Cat$i$ evaluations are defined in Section 3.5.3.1.

We do not apply an optimization/search algorithm to find the best assignment over the predictions according to the defined constraints. The constraints are only applied based on the order of the steps to ensure that the later predictions are consistent with the ones made before.

3.5.3 Experiments

Implementation Details We use the SGD optimizer implemented by Pytorch [124] to update the model parameters. The learning rate for the Propara implementation is set to $3 - e4$ and is updated by a scheduler with a 0.5 coefficient every 50 steps. We use $1 - e6$ as the learning rate and a scheduler with 0.5 coefficient to update the parameters every ten steps on the NPN-Cooking implementation. The implementation code is publicly available at GitHub[1].

We use RoBERTa [95] question-answering architecture provided by HuggingFace [178]. RoBERTa is pretrained with SQuAD [132] and used as our base language model to compute the token representations. Our model executes batches containing an entity at every step and makes updates based on the average loss of entities per procedure. The network parameters are updated after executing one whole example. The implementation code will be publicly available on GitHub after acceptance.

3.5.3.1 Evaluation

Propara: We have to process our (Status S, Location L) predictions at each step to generate a similar tabular format as in Table 3.1. We define r_e^i as a row in this table which stores the predictions related to entity e at step t_i. To fill this row, we first process the status predictions. If the status prediction S is either "-" or "?", we fill those values directly in the after location column. The before location column value of r_e^i is always equal to the after location column value of r_e^{i-1}. If the status is predicted to be a "Known Location", we fill the predicted location span L into the after location column of r_e^i.

The action column is filled based on the data provided in the before and after locations columns. If the before location is/isn't "-" and the after location is not/is "-", then the action is "Create"/"Destroy". If the before and after locations are equal, then the action is "None" and if the before and after locations are both spans and are different from each other, the action is "Move".

NPN-Cooking location changes: We evaluate our model on the NPN-Cooking benchmark by computing the accuracy of the predicted locations at steps where the locations of ingredients change. We use the portion of the data that has been annotated by the location changes to train and evaluate our model. In this evaluation, we do not use the status prediction part of our proposed TSLM model. Since training our model on the whole training set takes a very long time (around 20 hours per iteration), we use fewer samples for training. This practice is also used in other prior work [35].

3.5.3.2 Results

The performance of our model on Propara dataset [32] is quantified in Table 3.3. Results show that our model improves the SOTA by a 3.1% margin in the F1 score and improves the Recall metric with 10.4% on the document-level evaluation. On the sentence-level evaluation, we outperform SOTA models with a 5.11% in Cat1, 7.49% in Cat2, and by a 3.4% margin in the macro-average. We report Table 3.3 without considering the consistency rules and evaluate the effect of those in the ablation study in Sec. 3.5.3.3.

In Table 3.5, we report a more detailed quantified analysis of the TSLM model's performance

[1]https://github.com/HLR/TSLM

Model	Accuracy	Training Samples	Prediction task
NPN-cooking [16]	51.3	$\sim 83,000$ (all data)	Classification
KG-MRC [35]	51.6	$\sim 10,000$	Span Prediction
DynaPro [6]	62.9	$\sim 83,000$ (all data)	Classification
TSLM (Our Model)	63.73	$\sim 10,000$	Span Prediction
	64.45	$\sim 15,000$	Span Prediction

Table 3.4 Results on the NPN-Cooking benchmark. Both class prediction and span prediction tasks are the same but use two different settings, one selects among candidates, and the other chooses a span from the recipe. However, each model has used a different setting and a different portion of the training data. The information on the data splits was not available which makes a fair comparison hard.

Criteria	Precision	Recall	F1
Inputs	89.8	71.3	79.5
Outputs	85.6	91.4	88.4
Conversions	57.7	56.7	57.2
Moves	40.5	56	47

Table 3.5 Detailed analysis of TSLM performance on the Propara test set on four criteria defined in the document-level evaluation.

based on each different criterion defined in the document-level evaluation. Table 3.5 shows that our model performs best on detecting the procedure's outputs and performs worst on detecting the moves. Detecting moves is essentially hard for TSLM as it predicts outputs based on the whole paragraph at once. Outperforming SOTA results on the input and output detection suggest that the TSLM model can understand the interactions between entities and detect the entities which exist before the process begins. The detection of input entities is one of the weak aspects of the previous research that we improve here.

A recent unpublished research [192] reports better results than our model. However, their primary focus is on common-sense reasoning, and their goal is orthogonal to our main focus in proposing the TSLM model. Such approaches can be later integrated with TSLM to benefit from common-sense knowledge on solving the Propara dataset.

The reason that TSLM performs better at *recall* and worse at *precision* is that our model looks at the global context, which increases the recall and lowers the precision when local information is strongly important. The same phenomenon (better recall) is observed in ProGlobal, which also considers global information as we do, compared to ProLocal.

Table 3.4 shows our results on the NPN-Cooking benchmark for the location prediction task. Results are computed by only considering the steps that contain a location change and are reported by computing the accuracy of predicting those changes. Our results show that TSLM outperforms the SOTA models with a 1.55% margin on accuracy even after training on 15,000 training samples. To be comparable with the KG-MRC [35] experiment on NPN-Cooking which is only trained on 10k samples, we report the performance of our model trained on the same number of samples, where TSLM gets a 12.1% improvement over the performance of KG-MRC [35].

3.5.3.3 Ablation Study

To evaluate the importance of each module one at a time, we report the performance of the TSLM by removing the noun-phrase filtering at inference, the consistency rules, timestamp embedding, SQuAD [132] pre-training, and by replacing RoBERTa [95] with BERT [43]. These variations are evaluated on the development set of the Propara dataset and reported in Table 3.6. As stated before and shown in Table 3.6, it is impossible to remove the timestamp embedding as that is the only part of the model enabling changes in the answer at each step. Hence, by removing that, the model cannot converge and yields a 25% decrease in the F1 score. The simple consistency and span filtering rules are relatively easy to be learned by the model based on the available data, therefore adding those does not affect the final performance of the model.

TSLM$_{\text{BERT}}$ experiment is designed to ensure a fair comparison with previous research [6] which has used BERT as their base language model. The comparison of *TSLM$_{BERT}$* to *-SQuAD Pre-training* and *- Timestamp Embedding* in Table 3.6 indicates that using RoBERTa instead of BERT is not as much important as our main proposal (using Time-stamp encoding) in TSLM model. Also, TSLM$_{\text{BERT}}$ achieves 66.7% F1 score on the Propara test set, which is 1.2% better than the current SOTA performance.

By removing the SQuAD pre-training phase, the model performance drops with a 10.6% in the F1 score. This indicates that despite the difference between procedural text understanding and the general MRC tasks, it is quite beneficial to design methods that can transfer knowledge from other QA data sources to help with procedural reasoning. This is crucial as annotating procedural texts

Model	P	R	F1
$\text{TSLM}_{\text{RoBERTa}}$	72.9	74.1	**73.5**
- constraints	73.8	73.3	**73.5**
- noun-phrase filtering	73.5	73.3	73.4
- SQuAD Pre-training	78.8	52.2	62.8
- Timestamp Embedding	94.6	32.6	48.5
$\text{TSLM}_{\text{BERT}}$	69.2	73.5	71.3

Table 3.6 Ablation study results on the Propara document-level task development set. "- constraints", "- Span filtering", and "- Timestamp Encoding" show our model performance while removing those modules. *-SQuAD Pre-training* is when we do not pre-train our base language model on SQuAD. $\text{TSLM}_{\text{BERT}}$ is when we use BERT as the base language model.

is relatively more expensive and time-consuming.

3.5.4 Discussion

We provide more samples to support our hypothesis in solving the procedural reasoning task and answer some of the main questions about the ideas presented in the TSLM model.

Why is the whole context important? The main intuition behind TSLM is that the whole context, not just previous information, matters in reasoning over a process. Here, we provide some samples from Propara to show why this intuition is correct. Consider this partial paragraph, "Step i: With enough time the pressure builds up greatly. Step i 1: The resulting volcano may explode.". Looking at the annotated status and location, the "volcano" is being created at Step i without even being mentioned in that step. This is only detectable if we look at the next step saying "The resulting Volcano...".

As another example, consider this partial paragraph: "Step i: Dead plants form layers called peat. ... Step i 3: Pressure squeezes water out of the peat.". The annotation indicates that the location of "water" is being changed to "peat" at step i, which is only possible to detect if the model is aware of the following steps indicating that the water comes out of the peat.

Positional Embedding VS Time-stamp encoding: As mentioned before, the whole context (future and past events) is essential for procedural reasoning at a specific step. However, the reasoning should focus on one step at a time, given the whole context. While positional encoding encodes the order of information at the token level for reasoning over the entire text, we need another level of encoding to specify the steps' positions (boundaries) and, more importantly, to indicate the step that the model should focus on when answering a question.

Advantages/Disadvantages of TSLM model: TSLM integrates higher-level information into the token representations. This higher-level information can come from event-sequence (time of events), sentence-level, or any other higher source than the token-level information. The first advantage of TSLM is that it enables designing a model which is aware of the whole context, while previous methods had to customize the input at each step to only contain the information of earlier steps. Furthermore, using TSLM enables us to use pretrained QA models on other datasets without requiring us to retrain them with the added time-stamped encoding. One main disadvantage of the TSLM model, which is natural due to the larger context setting in this model, is not being sensitive to local changes, which is consistent with the observation in the comparison between ProGlobal and ProLocal models.

3.6 Exploiting Semantic Parsers to Understand the Process

We discussed a series of challenges for the procedural reasoning task in Section 3.2. Except for the common-sense [192] and the ability to make consistent global decisions actions [61], the other challenges might have only been indirectly tackled in the recent research [68, 49], but have neither been addressed explicitly nor properly evaluated to measure their success on resolving these challenges. Here, we evaluate whether semantic parsers can alleviate some of these challenges. Semantic parsers provide semantic frames identifying predicates and their arguments in a sentence. For instance, in the sentence 'Move bag to the yard', "Move" is the predicate, "bag" and "the yard" are the arguments with types "affected"[2] and "location" respectively. Such semantic information can help disambiguate multi-verb local connections between predicates and arguments [68]. They can also provide meaningful local relations, making it easier to connect global information to infer entities' states. For instance, in the same sentence, "Magma cools to form lava", "Magma" is noted as the 'affected' and 'lava' is the result of the predicate 'form'. This makes it easier to infer that the location of 'lava' should match the last location of 'magma'.

[2]referred to as 'Patient' in some other parsing formalisms.

For our study, we consider both the classic semantic role labeling (SRL) [3], based on [152], which is a relatively shallow semantic parsing model, as well as the deep semantic parser TRIPS[4] [53, 4]. To investigate the effect of semantic parsing on procedural reasoning, we analyze its effect as a standalone symbolic model as well as its integration in a neuro-symbolic model that combines semantic parsing with state-of-the-art neural models to solve the procedural reasoning task.

First, we design a set of heuristics to extract a symbolic abstraction from the TRIPS parser, called PROPOLIS. We use this baseline to further showcase the effectiveness of semantic parsing information in solving the procedural task. Next, we integrate the semantic parsers with two well-established procedural reasoning neural backbones, namely NCET [61] and TSLM [49] (and its extension CGLI [99]), through encoding the semantic relations as a graph attention neural network (GAT) [154].

For our experiments, we use Propara dataset [170] that introduces the procedural reasoning task over natural events that are described in English. We realized the existing evaluation metrics of this dataset do not reflect the actual performance of the models and fail to identify the challenges and shortcomings of the models. Consequently, we propose new evaluation criteria to shed light on the differences between the models, even when they perform similarly based on the prior metrics.

In summary, our contributions are (1) Proposing a symbolic model (Propolis) to solve the procedural reasoning task based on semantic parsing, (2) Proposing a set of new evaluation metrics that can identify the strengths and weaknesses of the models, and (3) Showcase the benefits of integrating semantic parsing into the neural models. The code and models proposed in this work are all available in GitHub [5].

3.6.1 Semantic Parsing

We investigate two different modeling approaches to solve this problem. First, we use a symbolic and parsing-based model, and second, we integrate semantic parsing with neural models. We use two different sources for semantic extraction: SRL and TRIPS.

[3]https://demo.allennlp.org/semantic-role-labeling
[4]http://trips.ihmc.us/parser/cgi/parse
[5]https://github.com/HLR/ProceduralSemanticParsing

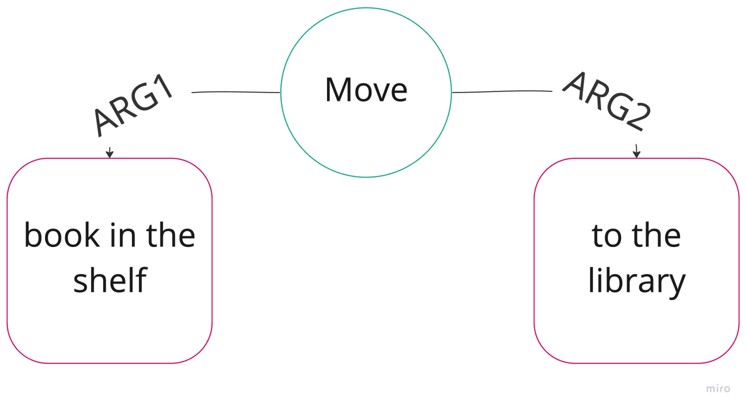


Figure 3.4 The Semantic Role Labeling annotation for the sentence “Move the book in the shelf to the library” represented as a graph.

In general, SRL is coarse-grained and shallow compared to TRIPS. The connections in TRIPS are not limited to the pairwise connections between predicates and arguments but are extended to the semantic connections between any two words. Since TRIPS relies on a general purpose ontology, it also augments the arguments and predicates with additional information about a set of possible features (mobility, container, negation) and mapping of the words to hierarchical ontology classes (i.e., mapping “water” to “beverage”). SRL is centered around the semantic frames of the verbs (predicates) and identifies each predicate’s main and adjunct (mainly time and location) arguments in the sentence. Figure 3.4 and 3.5 show examples of the SRL and TRIPS parses, respectively.

The symbolic model only uses the TRIPS parser as it provides more extended extractions and meaningful relations, while both SRL and TRIPS are used for integration with the neural baselines.

3.6.2 PROPOLIS: Symbolic Procedural Reasoning

We propose the PROPOLIS model, which solves the procedural reasoning task merely by symbolic semantic parsing. PROPOLIS operates on the TRIPS parser in three steps. First, it makes an abstraction over the original parse to summarize the information in the graph and include a smaller set of actions and changes in objects and their locations. Second, it uses a set of rules to transform the abstracted parses into clear actions and identifies the affected objects by the actions, using the semantic roles, while extracting an ending location or starting location. Lastly,

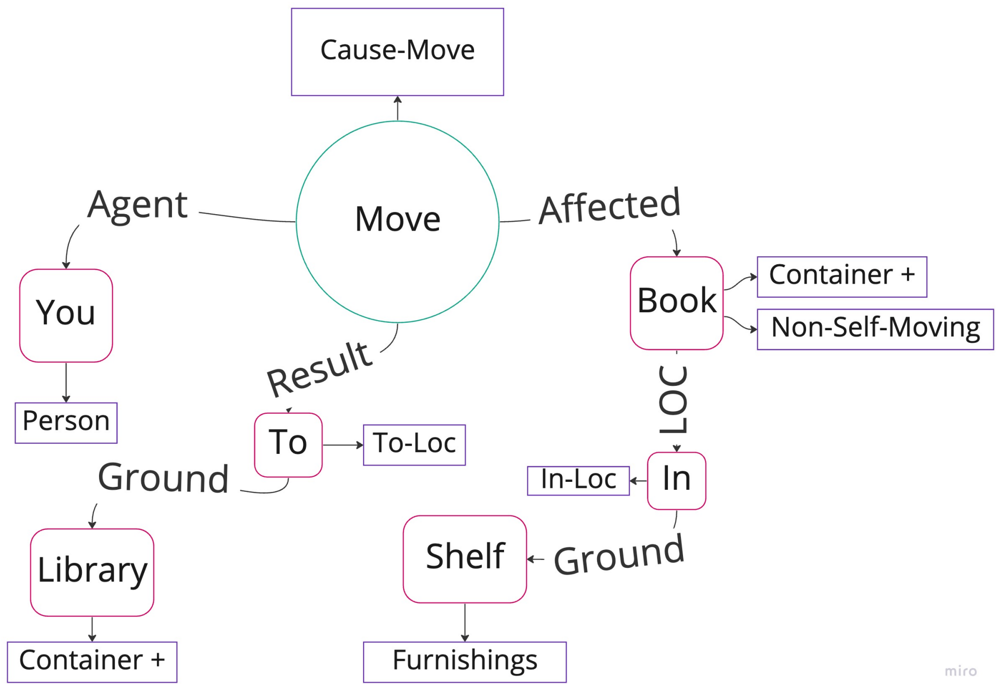


Figure 3.5 The TRIPS parse for the sentence “Move the book in the shelf to the library” represented as a graph with named edges.

it performs global reasoning to connect the local decisions and produce a consistent sequential set of actions/locations for each entity of interest.

3.6.2.1 Graph Abstraction

The original TRIPS parse includes many concepts and edges that do not directly affect entities’ location or existence. Therefore, we make a more concise graph abstraction to facilitate processing the entities, actions, and locations. To obtain a more informative abstraction, firstly, the relevant classes of the TRIPS ontology are mapped to action classes defined in the Propara dataset (Create, move, or destroy). For instance, the verb ‘flow’ is first mapped to the ‘fluidic motion’ class in the TRIPS ontology, which is a child of the ‘motion‘ class, and the ‘motion’ class is mapped to the ‘move’ action in the Propara dataset. This will help distinguish the predicates that signal a change in the location or existence of objects. Second, the important arguments are identified in the parse, and the locations are extracted. The graph is decomposed to include a set of events with their arguments. Each event may contain different roles such as "agent", "affected", "result", "to_location", "from_location", or other roles required by its semantic frame.

Main Predicate	Roles	Decisions
Move	Affected, Agent	The "Affected" is being moved.
Move	Agent	The "Agent" is being moved.
Destroy	Affected	The "Affected" is being destroyed
Create	Affected_Result, Affected	The "Affected_Result" is being created
Create	Affected	The "Affected" is being created
Change	Affected, Res	The "Affected" is being destroyed, and the "Res" is being created

Table 3.7 The list of rules used to evaluate the effect of actions on various roles of the semantic frame.

Because the TRIPS system handles much of the variation expected in sentence constructions, we can use a relatively compact specification for defining the events and relationships of interest while coping with fairly complex and nested formulations. We capitalized on the TRIPS ontology and parser to develop a compact and easy-to-maintain specification of event extraction rules. Instead of having to write one rule to match each keyword/phrase that could signify an event, many of these words/phrases have already been systematically mapped to a few types in the TRIPS ontology. For instance, demolish, raze, eradicate, and annihilate are all mapped to the TRIPS ontology type "ONT::DESTROY". In addition, the semantic roles are consistent across different ontology types. The parser handles various surface structures, and the logical form contains normalized semantic roles. For example, in the following sentence:

- The bulldozer demolished the building
- The building was demolished
- The demolition of the building
- Building demolition

, all the parses result in the same basic logical form with the semantic roles "AFFECTED: the building" and, where applicable, "AGENT: the bulldozer". Thus, we needed very few extraction rule specifications for each event type, covering a wide range of words and syntactic patterns.

3.6.2.2 Rule-based Local Decisions

We use a set of heuristic rules to map the abstracted graph onto actual actions over the entities of interest. The rules are written according to the semantic frames and the type of predicates and arguments in each parse. For instance, if a semantic frame is mapped to ‘Move’ and has both the ‘agent’ and ‘affected’ arguments, then the ‘affected’ argument specifies the object being moved. The same frame with only an ‘agent’ argument indicates a move for the object in the ‘agent’ role. Table 3.7 shows the most frequent templates we used to transform the local parses into actual decisions over the entities. To handle the location arguments from the parses, we also consider the two cases on ‘from_loc’ and ‘to_loc’. In the specific case of a “destroy” event, any location attached to the semantic frame is considered the ‘from_loc’ for the item being destroyed.

3.6.2.3 Global Reasoning

The two first steps are merely based on the local sentence-level actions of each step. We need additional global reasoning over the whole procedure to predict the outputs. Global reasoning ensures that local decisions form a valid global sequence of actions for a given entity. For instance, if an entity is predicted to be destroyed at step 2 and moved at step 3, we consider the ‘destroyed’ action a wrong local decision since a destroyed object cannot move later in the process. The graph also contains passive indications of object location in phrases such as "the book on the shelf" or even indications of prior locations in terms of a ‘from_location’ argument. These phrases do not generate actions but provide information that should be used in previous steps. For example, if step t has a local prediction ‘Move’ for entity e with no target location and step t 1 has a ‘from_location’ for entity e, then the ‘from_location’ should be used as the target location of the ‘Move’ action in the previous step.

To perform the global reasoning over the local predictions, we first do a forward pass through the actions and location predictions and ensure they are globally consistent. To do so, we start from the first predicted action and check the following on every next step prediction:

- If the current action is None, then we skip this step!

- If the last observed action is “Create” or “Move”,
  - If the current action in “Create” and the location of this action is the same as the last observed location, then the new “Create” action is transformed to “None”.
  - IF the current action is “Create” and the location of this action is different from the last observed location, then the new action is changed to “Move”.
  - Otherwise, the new action is kept the same, and the last observed action is updated.
- If the last observed action is “Destroy”,
  - If the current action is “Destroy” and it has a location different from the last observed location, then the action is changed to “Move”.
  - If the current action is “Destroy” and it has a location similar to the last observed location, then the action is changed to “None”.
  - Otherwise, the new action is kept the same, and the last observed action is updated.

After fixing the sequence of actions, we first check whether the entity gets created at any of the steps or is just moved or destroyed during the process. If the entity is not created, its initial location is equal to the first ‘from_loc’ in any subsequent actions. We then use the following criteria to fix the locations in a forward pass over the local decisions:

- If the action is “Move” but there is no final location, the final location is the first ‘from_loc’ from any of the subsequent actions before the next “Move” event.
- If the object is being “Moved”, then its final location should be changed. If the action does not indicate a new location or the information is missing, we replace the final location with ‘?’ to indicate an unknown location.
- If the action is “None”, the last location is kept unchanged for the new step.

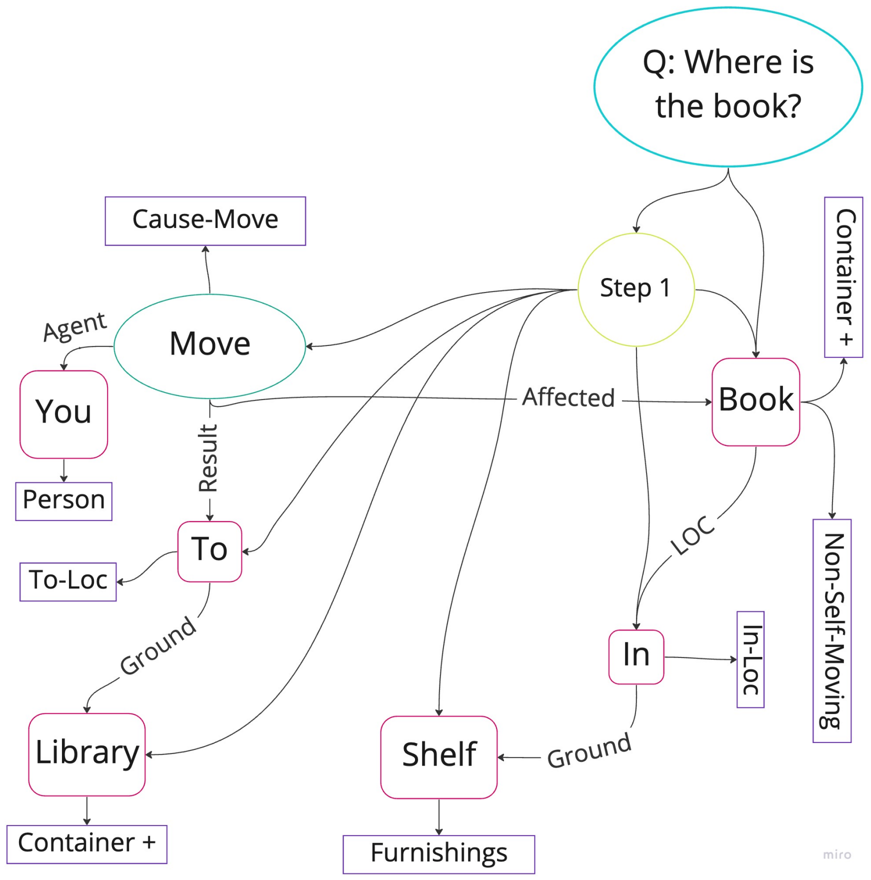


Figure 3.6 The QA graph for the query of "Where is the book" and the sentence "Move the book on the shelf to the library".

3.6.3 Integration with Neural Models

Here, we investigate whether explicitly incorporating semantic parsers with neural models can help better understand the procedural text. We choose two of the recently proposed and most commonly used backbone architectures for procedural reasoning tasks, namely NCET [61] and TSLM [49] (and its extension CGLI [99]). Similar to [68], we rely on a graph attention network (GAT) to integrate the information from the semantic parsers into the neural baselines.

Following [68], the nodes in this graph are either (1) predicates in the semantic frames, (2) mentions of entities of interest (Exact match or Co-reference), or (3) noun phrases in the sentence. An edge in the SRL graph exists between two nodes if they have a (predicate, argument) connection or they are both parts of the same verb semantic frame (argument to argument) [68]. It is relatively straightforward to build a semantic graph with the TRIPS parser because it outputs the parse as a graph.

An edge is created between any pairs of nodes (phrases) in the graph if any subsets of these two phrases are connected in the original parse. The edge types are preserved. Since not all the nodes in the original parse are present in the new simplified graph, we may lose some key connections. To fix this, if two nodes (phrases) are not connected in the new graph but have been connected in the original one, we find the shortest path between them in the original parse and connect them with a new edge with the type being the concatenation of all the edge types in the path. Lastly, nodes are connected across sentences based on either an exact match or co-reference resolution.

Both NCET and TSLM models are trained based on Cross Entropy to compute the loss for both actions and locations. The final loss of the model is calculated by $L_{total} = L_{action} \quad \lambda * L_{location}$, where λ is a balancing hyper-parameter.

3.6.3.1 Integration with NCET as Backbone

The NCET model uses a language model to encode the context of the procedure and compute representations for mentions of entities, verbs, and locations. These representations are used in two sub-modules to predict actions and locations. To integrate the semantic parsers with the NCET architecture, we use the output of the language model to initialize the semantic graph representations. Then multiple layers of graph attention network (based on TransformerConv [154]) are applied to encode the graph structure. We combine the updated graph representations with the initial mention representations. These combined representations are later used in subsequent prediction modules.

More formally, we start by using a language model to encode the context of the process $h' = LMS$, where S is the procedure, and h' is the embedding output from the language model. The representations are further encoded by a BiLSTM $h = BiLSTMh'$.

Graph Attention Network Since each node in the semantic graph corresponds to a subset of tokens in the original paragraph, we use the mean average of these tokens' representation to initialize the nodes' embedding denoted as v_i^0. If the graph contains edge types, the edges between every two nodes i and j are denoted by e_{ij} and are represented by the average token embedding through the same LM model used for encoding the story, $e_{ij} = MeanLMe_{ij}^{text}$. Lastly, we use C layers of TransoformerConv [154] to encode the graph structure. TransformerConv uses the following formula to update the representation of the nodes (v_i) in the graph.

$$\mathbf{v}_i^{l1} =$$

$$\mathbf{W}_1\mathbf{v}_i^{l} {}_{j\in\mathcal{N}i} \alpha_{i,j}\left(\mathbf{W}_2\mathbf{v}_j^{l}\ \ \mathbf{W}_6\mathbf{e}_{ij}\right),$$

where $\mathcal{N}i$ represents the neighbors of node i in the graph, l is the layer, and the coefficient $\alpha_{i,j}$ is computed using the following formula:

$$\alpha_{i,j} =$$

$$softmax\left(\frac{\left(\mathbf{W}_3\mathbf{v}_i^{l}\right)^{\top}\left(\mathbf{W}_4\mathbf{v}_j^{l}\ \ \mathbf{W}_6\mathbf{e}_{ij}\right)}{\sqrt{d}}\right)$$

Representing Mentions To integrate the semantic parses with the baseline model, we use both representations obtained from the language model and the graph encoder to represent entities, verbs, and locations in the process. Mention representations are denoted by $r_t^m = \left[Mh_t^m; Mh_{g_t}^{m}\right]$, where t is one step of the process, h_t^m is the average representation of tokens in the story corresponding to the mention m in step t, $h_{g_t}^{m}$ is the average embedding of nodes corresponding to mention m in step t, and the function M replaces the representations with zero if there is no mention of m in step t.

Location Prediction We first encode the pairwise representation of an entity e and location candidate lc at each step t, denoted by $\mathbf{x}_t^{e,lc} = \left[\mathbf{r}_t^e; \mathbf{r}_t^{lc}\right]$. Next, we use an LSTM to encode the step-wise flow of the pair representation to get $\bar{h}_t^{e,lc} = LSTM\left[\mathbf{x}_t^{e,lc}\right]$. Finally, the probability of each location candidate lc to be the location of entity e at step t is calculated by a $softmax$ over the potential candidates, $p^{e,lc_t} = SoftmaxW_{loc}^t\bar{h}_t^{e,lc}$, where W_{loc}^t is the learning parameters of a single multi-layer perceptron.

Action Prediction To predict the action for entity e at step t, we create a new representation for the entity based on its mention and the sentence verbs, denoted by $x_t^e = [r_t^e; Mean_{v\in npe_t} r_t^v]$, where npe_t is the set of verbs whose corresponding node in the graph has a path to any nodes representing

Tag	Description
O_D	Entity does not exist after getting destroyed
O_C	Entity does not exist before getting created
E	Entity exists and does not change
C	Entity is created
D	Entity is destroyed

Table 3.8 The list of output tags/actions in the Propara dataset is introduced to facilitate a linear correlation between actions.

	Local	Global Loc	Global Ent		Global Loc and Ent	Ambiguous
	Both	Both	Actions	Locations	Both	Actions
Train	885	367	438	340	114	593
Dev	116	44	66	3	9	76
Tests	105	61	98	71	18	110

Table 3.9 The number of decisions per evaluation category with the new decision-level metric. “Both” refers to both location and action decisions and is used since the number of those decisions is the same in most cases. The number of decisions in the ‘Global Ent’ case can differ in terms of actions and locations because this category also considers ‘destroy’ events with no corresponding locations.

entity e in step t. The final representations are then produced using a BiLSTM over the steps, $h_t^e = LSTMx_t^e$. Lastly, a neural CRF layer is used to consider the sequential structure of the actions by learning transition scores during the training of the model [61]. The set of possible actions is shown in Table 3.8.

3.6.3.2 Integration with TSLM as Backbone

We have discussed the TSLM model in more detail in Section 3.5. The TSLM [49] model reformulates the procedural reasoning task as a question-answering problem. The model simply asks the question, ‘Where is entity e?’ at each step of the process. To include the context of the whole process when asking the same question at different steps, TSLM further introduces a time-aware language model that can encode additional information about the time of events. Given the new encoding, each step of the process is mapped to either past, present, or future. TSLM uses the answer to the question at each step to form a sequence of decisions over the location of entity e. To integrate the semantic graph with this model, we first extend the graph by adding a question node. The graph is then initialized using the time-aware language model. The encoded representations of the graph, after applying multiple layers of GAT, are combined with the original token representations and used for extracting the answer to the question.

Initial Representation For each entity e and timestamp t, the string "where is e? s1 </s> s2 </s> ... s_m </s>" is fed into the time-aware language model. Accordingly, the tokens' representations for timestamp t are $h_{et}^{i} = LMS, t$.

Graph Attention Module Inspired by [194], we add new nodes to the semantic graph to represent the question and each step of the process. We connect the question node to any node in the graph representing the entity of interest e, and each step node to all the tokens in their corresponding sentence. All the node embeddings are initialized by the average embedding of their corresponding tokens in the procedure. We use C layers of graph attention network (TransformerConv), similar to Section 3.6.3.1, to encode the graph structure.

Location Prediction For predicting the locations of entities, that is, the answer to the question, we predict the answer among the set of location candidates. This is different from the common practice of predicting start/end tokens. We represent each location candidate by combining representations from both the graph and the time-aware language model, denoted by $r_t^{lc} = \left[h_{et}^{lc}, h_g^c l_t^{lc}\right]$, where $h_g^c l_t^{lc}$ is the representation of the lc from the last layer of the GAT. The answer is then selected by calculating a $softmax$ over the set of location candidates, $p_t^{lc} = Softmax W^{location} r_t^{lc}$.

Action Prediction Similar to CGLI [99] model, we explicitly predict the actions of entities alongside the locations. First, the model extracts each timestamp's "CLS" tokens and builds sequential pairs of CLS_t^e, CLS_{t1}^e. Then, it produces a change representation vector for each of these pairs, denoted by $r_t^e = FCLS_t^e; CLS_{t1}^e$. Lastly, the sequence of r_t^e logits is passed through the same neural CRF layer used by the NCET model, introduced in Section 3.6.3.1, to generate the final probability of actions.

3.6.4 Evaluation

We use three evaluation metrics to analyze the performance of the symbolic, sub-symbolic, and neural baselines. The first metric is sentence-level and proposed in [32]. The second metric is a document-level evaluation proposed by [167]. Both of these metrics evaluate higher-level procedural concepts that can be inferred from the model predictions rather than the raw decisions.

These metrics give more importance to the actions compared to the location decisions. Although they can successfully evaluate some aspects of the models, they fail to measure the research progress in addressing the challenges of the procedural reasoning task. We extend these evaluations with a new decision-level evaluation metric that considers almost all model decisions with a similar weight and evaluates the models based on the difficulty of the reasoning process.

Both sets of existing evaluation metrics of the Propara dataset do not directly evaluate the model's predictions but rather evaluate higher-level procedural concepts which can be inferred from the sets of decisions (i.e., an entity being input/output). Given their evaluation criteria, one model may surpass another in the number of correct decisions but still obtain a lower performance.

Therefore, we propose a new evaluation metric (**decision-level**) that directly evaluates the models' decisions. This evaluation metric is designed to consider the difficulty of the reasoning process and help better identify the core challenges of the task. We divide the set of decisions into five categories based on the presence of the entity e and the location l at each step t. We denote any mention of e by m^e, any mention of l by m_l, the action for entity e at step t by tag_t^e, and the text of the current step by S_t. The following specifies the five categories and how a decision falls under them.

Local Decision: A decision where (1) $m^e \in S_t$, (2) $m^l \in S_t$, and (3) $tag_t^e \in \{Move, Create\}$

Global Location Decision: A decision where (1) $m^e \in S_t$, (2) $m^l \notin S_t$, and (3) $tag_t^e \in \{Move, Create\}$

Global Entity Decision: A decision where (1) $m^e \notin S_t$, (2) $m^l \in S_t$ or $l = " - "$, and (3) $tag_t^e \in \{Move, Create, Destroy\}$

Global Entity and Location Decision: A decision where (1) $m^e \notin S_t$, (2) $m^l \notin S_t$, and (3) $tag_t^e \in \{Move, Create\}$

Ambiguous Local Action: A decision where (1) $m^e \in S_t$ and (2) S_t contains multiple action verbs.

Table 3.9 shows the detailed statistics of the number of decisions falling under each of these five categories for the Propara dataset. Evaluating the performance of models given the new decision-level metric will clarify the lower-level challenges in the reasoning over states and locations of entities simultaneously. Getting accurate predictions in any of these categories of decisions requires the models to have different reasoning capabilities.

The local decisions mostly require a sentence-level understanding of the action and its consequences. The global location decisions require reasoning over the current step and the ability to connect the local information to the global context. The predictions for the category of the global entity mostly require reasoning over complex co-references (we have already considered simple co-references such as pronouns as mentions of the entity) or the ability to recover missing pronouns in a sentence such as "Gradually mud piles over (them)". The global entity and location decisions are the most challenging cases, which require reasoning over local and global contexts, complex co-reference resolution, and handling of missing pronouns. The ambiguous decisions mainly require local disambiguation of (entity, role, predicate) connections when multiple predicates are present in the sentence. Moreover, common sense is required for a subset of all the decision categories.

3.6.5 Experiments

Implementation details We use the PyTorch geometric [6] library to implement all the graph attention models and Huggingface library [177] for implementing the language models. For the NCET model and its extensions based on semantic parsers, the best model is selected by a search over the $\lambda \in \{0.3, 0.4\}$, the learning rate in $\{3e-5, 3.5e-5, 5e-5\}$. The number of graph attention layers are set to 2 and the batch size is set to 8 process. All models use Bert-base as the selected language model for encoding the context. We further use RAdam [93] to optimize the model parameters of both the language models, the LSTM, and the classifiers. For the CGLI method, we use the exact hyper-parameters as specified in [99]. We further use 15 layers of graph attention network with the input from the fifth layer of the time-aware language models. The gradients from the graph attention network (GAT) would not back-propagate to the original language model and only affect the parameters in the GAT model.

[6]https://pytorch-geometric.readthedocs.ios

#Row	Models	Sentence-level evaluation					Document-level evaluation		
		Cat1	Cat2	Cat3	Macro-avg	Micro-avg	Precision	Recall	F1
1	ProLocal	62.7	30.5	10.4	34.5	34.0	**77.4**	22.9	35.3
2	ProGlobal	63	36.4	35.9	45.1	45.4	46.7	52.9	49.4
3	KG-MRC	62.9	40	38.2	47	46.6	64.5	50.7	56.8
4	PROPOLIS(ours)	69.9	37.71	5.6	37.74	36.67	70.9	50.0	58.7
5	NCET (re-implemented)	75.54	45.46	41.6	54.2	54.38	68.4	63.6	66
6	REAL(re-implemented)*	78.9	48.31	41.62	56.29	56.35	67.3	64.9	66.1
7	NCET + SRL(ours)	77.1	46.35	42	55.16	55.32	67.8	65.2	66.5
8	NCET + TRIPS(ours)	77.1	48.12	43.36	56.19	56.32	72.5	65.4	68.8
9	NCET + TRIPS(Edge)(ours)	75.68	47.6	45.71	56.33	56.37	69.9	65.5	67.6
10	NCET + PROPOLIS(ours)	78.54	48.69	44.26	57.16	57.31	74.6	65.8	69.9
11	DynaPro	72.4	49.3	44.5	55.4	55.5	75.2	58	65.5
12	KOALA	78.5	53.3	41.3	57.7	57.5	77.7	64.4	70.4
13	TSLM	78.81	56.8	40.9	58.83	58.37	68.4	68.9	68.6
14	CGLI	80.3	**60.5**	48.3	**63.0**	**62.7**	74.9	**70**	**72.4**
15	CGLI + TRIPS (ours)	**80.62**	58.94	**49.08**	62.88	62.68	74.5	68.5	71.4

Table 3.10 The table of results based on sentence-level and document-level evaluation of the Propara Dataset. * Since the code for the REAL model is not available, we have re-implemented the architecture based on the guidelines of the paper and the communications. The graph is first abstracted using the PROPOLIS graph abstraction phase and then used instead of the Trips parse as input to the model.

Model	Local			Global Loc			Global Ent			Global Loc and Ent			Amb
	A	L	Both	A	L	Both	A	L	Both	A	L	Both	A
KOALA	74.3	65.7	59.0	**86.9**	24.6	22.9	1.0	7.0	0.0	5.6	11.1	0	73.63
PROPOLIS	55.2	19.0	19.0	63.9	1.6	1.6	0.0	9.9	0.0	0.0	0.0	0.0	52.7
NCET	69.5	62.8	60.0	70.5	36.1	29.5	3.1	5.6	0.0	0.0	0.0	0.0	57.2
NCET + SRL	68.6	65.7	61.9	77.0	36.1	31.1	10.2	5.6	0.0	5.5	5.5	0.0	62.7
NCET + TRIPS	71.4	67.6	**63.8**	75.4	42.6	36.1	10.2	9.9	2.8	5.5	11.1	0.0	63.6
NCET + PROPOLIS	71.4	64.8	61.9	83.6	36.1	34.4	3.1	7.0	0.0	5.5	5.5	0.0	70.9
CGLI	65.7	62.9	54.3	75.4	59.0	50.8	**19.4**	19.7	11.3	22.2	**27.8**	11.1	70.0
CGLI + TRIPS	**75.2**	**70.5**	61.9	80.3	**60.6**	**52.2**	17.3	**22.5**	**12.7**	**27.8**	**27.8**	**16.7**	**74.5**

Table 3.11 The results of the models on the new extended evaluation metric (decision-level) in terms of accuracy (%). 'A' means the action is correct, 'L' means the location is correct, and 'Both' means both the action and location are correct. Local ambiguous cases.

3.6.5.1 Results

Here, we summarize the performance of strong baselines compared with the symbolic (PROPOLIS) and integrated models. Table 3.10 shows the performance of models in the two conventional metrics of the Propara dataset, and Table 3.11 shows the performance of models based on the decision-level metric. We summarize our findings in a set of question-answer pairs.

Q1. Can semantic parsing alone solve the problem reasonably? Based on Table 3.10, the PROPOLIS model outperforms many of the neural baselines (document-level F1-score of row#4 compared to rows #1 to #3), showing that deep semantic parsing can provide a general solution for the procedural reasoning task to some extent without the need for training data. This model performs relatively well on action-based decisions (cat1) but fails to extract the proper location decisions (cat3). This is because many locations are inferred based on common sense rather than the verb semantic frames. Notably, the set of rules written on top of PROPOLIS is local and simple and can be further expanded to improve performance. Table 3.11 further indicates that the predictions of the PROPOLIS model on the actions are much closer to the SOTA models than its predictions for the entities' location. The good performance of PROPOLIS on the action decisions for the "Global Location" category can further show that the local context can mostly indicate the action even if retrieving the result of the action (location) requires more reasoning steps. Lastly, since PROPOLIS is a model built over local semantic frames, it dramatically fails to make accurate decisions when the entity does not appear in the sentence (Global Ent).

Q2. Can the integration of semantic parsing improve the neural models? We evaluate this based on the two strong baselines, NCET and TSLM. When semantic parsers are integrated into NCET, all three evaluation metrics improve (compare rows #7 to #10 with row #5). This improvement is even better if the source of the graph is the abstracted parse from the PROPOLIS method (row #10). Semantic parsers improve NCET's performance in all categories of decisions, particularly in local ambiguous sentences and decisions requiring reasoning over global locations. Notably, the integration of PROPOLIS with the NCET model significantly boosts the ability to disambiguate local information in sentences with multiple action verbs.

The integration of the semantic graph slightly hurts the performance of the CGLI baseline when using conventional metrics (1%). However, it outperforms this baseline on "cat3" (0.78%), which is the only evaluation that directly considers location predictions. Notably, the original CGLI model (baseline) uses the pre-trained classifiers from SQUAD [132] to predict the start/end tokens from the paragraph as the locations (answer to the question). However, since the integrated method extracts candidates from the graph in the form of spans, it cannot reuse the same pre-trained classifier parameters. This may contribute to the drop in performance since CGLI performs 2% lower on the document-level F1 score when SQUAD pre-training is removed [99]. Despite the drop in performance based on the conventional metrics, the integrated QA-model (CGLI + TRIPS) outperforms the baseline in almost all the criteria in the new evaluation (Table 3.11), especially on decisions that only require local reasoning or local disambiguation. This is due to the global nature of the TSLM (or CGLI) backbone, which predicts the locations based on the whole story and ignores many of the local signals, whereas the graph can help directly extract the local relations.

Q3. How can the decision-level metrics help understand models' weaknesses and strengths? Based on the results in Table 3.11, the NCET model is better at reasoning over the local context than the global context. It also clarifies that although the TSLM (or CGLI) model can properly reason over multiple steps, it is not as competitive as the NCET model in the local cases. However, integrating semantic parsers could improve the models to close the gap on both local and global aspects and has a complimentary influence on the initial performance of the baselines. As a general conclusion based on our new evaluation metrics, we can argue that the most challenging decisions are the ones that require reasoning over missing mentions of entities in the local context. Addressing this challenge may require external reasoning over common-sense, performing the complex co-reference resolution, or handling missing pronouns.

3.6.6 Discussion

Here, we discuss some of the potential concerns that may arise with the usage of symbolic systems such as TRIPS and the new evaluation criteria.

Coverage and rule crafting of PROPOLIS. Our implementation of the symbolic method and the integrated models rely on the knowledge extracted from very fine-grained semantics covered in TRIPS. Consequently, a small mapping effort was needed to create such a system. The mapping between actions in Propara and verbs is straightforward since verbs are automatically mapped onto ontological classes that provide the type of actions based on the parse. Hence, defining the mapping rules for the most general relevant ontology types of verbs is sufficient because all the descendent types will follow the same mapping (See Table 3.7). Additionally, the effort needed for the pre-processing and designing of the mapping rules is similar to the hyperparameter tuning of neural models. Since mapping is based on common sense rather than trial-and-errors in hyperparameter tuning, finding an optimal solution may even take less effort.

Out-Of-Vocabulary words in parses. TRIPS automatically maps words to ontology classes using

WordNet [110]. This gives us considerable vocabulary coverage and reduces OOV risk. TRIPS can identify the role of the unseen words (not available in WordNet) based on the sentence syntax and will not produce errors when encountering unseen words. In the same way, PROPOLIS and integrated models will not be affected.

Effectiveness of the new evaluation metric. The previously proposed high-level evaluations are strict and do not accurately reflect the quality and quantity of the lower-level model decisions. Thus they do not adequately reveal the models' abilities. For example, when compared at high-level metrics, two models may have the same performance value of 20%, while their decision accuracy may be 60% and 10%. This issue is reflected during training epochs too when the models' performance remains the same despite the decisions on the train set continuing to improve. Therefore, it seems more appropriate to evaluate the models based on the same objective criteria used for training them (decision-level). However, the previously used metrics can be secondary evaluations to measure how well the model captures higher-level procedural concepts.

CHAPTER 4

MULTI-MODAL PROCEDURAL ABSTRACTION

Procedural abstraction is challenging due to the difficulty of identifying evolving events and entities. In contrast to the summarization task, where the summary represents the whole context and usually contains sentences from the original content, the abstraction is rather a very short description of the process trying to capture its essential actions and objects in an abstract format.

To investigate this task, we study this in a simplified version, where the abstract is a 4-phrase action that should make sense in a specific chronological order with respect to the original text. Here, we use the RecipeQA benchmark and the Textual Cloze task. Figure 4.1 shows an example of the task. We discussed the task and our proposal for integrating latent alignment between the abstract and the recipe steps in more detail in Section 4.1.

4.1 Task Definition and Challenges

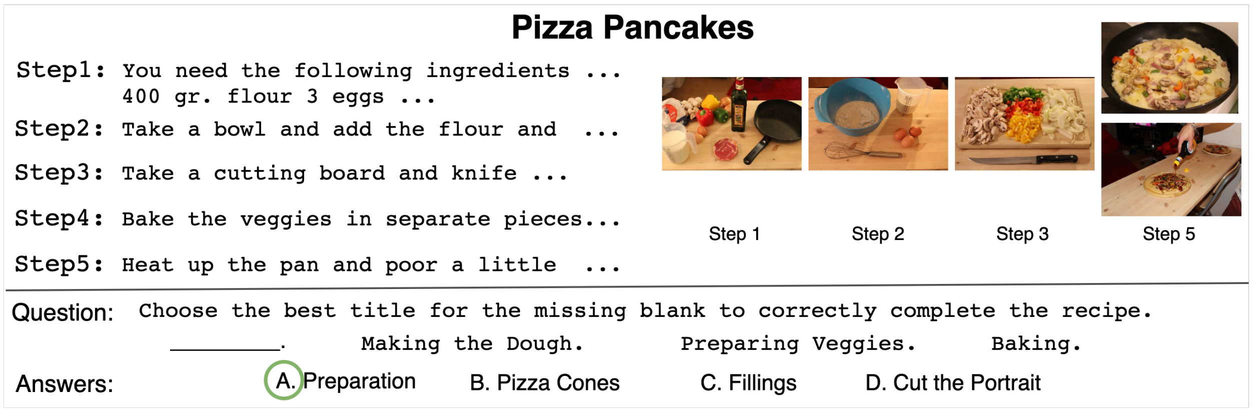
We tackle the task of procedural reasoning in a multimodal setting for understanding cooking recipes. The RecipeQA dataset [184] contains recipes from internet users. Thus, understanding the text is challenging due to the different language usage and informal nature of user-generated texts. The recipes are along with images provided by users, which are taken in an unconstrained environment. This exposes a level of difficulty similar to real-world problems.

The tasks proposed with the dataset include textual cloze, visual cloze, visual ordering, and visual coherence. Here, we focus on textual cloze. An example of this task is shown in Figure 4.1.

4.1.1 Task Definition

The task involves processing a set of multimodal instructions denoted by I. The input also includes three textual items from the question, denoted by q_i, q_j, q_k, and a placeholder, denoted by p_l, that is to be replaced by the correct answer. The index l can be between any of the two indexes from the question items, i.e., $i < l < j$ or $j < l < k$. The task requires selecting the correct answer, denoted by a, from a set of four options, denoted by $A = \{a_1, a_2, a_3, a_4\}$, based on the given set of instructions.

Let S denote the sequence of the three question items and the correct answer that correctly



Pizza Pancakes

```
Step1: You need the following ingredients ...
       400 gr. flour 3 eggs ...
Step2: Take a bowl and add the flour and  ...
Step3: Take a cutting board and knife ...
Step4: Bake the veggies in separate pieces...
Step5: Heat up the pan and poor a little  ...
```

Question: `Choose the best title for the missing blank to correctly complete the recipe.`

`_________.    Making the Dough.    Preparing Veggies.    Baking.`

Answers: [✓ A. Preparation]   B. Pizza Cones   C. Fillings   D. Cut the Portrait

Figure 4.1 A sample of textual cloze task in the RecipeQA, which is a multi-choice question with one valid answer

describes the steps of the recipe. Then, the task involves finding the correct answer a such that the sequence S is a valid description of the recipe. Mathematically, the task can be formalized as follows:

Given a set of multimodal instructions $I = \{i_1, i_2, ..., i_n\}$, where each i_j is a combination of visual and textual information, and three textual question items q_i, q_j and q_k with a placeholder p_l, the task is to select the correct answer a from a set of options $A = \{a_1, a_2, a_3, a_4\}$, such that the sequence $S = \{q_i, q_j, q_k, a\}$ (sorted based on the $_x$ indexes) correctly describes the steps of the recipe, i.e., S is a valid recipe sequence. Mathematically,

$$a = \arg\max_{a_i \in A} PS|I, a_i$$

where $PS|I, a_i$ denotes the probability of the sequence S given the set of instructions I and the option a_i.

4.2 Latent Alignment of Procedural Concepts

To design a semantically augmented model, we rely on the intuition that given question items, and each answer describes exactly one step of the recipe. Hence, we design a model to make explicit alignments between the candidate answers and each step and use those alignment results, given the question information. This alignment space is latent due to not having any direct supervision based on provided annotations.

Using multimodal information and representations by making a joint space for comparison has been broadly investigated in the recent research [66, 179, 86, 161, 189, 52, 166, 116]. Our work differs from those as we do not have direct supervision on multimodal alignments. Moreover, the task we are solving uses the sequential nature of visual and textual modality as a weak source of supervision to build a neural model to compare the textual representation of context and the answers for a given question representation.

Our model exploits the latent alignment space and the positional encoding of questions and answers while applying a novel approach for constraining the output space of the latent alignment. Moreover, we exploit cross-modality representations based on cross-attention to investigate the benefits of information flow between images and instructions. We compare our results to the provided baselines in [184] and achieve state-of-the-art by improving over 19%.

4.2.1 Forming Latent Alignments

To incorporate the order of the sequence in question items and the placeholder, we utilize a one-hot encoding vector of positions to be concatenated with the candidate answers and question items' representations.

We give the instructions to a sentence splitter using Stanford Core NLP library [102]. The output is then tokenized by Flair data structure [3] and embedded with BERT [43]. The words' embeddings are passed to an LSTM layer and the last layer is used as the instruction representation. We propose two different approaches to include image representations. These proposals are described in Section 4.3.2.1. An overview of our approach is shown in Figure 4.2.

Question representation is the last layer of an LSTM on question items. The representation of each question item is the concatenated vector of a one-hot position encoding and word embedding obtained from BERT. The candidate answers' representations are computed using the same approach. We concatenate the question representation to each instruction. Then, the similarity of each candidate answer and instruction is computed using the cosine similarity and forms a similarity matrix. We use S to denote the similarity matrix. The rows of this matrix are candidate answers, and the columns represent the recipe steps. The value of S_{ij} indicates the similarity score

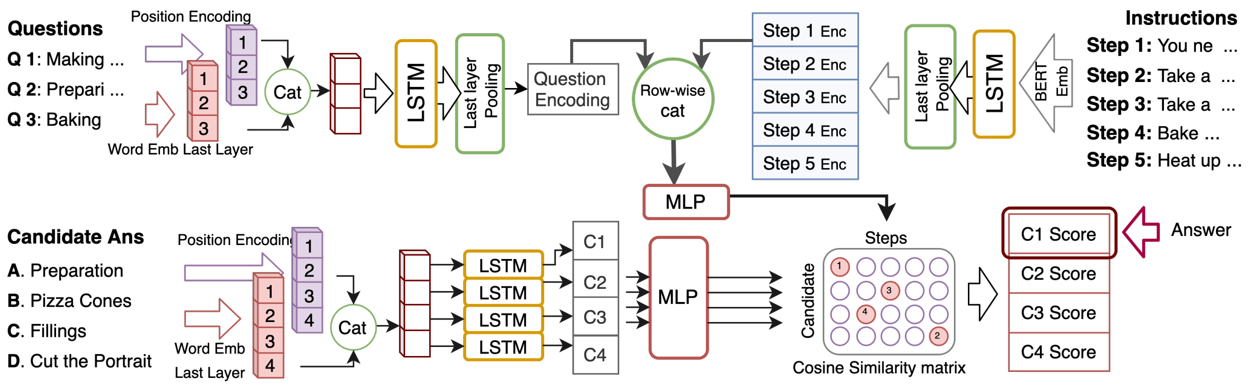


Figure 4.2 An overview of the proposed model. The Bert Embeddings are used in a frozen manner while the LSTM would further fine-tune the representation based on the target task.

of candidate i and step j.

For training the model, we define two different objectives directly applied to the similarity matrix. The textual cloze task does not have the direct supervision required for the alignment between candidates and steps, and our objective is designed to use the answer to the question to train this latent space of alignments. For imposing the constraint of the alignment to be disjoint between steps and candidates, one way is to simply compute the maximum of each row in the similarity matrix and use that as the aligned step for each candidate answer; However, we introduce constrained max-pooling that is a more sophisticated approach as shown in Figure 4.3. We compare these two alternatives in the experimental results. We apply an iterative process to select the most related pair of instructions (a column) and answer candidate (a row) while removing the related column and row each time until all candidate answers find their aligned instruction. We denote the final selected maximum scores by $m = S_{1i_1}, S_{2i_2}, S_{3i_3}, S_{4i_4}$, where $i_c \in 1, number\_of\_steps$ is the index of the step with maximum alignment score with candidate $c$and for all pairs of candidates c and d, $c \neq d \implies i_c \neq i_d$.

Respectively, we define two following objectives. The first objective maximizes the distance between the maximum score of the correct answer and the maximum score of another random wrong answer candidate. Furthermore, fixing the instruction with the maximum alignment with the correct answer decreases the score of the other candidates' alignments with that instruction. The second objective increases the maximum similarity score of the answer to approach one while

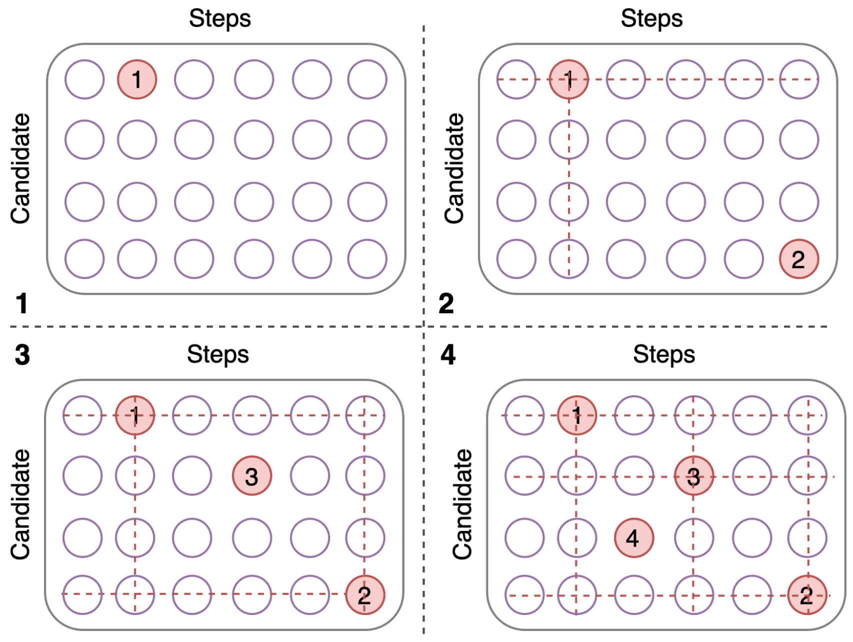


Figure 4.3 The matrix operation for constrained max-pooling. After selecting each maximum value in the available matrix, the row and the column corresponding to the value are removed from the matrix.

decreasing the other maximum scores to be lower than 0.1.

$$\begin{aligned} Loss &= \max 0, S_{ri_r} - S_{ai_a} \;\; 0.1 \\ &\overset{4}{\underset{c \neq a}{}} \max 0, S_{ci_a} - S_{ai_a} \;\; 0.1 \end{aligned} \tag{4.1}$$

$$Loss = 1 - S_{ai_a} \;\; \overset{4}{\underset{c \neq a}{}} \max 0, S_{ci_c} - 0.1 \tag{4.2}$$

Where $a \in \{1, 2, 3, 4\}$ is the correct answer number and r is a random index from $\{1, 2, , 3, 4\} - \{a\}$. The main difference between objective 4.1 and objective 4.2 is the regularization term on the selected instruction column in the alignment matrix.

4.3 Experiments and Discussion

4.3.1 Baselines

Hasty Student [172] is a simple approach considering only the similarity between elements in question and candidate answers. This baseline fails to get good results due to the intrinsic of the task.

Impatient Reader [65] computes attention from answers to the recipe for each candidate, and

Models	Accuracy	p@2
Human	73.6	-
Hasty Student	26.89	-
Impatient Reader	28.03	-
Impatient Reader (multimodal)	29.07	-
Model-Obj 4.1	46.35	**78.7**
Model-Obj 4.2	43.36	-
Model-Obj 4.1 (multimodal)	45.41	
Model-Obj 4.1 (multimodal) + LXMERT	**47.5**	77.5
Model-Obj 4.1 (multimodal) + LXMERT - ConstrainedMaxPooling	46.9	76.3

Table 4.1 Evaluation on the test set of the RecipeQA dataset. The Multi-modal setting simply applies a late fusion, while LXMERT applies an early fusion for modalities.

despite being a complicated approach, yet it fails to get good results on the task. Moreover, the multimodal Impatient reader approach uses both instructions and corresponding images.

4.3.2 Results

The RecipeQA textual cloze task contains 7837 training, 961 validation, and 963 test examples. A learning rate of $4-e1$ is used for the first half and then $8-e2$ for the second half of training iterations. We use the momentum of 0.9 for all variations of our model. We train for 30 iterations with a batch size of 1 and optimize the weights using an SGD optimizer. For word embedding, the pre-trained BERT embedding in the Flair framework is used. For the image representations, ResNet50 [63] pre-trained on Imagenet [142] using PyTorch library [125] is applied.

Table 4.1 presents the experimental results. We call the model variations which use the loss objective in Equation (4.1) as Model-obj 4.1 and the ones that use the loss in Equation (4.2) as Model-obj 4.1. Using the objective in Formula (4.1) yields better results in all experiments. This indicates the benefit of using the column-wise disjoint constraint on the similarity matrix. Also, using multimodal information yields a 1.12% improvement. We elaborate further on the comparison between multimodal and unimodal results in Section 4.3.3.

We provide our Pytorch implementation publicly available on Github [1].

[1]https://github.com/HLR/LatentAlignmentProcedural

4.3.2.1 Multimodal Results

In order to investigate the usefulness of the images in solving the textual cloze task, we propose two different models that incorporate the image representation in addition to the textual information of recipe steps. The first variation receives ResNet50 representations of the images and, after applying an LSTM layer, pulls the last layer as image representation. Finally, it concatenates the image representation to the question and instruction representation in the main architecture before applying the MLP and computing the cosine similarities.

As shown in figure 4.4, the second variation uses a more complex architecture introduced in LXMERT [166]. We modify the architecture of LXMERT and apply it to the word embedding and image representations to flow the information from one to another. The updated word embedding and image representations are passed to an LSTM, and its last layer is used to represent the visual and textual information of a step. In the end, these representations are concatenated with each other and the question representation to build the instruction vector representation. We report the results of these model variations in Table 4.1. Using the cross-modality representations based on LXMERT provided an extensive way to flow the information from text and image to each other and yield the best results.

4.3.3 Discussion and Analysis

We did a qualitative analysis using some examples and their results to understand the behavior of the proposed model better. Our model can almost detect all matched candidates with the instructions (in case multiple matches exist) but fails to choose the one that completes the sequence of the question items. This indicates the shortage of procedural hints inside our architecture while the latent alignment is proven to be practical. Analyzing the results, we found interesting cases where multimodal or unimodal architectures could yield more accurate predictions.

Multimodal - , Unimodal +:

- Images contain misleading information (see example in Figure 4.5).

- Image quality is low.

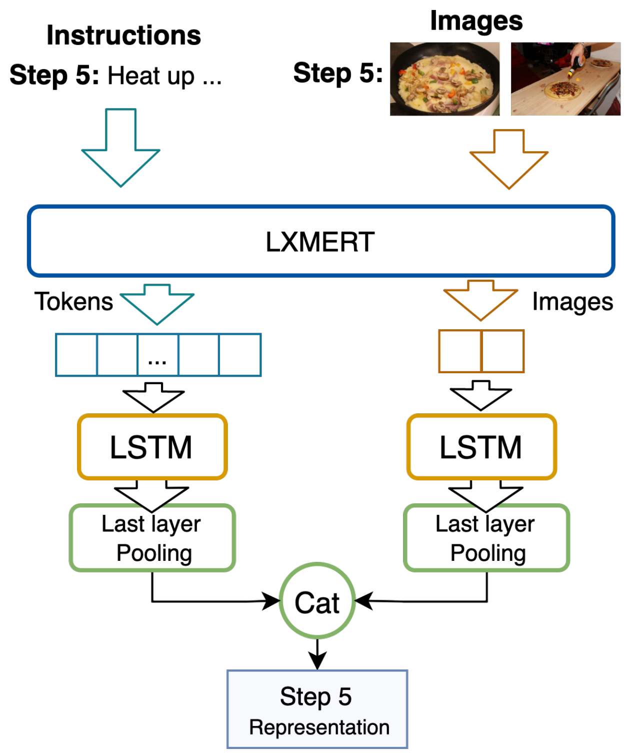
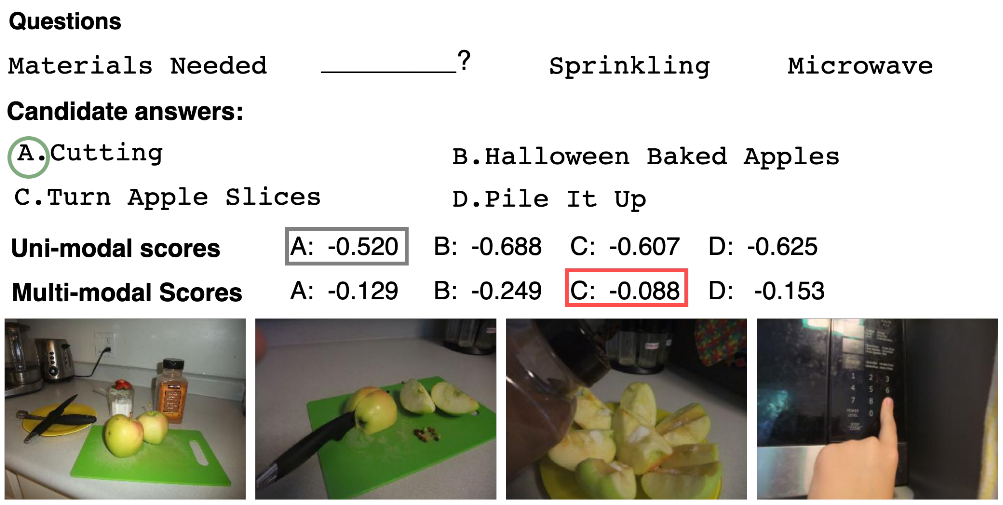


Figure 4.4 Showing the use of LXMERT for integrating multimodal information on steps. The model parameters of LXMERT are frozen.



Questions

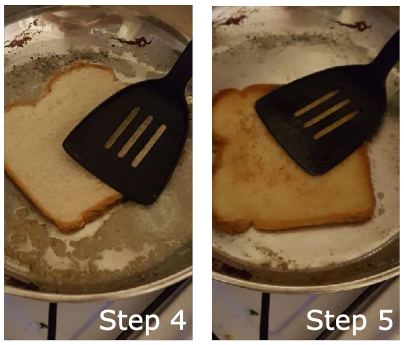
`Materials Needed` ________? `Sprinkling` `Microwave`

Candidate answers:

[✓ A.]Cutting	B.Halloween Baked Apples
C.Turn Apple Slices	D.Pile It Up

Uni-modal scores	A: -0.520	B: -0.688	C: -0.607	D: -0.625
Multi-modal Scores	A: -0.129	B: -0.249	C: -0.088	D: -0.153

Figure 4.5 Showcasing the setting where utilizing the vision modality misleads the model to choose apple slices (object) rather than the cutting option (action).



Step 4: Push the bread down with a spatula to get it toasty.
Step 5: Once it has started to sizzle flip it. The new top side should be a golden brown.

Questions

1.Add Butter. 2.Press the Bread.
3.________? 4.Cut the Sandwich.

Candidate answers: A.Grilled Cheese Experiment B.Grilled Cheese C.Flip the Sandwich [✓ D.]Flip the Bread!

Uni-modal scores	A: -0.743	B: -0.760	$\boxed{\text{C: -0.522}}$	D: -0.591
Multi-modal Scores	A: -0.298	B: -0.287	C: -0.122	$\boxed{\text{D: -0.110}}$

Figure 4.6 Showcasing the setting where the vision modality helps the model to understand "it" refers to bread rather than a sandwich.

- Images are not showing the steps correctly.
- Text contains direct mentions of candidate answers.

Multimodal + , Unimodal -:

- The sequence of the images provide detailed steps and good quality.
- The entities in candidate answers are shown in the pictures but not in the text.
- The recipe instructions are very short, and the images provide more information.

In some cases, the multimodal information can fix the errors resulting from ignoring the order of events in the proposed architecture. Our intuition is that, although the textual model does not contain information from previous steps, the images carry useful information on what has already been done. Figure 4.6 shows an example of this, where co-reference resolution is required to answer the question correctly.

Experimenting with ResNet101 for the multimodal architecture resulted in lower performance. We confirmed this experiment by re-implementing the Hasty Student approach with ResNet101 on the visual coherence task (which has 68% accuracy with ResNet50) and obtained 35% lower performance than the performance of the model utilizing ResNet50. This can be due to the lack of quality of images, which results in extra noise when using a more complex network. Thus, ResNet50 achieves better accuracy by producing more abstract representations of the images.

CHAPTER 5

NEURAL LEARNING WITH LOGICAL CONSTRAINTS

Recent advancements in machine learning are proven very effective in solving real-world problems in various areas, such as vision and language. However, there are still remaining challenges. First, machine learning models mostly fail to perform well on complex tasks where reasoning is crucial [149] while human performance does not drop as much when more steps of reasoning are required. Second, deep neural networks (DNNs) are known to be data-hungry, making them struggle on tasks where the annotated data is scarce [87, 198]. Third, models often provide results that are inconsistent [88, 57] even when they perform well on the task. Prior research has shown that even large pre-trained language models performing well on a specific task may suffer from inconsistent decisions and indicate unreliability when attacked under adversarial examples and specialized test sets that evaluate their logical consistency [57, 113]. This is especially a major concern when interpretability is required [106], or there are security concerns over applications relying on the decisions of DNNs [19].

To address these challenges, one direction that the prior research has investigated is neuro-symbolic approaches as a way to exploit both symbolic reasoning and sub-symbolic learning. Here, we focus on a subset of these approaches for the integration of external knowledge in deep learning. Knowledge can be represented through various formalisms such as logic rules [67, 118], Knowledge graphs [195], Context-free grammars [41], Algebraic equations [159], or probabilistic relations [28]. A more detailed investigation of available sources of knowledge and techniques to integrate them with DNNs is surveyed in [174, 36]. Although integrating knowledge into DNNs is done in many different forms, we focus on explicit knowledge about the latent and/or output variables. More specifically, we consider the type of knowledge that can be represented as declarative constraints imposed (in a soft or hard way) on the models' predictions, during training or at inference time. The term knowledge integration is used in the scope of this assumption in the remainder of this paper.

In this chapter, we discuss our efforts to progress research toward integrating domain knowledge into deep neural models. Here, the domain knowledge is expressed as a set of constraints over the latent/output variables of the neural network.

To facilitate research in this direction, we first propose a declarative framework, DomiKnowS. This framework aims to provide a declarative interface to define the knowledge of the task in terms of a graph of concepts involved in the task, a first-order logic interface to define constraints over the concepts, an interface to design neural components, and an interface to design learning or execution objectives. DomiKnowS provides seamless integration of knowledge in terms of constraints with the deep neural models by implementing generalizable conversions between the first-order logic constraints to various formalisms used in integration methods. We will discuss this framework in more detail in Section 5.1. Implementing and designing DomiKnowS is a great team effort, and my contributions have been on developing various techniques to enhance the generalizability of the framework, implementing and developing constraint integration methods, debugging the framework capabilities, and showcasing its abilities on multiple challenging tasks.

Building on top of DomiKnowS, we provide the first standard collection of benchmarks (GLUECons) to evaluate the integration methods. This benchmark provides extensive evaluation criteria and task categorizations that help identify the advantages and disadvantages of various constraint integration methods. GLUECons provides an overview of the existing approaches for constraint integration during training or inference on nine distinctive tasks categorized into five categories based on the source of the constraints. We discuss this benchmark in Section 5.2. GLUECons is a big team effort that contains over 100 experiments on various timelines. My contribution to the project involves proposing the need for the benchmark, leading the team effort, proposing new evaluation criteria, summarizing the results of the huge experiment pull into discussion points, and implementing the experiments needed for multiple of the selected tasks in the benchmark.

5.1 Declarative Constraint Integration Framework

Current deep learning architectures are known to be data-hungry with issues mainly in generalizability and explainability [120]. While these issues are hot research topics, one approach to address them is to inject external knowledge directly into the models when possible. While learning from examples revolutionized the way that intelligent systems are designed to gain knowledge, many tasks lack adequate data resources. Generating examples to capture knowledge is an expensive and lengthy process and especially not efficient when such knowledge is available explicitly. Therefore, one main motivation of our DomiKnowS is to facilitate the integration of domain knowledge in deep learning architectures, particularly when this knowledge is represented symbolically. We highlight the components of this framework that help to combine learning from data and exploiting knowledge in learning, including 1) Learning problem specification, 2) Knowledge representation 3) Algorithms for integration of knowledge and learning. Currently, DomiKnowS implementation relies on PyTorch and off-the-shelf optimization solvers such as Gurobi. However, it can be extended by developing hooks for other solvers and deep learning libraries since the interface is generic and independent from the underlying computational modules.

In general, the integration of domain knowledge can be done 1) using pretrained models and transferring knowledge [42, 113], 2) designing architectures that integrate knowledge expressed in knowledge bases (KB) and knowledge graphs (KG) in a way that the KB/KG context influences the learned representations [185, 163], or 3) using the knowledge explicitly and logically as a set of constraints or preferences over the inputs or outputs [89, 118, 115, 159]. Our current library aims at facilitating the third approach. While applying the constraints on input is technically trivial and could be done in a data pre-processing step, applying constraints over outputs and considering those structural constraints during training is a research challenge [118, 89, 60]. This requires encoding the knowledge at the algorithmic level. However, given that the constraints can be expressed logically and symbolically, having a language to express such knowledge in a principled way is lacking in the current machine-learning libraries.

Using our developed DomiKnowS library, the domain knowledge will be provided symbolically and by the user utilizing a logical language that we have defined. This knowledge is used in various ways: a) As soft constraints by considering the violations as a part of the loss function, this is done using a prim-dual formulation [118] and can be expanded to probabilistic and sampling-based approaches [183] or by mapping the constraint to differentiable operations [90] b) mapping the constraints to an integer linear program and performing inference-based training by masking the loss [60]. Independent from the training paradigm, the constraint can always be used as hard constraints during inference or not used at all. An interactive online demo of the framework is available at Google Colab[1], and the framework is accessible on GitHub[2].

5.1.1 Related Research

Integrating domain knowledge in learning relates to tools that try to express the prior or posterior information about variables beyond what is in the data. This relates to probabilistic programming languages such as [126], Venture [103], Stan [22], and InferNet [112]. The logical expression of domain knowledge is used in probabilistic logical programming languages such as ProbLog [37], PRISM [147], the recent version of Problog, that is, Deep Problog [100], Statistical Relational Learning tools, such as Markov logic networks [45], Probabilistic soft logic [18], Bayesian Logic (BLOG) [109], and slightly related to learning over graph structures [196]. Considering the structure of the output without its explicit declaration is considered in structured output prediction tools [141]. This library is mostly related to the previous efforts for learning-based programming and the integration of logical constraints in learning with classical machine learning approaches [136, 73, 74]. Our framework makes this connection to deep neural network libraries and arbitrarily designed architectures. The unique feature of our library is that the graph structure is defined symbolically based on the concepts in the domain. Despite Torch-struct [141], our library is independent of the underlying algorithms, and arbitrary structures can be expressed and used based on various underlying algorithms. In contrast to DeepProbLog, we are not limited to probabilistic inference, and any solver can be used for inference depending on the training paradigm that is used for exploiting the logical constraints. Probabilistic soft logic is another framework that considers logical constraints in learning by mapping the constraint declarations to a Hing loss Markov random field [11]. DRaiL is another declarative framework that uses logical constraints on top of deep learning and converts them to an integer linear program at inference time [191]. None of the above-mentioned frameworks accommodate working with raw sensory data nor help in putting that

[1]https://hlr.github.io/domiknows-nlp/
[2]https://github.com/HLR/DomiKnowS

in an operational structure that can form the domain predicates and be used by learning modules while our framework tries to address that challenge. We support training paradigms that make use of the inference as a black box, and in those cases, any constraint optimization, logical inference engine, or probabilistic inference tool can be integrated and used based on our abstraction and the provided modularity.

5.1.2 Declarative Learning-based Programming

We use the Entity-Mention-Relation (EMR) extraction task to describe the framework.

Given an input text such as "*Washington is employed by Associated Press.*", the task is to extract the entities and classify their types (e.g., people, organizations, and locations) as well as relations between them (e.g., works for, lives in). For example, for the above sentence *[Washington]* is a $person$ *[Associated Press]* is an $organization$ and the relationship between these two entities is $work\text{-}for$. We choose this task as it includes the prediction of multiple outputs at the sentence level, while there are global constraints over the outputs.[3]

In DomiKnowS, first, using our python-based specification language, the user describes the problem and its logical constraints declarativly and independent from the solutions. Second, it defines the necessary computational units (here, PyTorch-based architectures) and connect the solution to the problem specification. Third, a program instance is created to execute the model using a background knowledge integration method with respect to the problem description.

5.1.2.1 Problem Specification

To model a problem in DomiKnowS, the user should specify the problem domain as a conceptual graph $\mathcal{G}V, E$. The nodes in V represent concepts and the edges in E are relationships. Each node can take a set of properties $P = P_1, P_2, ..., P_n$. Later, the logical constraints are expressed using the concepts in the graph. In EMR task, the graph contains some initial NLP concepts such as *sentence*, *phrase*, *pair* and additional domain concepts such as *people*, *organization*, and *work-for*.

Concepts Each problem definition can contain three main types of concepts (nodes).

[3]Please note this is just an example of a learning problem and does not have anything to do with the main functionality of the framework.

Basic Concepts define the structure of the input of the learning problem. For instance *sentence*, *phrase*, and *word* are all basic concepts that can be defined in the EMR task.

Compositional Concepts are used to define the many-to-many relationships between the basic concepts. Here, the *pair* concept in the EMR task is a compositional concept. This is used as the basic concept for the relation extraction task.

Decision Concepts are derived concepts that are usually the outputs of the problem and subject to prediction. They are derived from basic or compositional concepts. The *people*, *organization*, and *work-for* are examples of derived concepts in the EMR conceptual graph. Following is a partial snippet showing the definition of basic and compositional concepts for the EMR task.

```
word = Concept(name='word')
phrase = Concept(name='phrase')
sentence = Concept(name='sentence')
pair = Concept(name='pair')
```

The following snippet also shows the definition of some derived concepts in the EMR example.

```
entity = phrase(name='entity')
people = entity(name='people')
org = entity(name='organization')
location = entity(name='location')
work_for = pair(name='work_for')
located_in = pair(name='located_in')
```

The *entity*, *people*, *organization* and *location* are the derived concepts from the *phrase* concept, and the rest are derived from the *pair* concept.

Edges

After defining the concepts, the user should specify existing relationships between them as edges in the conceptual graph. Edges are used to either map instances from one concept to another or generate instances of a concept from another concept. DomiKnowS only supports a set of predefined edge types, namely *is_a*, *has_a*, and *contains*.

is_a is automatically defined between a derived concept and its parent. In the EMR example, there is an *is_a* edge between *people* and *entity*. The *is_a* edge is mostly used to introduce hierarchical constraints and relate the basic and derived concepts.

Has_a connects a compositional concept to its components (also referred to as *arguments*). In the EMR example, *pair* concept has two *has_a* edges to the *phrase* concept to specify the *arg1* and *arg2* of the composition. We allow an arbitrary number of arguments in a *has_a* relationship, see below.

```
pair.has_a(arg1=phrase, arg2=phrase)
```

Contains edge defines a one-to-many relationship for two concepts to represent a (parent, child) relationship between them. Here, the number of parents of a concept is not necessarily limited to be only one. Following is a sample snippet to define a *contains* edge between *sentence* and *phrase*:

```
sentence.contains(phrase)
```

Global Constraints

The constraint definition is the part where the prior knowledge of the problem is defined to enable domain integration. The constraints of each task should be defined on top of the problem using the specified concepts and relationships there.

The constraints can be 1) automatically inferred from the conceptual graph structure, 2) extracted from the standard ontology formalism (here OWL[4]), 3) explicitly defined using the logical constraint

[4]Ontology Web Language

language of DomiKnowS. The framework internally uses the defined constraints at the training time or the inference-time optimization depending on the integration method selected for the task. Here is an example of a constraint written in DomiKnowS's logical constraint language for the EMR task:

```
ifL(work_for('x'), andL(people(path= ('x',arg1)), organization(path='x',arg2)))
```

The above constraint indicates that a *work_for* relationship only holds between *people* and *organization*. Other syntactic variations of this constraint are shown in the Appendix.

To process constraints, DomiKnowS maps those to a set of equivalent algebraic inequalities or their soft logic interpretation depending on the integration method. We discuss this more in Section 5.1.3.2.

Following is an example of the mapping between OWL constraint, graph structure, and the logic Python constraint. The ontology definition in OWL:

```
<owl:ObjectProperty rdf:ID="work_for">
    <rdfs:domain rdf:resource="#people"/>
    <rdfs:range rdf:resource="#organization"/>
</owl:ObjectProperty>
```

or equivalent graph structure definition:

```
work_for.has_a(arg1=people, arg2=organization)
```

DomiKnowS's constrain language representation:

```
ifL(work_for('x'), andL(people(path= ('x',arg1)), organization(path='x',arg2)))
```

All three above constraints represent the same knowledge that a *work_for* relationship only holds between *people* and *organization*.

In order to map this logical constraint to ILP, the solver collects sets of candidates for each case in the constraint's concepts.

ILP inequalities are created for each of the combinations of candidates' sets. The internal nested *andL* logical expression is translated to a set of three algebraic inequalities. The new variable *varAND*) is created to transfer the result of the internal expression into the external one.

```
varAND <= varPhraseIsPeople
varAND <= varPhraseIsOrganization
varPhraseIsPeople + varPhraseIsOrganization <= varAND + 1
```

External *ifL* expression is translated to a single algebraic inequality (refers to the variable *varAND*):

```
varPhraseIsWorkFor <= varAND
```

5.1.2.2 Model Declaration

Model declaration phase is about defining the computational units of the task. The basic building blocks of the model in DomiKnowS are *sensors* and *learners*, which are used to define either deterministic or probabilistic functionalities of the model. Sensor/Learners interact with the conceptual graph by defining properties on the concepts (nodes). Each sensor/learner receives a set of inputs either from the raw data or property values on the graph and introduces new property values. Sensors are computational units with no trainable parameters; and learners are the ones which contain the neural models. As stated before, the model declaration phase only defines the connection of the graph properties to the computational units and the execution is done later by the program instances.

The user can use any deep learning architecture compatible with PyTorch modules alongside the set of pre-designed and commonly used neural architectures currently in the framework. To facilitate modeling different architectures and computational algorithms in DomiKnowS, we provide a set of predefined sensors to do basic mathematical operations and linguistic feature extraction. Following is a short snippet of defining some sensors/learners for the EMR task.

```
phrase['w2v'] = FunctionalSensor('text', forward=word2vec)
phrase[people] = ModuleLearner('w2v', module=Classifier(FEATURE_DIM))
pair[work_for] = ModuleLearner('emb', module=Classifier(FEATURE_DIM*2))
```

In this example, the sensor *Word2Vec* is used to obtain token representations from the "text" property of each *phrase*. There is also a straightforward linear neural model to classify *phrases* and *pairs* into different classes such as *people*, *organization*, etc.

5.1.3 Learning & Evaluation in DomiKnowS

To execute the defined model considering the specified conceptual graph, DomiKnowS uses program instances. A program instance is responsible to run the model, apply loss functions, optimize the parameters, connect the output decisions to the inference algorithms, and generate the final results and metrics. Executing the program instance relies on the problem graph, model declaration, dataloaders, and a backbone data structure called DataNode. **DataLoader** provides an iterable object to loop over the data. **DataNode** is an instance of the conceptual graph to keep track of the data instances and store the computational results of the sensors and learners. For the EMR task, the program definition is as follows:

```
program = Program(graph, poi=(sentence, phrase, pair), loss=NBCrossEntropyLoss(),
↪   metric=PRF1())
```

Here, the concepts passed to the *poi* field specifies the training points of the program. This enables the user to train the task based on any subsets of the concepts defined in the model.

The user should specify the domain knowledge integration method for each program instance. The available methods for integration are discussed in the next sections. After initializing the program, the user can call *train*, *test*, and *prediction* functionalities to train and evaluate the designed model. The below snippet is to run training and evaluation on the EMR task:

```
program.train(train_reader, test_reader, epochs=10, Optim=torch.optim.SGD(param,
↪   lr=.001))
program.test(new_test_reader)
```

Here, the user will specify the dataloaders for different sets of data and the hyper-parameters required to train the model.

5.1.3.1 Program Composition

The Program instances allow the user to define different training tasks without extra effort to change the underlying models. For example, one can define end-to-end models, pipelines, and two-step tuning paradigms just by defining different program instances and calling them one after another. For instance, we can seamlessly switch between the following learning paradigm variations on the EMR task.

End-To-End training:

```
program = Program(graph, poi=(phrase, sentence, pair))
program.train()
```

Pre-train phrase then just train on the pairs:

```
program_1 = Program(graph, poi=(phrase, sentence))
program_2 = Program(graph, poi=(pair))
program_1.train(); program_2.train()
```

Pre-train phrase and use the result in the end-to-end training:

```
program_1 = POIProgram(graph, poi=(phrase, sentence), ...)
program_2 = POIProgram(graph, poi=(phrase, sentence, pair), ...)
program_1.train(...)
program_2.train(...)
```

5.1.3.2 Inference and Optimization

DomiKnowS provides access to a set of approaches to integrate background knowledge in the form of constraints on the output decisions or latent variables/concepts. Currently, DomiKnowS addresses three different paradigms for integration: 1) Learning + prediction time inference (L+I) 2) Training-time integration with hard constraints 3) Training-time integration with soft constraints. The first method, which we refer to as enforcing global constraints, can also be combined and applied on top of the second and third approaches at inference time.

Prediction-time Inference: In the back-end of DomiKnowS, ILP [5] solvers are used to make inferences under global linear constraints [138]. The constraints are denoted by $\mathcal{C}(\cdot) \leq 0$. Without loss of generality, we can denote the structured output as a binary vector $y \in \mathcal{R}^n$. Given local predictions $F\theta$ from the neural network, the global inference can be modeled to maximize the combination of log probability scores subject to the constraints [138, 60] as follows,

$$\begin{aligned} F^{*}\theta &= \arg\max_{y} \log F\theta^{\top} y \\ \text{subject to} \quad & \mathcal{C}(y) \leq 0. \end{aligned} \tag{5.1}$$

To handle constraints in ILP, we create variables for each local decision of instances and transform the logical constraints to algebraic inequalities [136] in terms of those variables. Auxiliary variables are added to represent the nested constraints. The inference method can be extended to support other approaches, such as probabilistic inference and dynamic programming, in the future without modifying the other parts of the framework.

[5] Integer Linear Programming

Integration of hard constraint in training: Here, we use our proposed inference-masked loss approach (IML) [60] which constructs a mask over local predictions based on the global inference results. The main intuition is to avoid updating the model based on local violations when the global inference can recover true labels from the current predictions. Given structured prediction $F\theta$ from a neural network and its global inference $F^*\theta$ subject to the constraints, IML is extended from negative log-likelihood as follows

$$\begin{aligned} &\mathcal{L}_{\text{IML}}\left(F\theta, Y\right) = \\ &-\left(\left(1 - F^*\theta\right) \odot Y\right)^\top \log F\theta, \end{aligned} \tag{5.2}$$

where Y is the structured ground-truth labels and $\odot$ indicates element-wise product. We implemented $\mathcal{L}_{\text{IML}(\lambda)}$ which balances between negative log-likelihood and IML with a factor λ as introduced in [60]. IML works best for very low-resource tasks where label disambiguation cannot be learned from the data but can be done based on the available relational constraints between output variables. The constraint mapping for the IML uses the same module in the DomiKnowSthat is implemented to use the global constraint optimization tool (here ILP).

Integration of soft constraints in training: We use the primal-dual formulation of constraints proposed in [117] to integrate soft constraints in training the models. Primal-Dual considers the constraints in the neural network training by augmenting the loss function using Lagrangian multipliers Λ for the violations from the constraints by the set of predictions. The constraints are regularized by a hinge function $\left[\mathcal{C}\left(F\left(\theta\right)\right)\right]$. The problem is formulated as a min-max optimization where it maximizes the Lagrangian function with the multipliers to enforce the constraints and minimize it with the parameters in the neural network. Instead of solving the min-max primal, we solve the max-min dual of the original problem.

$$\max_{\Lambda} \min_{\theta} \mathcal{L}\left(F\left(\theta\right), Y\right) \;\; \Lambda^\top \left[\mathcal{C}\left(F\left(\theta\right)\right)\right]. \tag{5.3}$$

During training, we optimize by minimization and maximization alternatively. With the Primal-Dual strategy, the model learns to obey the constraints without requiring any additional inference. Primal-Dual is less time-consuming at prediction time than the previous methods as it does not need an additional inference-time optimization phase. It can also be used in a semi-supervised setting, exploiting domain knowledge instead of labeled data. Handling constraints in Primal-Dual is done by mapping them to their respective soft logical interpretations [118].

It is an open research topic to identify which of the integration methods performs best for different tasks. However, DomiKnowS makes it effortless to use one problem specification and run all the aforementioned methods.

5.2 Constraint Integration Benchmark

Hurdle of Knowledge Integration Unfortunately, most prior research on knowledge integration has only focused on evaluating their proposed method compared to baseline DNN architectures that ignore the knowledge. Consequently, despite each method providing evidence of its effectiveness [67, 118], there is no comprehensive analysis that can provide a better understanding of the use cases, advantages, and disadvantages of methods, especially when compared with each other. The lack of such analysis has made it hard to apply these approaches to a more diverse set of tasks by a broader community and provide a clear comparison with existing methods. We mainly attribute this to three factors: 1) the lack of a standard benchmark with systematic baselines, 2) the difficulty of finding appropriate tasks where constraints are applicable, and 3) the lack of supporting libraries for implementing various integration techniques.

Due to these three factors, many research questions are left open for the community, such as (1) The difference in the performance of models when knowledge is integrated during inference vs. training or both, (2) The comparison of the influence of integration methods when combined with simpler vs. more complex baselines, (3) The effectiveness of training-time integration models on reducing the constraint violation, (4) The impact of data size on the effectiveness of the integration methods.

Common Ground for Comparison Our contribution is providing a common ground for comparing techniques for knowledge integration by collecting a new benchmark to facilitate research in this area. Our new benchmark, called GLUECons, contains a collection of tasks suitable for constraint integration, covering a spectrum of constraint complexity, from basic linear constraints such as mutual exclusivity to more complex constraints expressed in first-order logic with quantifiers. We organize the tasks in a repository with a unified structure where each task contains a set of input examples, their output annotations, and a set of constraints (written in first-order logic). We limit the scope of knowledge in GLUECons to logical constraints[6].

Selected Tasks GLUECons contains tasks ranging over five different types of problems categorized based on the type of available knowledge. This includes **1)** Classification with label dependencies: Mutual exclusivity in multiclass classification using MNIST [81] and Hierarchical image classification using CIFAR 100 [77], **2)** Self-Consistency in decisions: What-If Question Answering [168], Natural Language Inference [17], BeliefBank [71], **3)** Consistency with external knowledge: Entity and Relation Extraction using CONLL2003 [144], **4)** Structural Consistency: BIO Tagging, **5)** Constraints in (un/semi)supervised setting: MNIST Arithmetic and Sudoku. These tasks either use existing datasets or are extensions of existing tasks, reformulated so that the usage of knowledge is applicable to them. We equip these tasks with constraint specifications and baseline results.

Evaluation For a fair evaluation and to isolate the effect of the integration technique, we provide a repository of models and code for each task in both PyTorch [125] and DomiKnows [47] frameworks. As described before, DomiKnows makes it easier to consistently test different integration methods while the rest of the configurations remain unchanged. For a more comprehensive evaluation, we introduce a set of new criteria in addition to the original task performances to measure 1) the effectiveness of the techniques in increasing the consistency with knowledge, 2) the execution run-time, 3) the effectiveness of methods in the low-data regime, 4) the ability to reduce the need for complex models, and 5) the ability to express various forms of knowledge.

Baselines We analyze and evaluate a set of knowledge integration methods to serve as baselines for GLUECons. Our baselines cover a set of fundamentally different integration methods, where the integration is addressed either during inference or training of DNNs. GLUECons can be used as blueprints to highlight the importance of integrating constraints with DNNs for different types of tasks and provides inspiration for building such constraints when working on new tasks.

[6]Throughout this paper, we use the terms constraint integration, knowledge integration, or integration methods interchangeably to refer to the process of integration of knowledge into the DNNs.

5.2.1 Constraint Integration in Prior Research

Knowledge integration, often, is considered a subset of Neuro-symbolic [39, 7, 69] approaches that build on the intersection of neural learning and symbolic reasoning. **(author?)** surveyed prior research on knowledge integration in three directions: knowledge source, knowledge representation, and the stage of knowledge integration. **(author?)** has also studied existing methods where the integration can be done through either transforming the input data, the loss function, or the model architecture itself. Knowledge integration has also been investigated in probabilistic learning frameworks [38, 135, 12] and their modern extensions which use neural learning [101, 69, 176]. Recent research has explored knowledge integration via bypassing the formal representations and expressing knowledge in the form of natural language as a part of the textual input [143, 26]. As of formal representations, knowledge integration has been addressed at both inference [83, 148, 31] and training time [67, 118, 182].

Inference-Time Integration The inference-based integration techniques optimize over the output decisions of a DNN, where the solution is restricted by a set of constraints expressing the knowledge [138, 23].

These methods aim at finding a valid set of decisions given the constraints, while their objective is formed using the output scores/probabilities generated by the learning models. As a result of this fixed objective and the fact that approximation approaches are generally used to find the best solution, we expect that the type of optimization technique will not significantly affect the performance of inference-time integration methods –our results discussed in the later sections provide multiple pieces of evidence confirming this hypothesis. Prior research has investigated such integration by using variants of beam search [62, 15, 31], path search algorithm [98], linear programming [138, 139, 23], finite-state/push-down Automata [41], or applying gradient-based optimization at inference [83, 84]. We use Integer Linear Programming (ILP) [138, 139] approach to evaluate the integration of the constraints at inference time. We use off-the-shelf ILP tools that perform an efficient search and offer a natural way to integrate constraints. However, constraints should be converted to a linear form to be able to exploit these tools [47, 76, 75].

Training-Time Integration Several recent techniques have been proposed for knowledge integration at training time [118, 67, 182]. Using constraints during training usually requires finding a differentiable function expressing constraint violation. This will help to train the model to minimize the violations as a part of the loss function. Integrating knowledge in the training loop of DNNs is a challenging task. However, it can be more rewarding than the inference-based integration methods as it reduces the computational overhead by alleviating the need for using constraints during inference. Although such methods cannot guarantee that the output decisions would follow the given constraints without applying further operations at inference-time, they can substantially improve the consistency with the constraints [88]. Prior research has investigated this through various soft interpretations of logic rules [118, 9], rule-regularized supervision [67, 59], re-enforcement learning [186], and black-box semantic [182] or sampling [2] loss functions, which directly train the network parameters to output a solution that obeys the constraints.

To cover a variety of techniques based on the previous research, we select Primal-Dual (PD) [118] and Sampling-Loss (SampL) [2] methods as baselines for our new benchmark. The PD approach relies on a soft logic interpretation of constraints, while the SampL is a black-box constraint integration. We discuss some of the existing methods in more detail in Section 'Baselines.'

Applications and Tasks Constraint integration has been investigated for several applications in prior research including SQL query generation [148], program synthesize [10, 46], semantic parsing [27, 82], question answering [9], entity and relation extraction [59], sentiment analysis [67], visual question answering [69], image captioning [8], and even text generation [98].

5.2.2 Criteria of Evaluation

We extend the evaluation of the constraint integration methods beyond measuring task performance. The list of proposed evaluation criteria for such an extended comparison is as follows.

Individual metrics of each task: The first criterion to evaluate the methods is the conventional metric of each task, such as accuracy or precision/recall/F1 measures.

Constraint Violation: Even when the integration method cannot improve the model's performance, improving the consistency of its predictions will make the neural models more reliable. A consistency measure quantifies the success of the integration method in training a neural network to follow the given constraints. We measure consistency in terms of constraint violation. We compute the ratio of violated constraints over all predicted outputs. A smaller number indicates fewer constraint violations and, consequently, a higher consistency with the available knowledge.

Execution Run-Time: Another critical factor in comparing the constraint integration methods is the run-time overhead. This factor becomes even more critical when the integration happens during inference. This criterion helps in analyzing the adequacy of each technique for each application based on the available resources and the time sensitivity of the decision-making for that application. We measure this evaluation criteria by simply computing the execution time of each integration method both during training and inference. This metric can reflect the overhead of each integration method more accurately by taking into account the new parameters that should be optimized and the additional computations with respect to the complexity of the constraints.

Low-data vs full-data performance: For many problems, there is no large data available either due to the high cost or infeasibility of obtaining labeled data. Integrating constraints with deep neural learning has been most promising in such low-resource settings [118, 59]. We measure the improvement resulting from the integration methods on both low and full data. This evaluation will help in choosing the most impactful integration method based on the amount of available data when trying to apply integration methods to a specific task.

Simple baseline vs Complex baseline: An expected impact of constraint integration in DNNs is to alleviate the need for a large set of parameters and achieve the same performance using a smaller/simpler model. Additionally, it is important to evaluate whether the integration method can only affect the smaller network or the very large SOTA models can be improved too. This will indicate whether large networks/pre-trained models can already capture the underlying knowledge from the data or explicit constraint integration is needed to inject such knowledge. In addition to the number of parameters, this metric also explores whether knowledge integration can reduce the need for pre-training. This is especially important for the natural language domain, where large pre-trained language models prevail.

Constraint Complexity: This criterion evaluates the limitations of each method for integrating different types of knowledge. Some methods consider the constraints a black box with arbitrary complexity, while others may only model a specific form of constraint. This criterion specifies the form/complexity of the constraints that are supported by each technique. To evaluate this, we characterize a set of constraint complexity levels and evaluate whether each technique can model such constraints.

5.2.3 Selected Tasks

GLUECons aims to provide a basis for comparing constraint integration methods. We have selected/created a collection of tasks where constraints can potentially play an important role in solving them. We provide five different problem categories containing a total of nine tasks. More details of tasks' constraints are available in the Appendix. This collection includes a spectrum of very classic tasks for structured output prediction, such as multi-class classification to more involved structures and knowledge, such as entity relation extraction and Sudoku.

5.2.3.1 Classification with Label Dependency

Simple Image Classification. In this task, we utilize the classic MNIST [40] dataset and classify images of handwritten digits in the range of 0 to 9. The constraint used here is the mutual exclusivity of the ten-digit classes. Each image can only have one valid digit label as expressed in the following constraint,

$$\text{IF } digit_i x \Rightarrow \neg \vee_{j=!i}^{j \in 0-9} digit_j x,$$

where $digit_i x$ is 1 if the model has predicted x to be an image representing the digit i. This task is used as a basic validation of the constraint integration methods, though it is not very challenging and can also be addressed by a "Softmax" function.

Hierarchical Image Classification. The hierarchical relationships between labels present a more complex label dependency in multi-label and multi-class tasks. We use the CIFAR-100 [78], which includes 100 image classes, each belonging to 20 parent classes forming a hierarchical structure.

This dataset with 60k images is an extension of the classic CIFAR-10 [78]. To create a smaller dataset, we select 10% of these 60k images. For this task, the output is a set of labels for each image, including one label for each level. The constraints are defined as,

$$\text{IF } L_1 \subset L_2 : L_1 x \Rightarrow L_2 x,$$

where L_1 and L_2 are labels, L_1x is $True$ only if the models assigns label L_1 to x, and $L_1 \subset L_2$ indicates that L_1 is a subclass of L_2.

5.2.3.2 Self Consistency in Decisions

DNNs are subject to inconsistency over multiple decisions while being adept at answering specific questions [21]. Here, we choose three tasks to evaluate whether constraints help ensure consistency between decisions.

Causal Reasoning. WIQA [168] is a question-answering (QA) task that aims to find the line of causal reasoning by tracking the causal relationships between cause and effect entities in a document. The dataset contains 3993 questions. Following [9], we impose symmetry and transitivity constraints on the sets of related questions. For example, the symmetry constraint is defined as follows: $symmetricq, \neg q \Rightarrow Fq, C \wedge \neg F \neg q, C$ where q and $\neg q$ represent the question and its negated variation, C denotes the document, and $\neg F$ is the opposite of the answer F.

Natural Language Inference. Natural Language Inference (NLI) is the task of evaluating a hypothesis given a premise, both expressed in natural language text. Each example contains a premise (p), hypothesis (h), and a label/output (l) which indicates whether h is “entailed,” “contradicted”, or “neutral” by p.

Here, we evaluate whether NLI models benefit from consistency rules based on logical dependencies. We use the SNLI [17] dataset, which includes 500k examples for training and 10k for evaluation. Furthermore, we include A_{1000}^{ESIM} [111], which is an augmented set over the original dataset containing more related hypotheses and premise pairs to enforce the constraints. Four consistency constraints (symmetric/inverse, transitive) are defined based on the (Hypothesis, Premise) pairs. An example constraint is as follows:

$$\mathrm{neutral}\,(h, p) \Rightarrow \neg\,\mathrm{contradictory}\,(p, h)\,,$$

where $\mathrm{neutral}\,(h, p)$ is *True* if h is undetermined given p. The complete constraints are described in [111].

Belief Network Consistency. The main goal of this task is to impose global belief constraints to persuade models to have consistent beliefs. As humans, when we reason, we often rely upon our previous beliefs about the world, whether true or false. We can always change our minds about previous information based on new information, but new beliefs should not contradict previous ones. Here, entities and their properties are used as facts. We form a global belief network that must be consistent with those derived from a given knowledge base. We use Belief Bank [71] dataset to evaluate the consistency perseverance of various techniques. The dataset consists of 91 entities and $23k$ ($2k$ train, $1k$ dev, $20k$ test) related facts extracted from ConceptNet [158]. There are $4k$ positive and negative implications between the facts in the form of a constraint graph. For example, the fact "Is a bird" would imply "can fly," and the fact "can fly" refute the fact "Is a dog". Formally, the constraints are defined as follows:

$$\begin{aligned}
&\forall F_1, F_2 \in \text{Facts};\\
&\qquad \text{IF } F_1, F_2 \in \text{Pos Imp} \Rightarrow \neg F_1 x \vee F_2 x\\
&\qquad \text{IF } F_1, F_2 \in \text{Neg Imp} \Rightarrow \neg F_1 x \vee \neg F_2 x,
\end{aligned}$$

"Pos Imp" means a positive implication.

5.2.3.3 Consistency with External Knowledge

This set of tasks evaluates the constraint integration methods in applying external knowledge to the DNNs' outputs.

Entity Mention and Relation Extraction (EMR). This task is to extract entities and their relationships from a document. Here, we focus on the CoNLL2003 [144] dataset, which contains about 1400 articles. There are two types of constraints involved in this task: 1) mutual exclusivity between entity/relationship labels and 2) a restriction on the types of entities that may engage in certain relationships. An example constraint between entities and relationship types is as follows:

$$\text{IF Work\_for}x1, x2 \Rightarrow \text{Person}x1 \land \text{Org}x2,$$

where $\text{Predicate}x$ is $True$ if the network predicted input x to be of type Predicate.

5.2.3.4 Structural Consistency

In this set of tasks, we evaluate the impact of constraint integration methods in incorporating structural knowledge over the task's outputs.

BIO Tagging. The BIO tagging task aims to identify spans in sentences by tagging each token with one of the "Begin," "Inside," and "Outside" labels. Each tagging output belongs to a discrete set of BIO tags $T \in$ ['O', 'I-*', 'B-*'], where '*' can be any type of entity. Words tagged with O are outside of named entities, while the 'B-*' and 'I-*' tags are used as an entity's beginning and inside parts. We use the CoNLL-2003 [144] benchmark to evaluate this task. This dataset includes 1393 articles and 22137 sentences. The constraints of the BIO tagging task are valid BIO sequential transitions; for example, the "before" constraint is defined as follows:

$$\text{If } Ix_i \; 1 \rightarrow Bx_i,$$

where 'B-*' tag should appear before 'I-*' tag. x_i and x_{i1} are any two consecutive tokens.

5.2.3.5 (Un/Semi) Supervised Learning

We select a set of tasks for which the constraints can alleviate the need for direct supervision and provide a distant signal for training DNNs.

Arithmetic Operation as Supervision for Digit Classification. We use the MNIST Arithmetic [14] dataset. The goal is to train the digit classifiers by receiving supervision, merely, from the sum of digit pairs. For example, for image pairs of 5 and 3 in the training data, we only know their labels' sum is 8. This dataset is relatively large, containing 10k image pairs for training and 5k for testing. This task's constraint forces the networks to produce predictions for pairs of images where the summation matches the ground-truth sum. The following logical expression is an example constraint for this task:

$$S\{img_1, img_2\} \Rightarrow$$
$$\bigvee_{M=max0,S-9}^{M=minS,9} Mimg_1 \wedge \{S-M\}img_2,$$

where $S\{img_1, img_2\}$ indicates that the given summation label is S and $Mimg_i$ indicates that the ith image has the label M.

Sudoku. This task evaluates whether constraint integration methods can help DNNs to solve a combinatorial search problem such as Sudoku. Here, integration methods are used as an inference algorithm with the objective of solving one Sudoku, while the only source of supervision is the Sudoku constraints. As learning cannot be generalized in this setting, it should be repeated for each input. The input is one Sudoku table partially filled with numbers, and the task is to fill in a number in each cell such that: "There should not be two cells in each row/block/column, with the same value" or formally defined as:

$$\text{IF } \mathrm{digit_i}x\wedge$$
$$\text{same\_row}x,y \vee \text{same\_col}x,y \vee \text{same\_block}x,y$$
$$\Rightarrow \neg\, \mathrm{digit_i}y,$$

where x and y are variables regarding the cells of the table, $i \in 0, n$ for a $n * n$ Sudoku, $digit_i x$ is $True$ only if the value of x is predicted to be i. For this task, we use an incomplete $9 * 9$ Sudoku for the full-data setting and a $6 * 6$ Sudoku representing the low-data setting.

5.2.4 Baselines

For constraints during training, we use the two following approaches.

Primal-Dual (PD). This approach [118] converts the constrained optimization problem into a min-max optimization with Lagrangian multipliers for each constraint and augments the original loss of the neural models. This new loss value quantifies the amount of violation according to each constraint by means of a soft logic surrogate. During training, they optimize the decisions by minimizing the original violation, given the labels, and maximizing the Lagrangian multipliers to enforce the constraints. It is worth noting that all related work in which constraint violation is incorporated as a regularization term in the loss objective follows very similar variations of a similar optimization formulation.

inference-masked loss (IML) [59] This method applies a mask over the loss of the erroneous instances where an inference mechanism can recover correct labels for them. This approach has been mainly proposed to avoid overfitting training data when the number of training samples is low. By not updating all possible labels in the training set, the method relies on existing constraints and injected knowledge into the text to recover the correct labels by considering the correlation of different output variables in a structured prediction task. Although this method affects the neural network weights during training, it still requires applying methods at inference time.

Semantic-Loss (SemL) [182] This approach proposes a semantic loss function to bridge between the neural decision and logical constraints. The loss functions reflect the closeness of neural network outputs to satisfying the constraints by calculating the satisfaction given any black-box solver. In this case, the authors have used Logical circuits to implement an efficient search over the possible output space and find satisfying instantiations. After finding all the satisfying instantiations, the loss is simply promoting the sum of the conditional probability of such instantiations, assuming that each output is an independent decision. The loss formulation is $\mathrm{L^s}\alpha, \mathbf{p} \propto -\log {}_{\mathbf{x}\models\alpha}\, {}_{i:\mathbf{x}\models X_i}\, \mathbf{p}i\, {}_{i:\mathbf{x}\models\neg X_i} \left(1-\mathbf{p}_i\right)$, where α is the constraint, p is the set of probabilities outputs generated by the network, X is an instantiation of the outputs, and $\mathbf{x} \models \alpha$ means that an instance x is satisfying the α constraint.

Sampling-Loss (SampL). This approach [2] is an extension of the semantic loss [182] where instead of searching over all the possibilities in the output space to find satisfying cases, it randomly generates a set of assignments for each variable using the probability distribution of the neural network's output. The loss function is formed as:

$$L^S\alpha, p = \frac{{}_{x^i\in X\wedge x^i\models\alpha}\, p\left(x^i \mid p\right)}{{}_{x^i\in X}\, p\left(x^i \mid p\right)},$$

where X is the set of all possible assignments to all output variables, and x^i is one assignment. Here, α is one of the constraints.

Task	**Strong Baseline**	**Simple Baseline**
Img Cls.	CNN MLP	MLP
Hier. Img Cls.	Resnet18 MLP	-
NLI	RoBERTa MLP	-
Causal Rea.	RoBERTa MLP	BERT MLP
BIO Tagging	BERT MLP	LSTM MLP
NER	W2V BERT MLP	W2V LSTM MLP
Ari. Operation	CNN MLP	-
BeliefNet.	RoBERTa + MLP	W2V + MLP
Sudoku$*$	$(n*n*n)$ Vector	-

Table 5.1 Baselines for each task. The basic models we used are RoBERTa [96], BERT [42], W2V [108], CNN [81], and MLP. The simple baseline means fewer parameters. [KEYS: Cls.=classification, Hier.=Hierarchical, NLI= Natural Language Inference, Rea.=reasoning, Ari.=Arithmetic, BeliefNet=Belief Network, W2V=Word to Vec.]. $*$ For the Sudoku, the model is not a generalizable DNN and relies on the integration methods as an inference algorithm to solve one specific table.

To utilize the constraints during prediction, we use the following approaches.

Integer Linear Programming. (ILP) [138] is used to formulate an optimization objective in which we want to find the most probable solution for $\log F\theta^{\top} y$, subject to the constraints. Here, y is the unknown variable in the optimization objective, and $F\theta$ is the network's output probabilities for each variable in y. The constraints on y are formulated as $\mathcal{C}(y) \leq 0$.

Search and Dynamic Programming. for some of the proposed benchmarks, when applicable, we use the A^* search or Viterbi algorithm to choose the best output at prediction time given the generated probability distribution of the final trained network [98].

5.3 Experiments and Discussion

This section highlights our experimental findings using proposed baselines, tasks, and evaluation criteria. Details on experimental designs, training hyper-parameters, codes, models, and results can be found on our website[7]. The basic architectures for each task are shown in Table 5.1.

The results of the experiments are summarized in Table 5.3. The columns represent evaluation criteria, and the rows represent tasks and their baselines. Each task's model 'row' records the strong/simple baseline's performance without constraint integration, and below that, the improve-

[7]https://hlr.github.io/gluecons/

	Mut Excl.	Seq. Struc.	Lin. Const	Log. Const	Log + Quan	Prog Const
Softmax	✓	✗	✗	✗	✗	✗
PD	✓	✓	✓	NC	NC	✗
SampL	✓	✓	✓	✓	✓	✓
ILP	✓	✓	✓	NC	NC	✗
A^*	✓	✓	NG	NG	NG	✗

Table 5.2 The limitation of integration methods with respect to constraint types. [KEYS: NC= Needs Conversion, NG= No Generalization, Mut Excl.= Mutual Exclusivity, Seq.= Sequential, Struc.=Structure, Lin.= Linear, Log.= Logical, Const.= Constraint, Quan.=Quantifiers, Prog Const= Any Constraints encoded as a program, A^* is abbreviation for the A^* Search algorithm.]

ments or drops in performance after adding constraints are reported. Here, we summarize the findings of these experiments by answering the following questions.

What are the key differences in inference-time and training-time integration performance? Notably, using ILP only in inference time outperformed other baselines in most of the tasks. However, it fails to perform better than the training-time integration methods when the base model is wildly inaccurate in generating the probabilities for the final decisions. This phenomenon happened in our experiments in the semi-supervised setting and can be seen when comparing rows [#44, #45] to #46. In this case, inference alone cannot help correct the model, and global constraints should be used as a source of supervision to assist with the learning process.

ILP performs better than the training-time methods when applied to simpler baselines (see column simple baseline performance). However, the amount of improvement does not differ significantly when applying ILP to the simpler baselines compared to the strong ones. Additionally, the training-time methods perform relatively better on simpler baselines than the strong ones (either the drop is less or the improvement is higher) (compare columns 'Strong Baseline' and 'Simple Baseline' for '+PD' and '+SampL' rows.)

How does the size of data affect the performance of the integration techniques? The integration methods are exceptionally effective in the low-data regime when the constraints come from external knowledge or structural information. This becomes evident when we compare the results of 'EMR' and 'BIO tagging' with the 'Self Consistency in Decision Dependency' tasks in column 'Low

	Tasks	#	Models	Strong Baseline Performance	Simple Baseline Performance	Low data		Constraint Violation*	Run-Time^{ms}	
						Size	Performance		Training	Inference
Classification with Label Dependency	Simple Img Cls^{F1}	1	Model	94.23%	87.34%	5%	88.78%	7.17%	34	27.5
		2	PD	↑0.14%	↓1.14%		↑4.40%	8.32%	36.6	-
		3	SampL	↓1.17%	↑0.49%		↑3.19%	9.04%	39	-
		4	ILP	↑0.24%	↑1.60%		↑1.70%	-	-	31.5
		5	SampL ILP	↓0.52%	↑2.02%		↑4.40%	-	-	-
		6	PD ILP	↑0.32%	↑0.39%		↑4.40%	-	-	-
	Hierarchical Img Cls^{F1}	7	Model	58.03%	52.54%	10%	31.33%	39.26%	55.3	48.43
		8	SampL	↑0.39%	↑0.54%		↑2.18%	36.57%	58.2	-
		9	ILP	↑2.88%	↑3.18%		↑1.90%	-	-	55.2
		10	SampL ILP	↑2.42%	↑3.52%		↑3.82%	-	58.2	55.2
Self Consistency in Decision Dependency	Causal^{A} Reasoning	12	Model	74.77%	73.80%	30%	60.49%	8.60%	104	46.2
		13	PD	↑2.17%	↑1.98%		↑1.10%	11.36%	118.1	-
		14	SampL	↑2.54%	↑2.17%		↑1.63%	4.37%	119.4	-
		15	ILP	↑4.03%	↑4.51%		↑1.88%	-	-	59.2
		16	SampL ILP	↑4.15%	↑4.25%		↑2.11%	-	-	-
		17	PD ILP	↑3.60%	↑4.30%		↑1.76%	-	-	-
	NLI^{A}	18	Model	74.00%	-	10%	68.65%	9.48%	29.2	10.7
		19	PD	↑0.25%	-		↑3.25%	7.26%	31.7	-
		20	SampL	↑0.55%	-		↑0.95%	5.00%	29.8	-
		21	ILP	↑8.90%	-		↑7.75%	-	-	14.3
		22	SampL ILP	↑8.20%	-		↑7.05%	-	-	-
		23	PD ILP	↑8.75%	-		↑10.1%	-	-	-
	Belief^{F1} Network	24	Model	94.90%	84.46%	25%	94.36%	0.22%	8.3	7.57
		25	PD	↑0.94%	↑0.87%		↓0.49%	0.16%	23.59	-
		26	SampL	↓0.29%	↓0.95%		↓3.03%	0.01%	8.5	-
		27	ILP	↑0.21%	↓0.10%		↓0.97%	-	-	11
		28	SampL ILP	↑1.10%	↓3.19%		↓2.31%	-	-	-
		29	PD ILP	↑2.68%	↑1.60%		↑0.51%	-	-	-
Consistency with EK	EMR^{F1}	30	Model	90.15%	85.22%	20%	82.00%	20.32%	210	200
		31	PD	↓1.00%	↓0.30%		↑2.42%	16%	245	-
		32	SampL	↓0.30%	↑0.50%		↑3.36%	16.7%	280	-
		33	ILP	↑3.02%	↑4.10%		↑8.86%	-	-	226
		34	SampL ILP	↑2.40%	-		↑7.83%	-	-	-
		35	PD ILP	↑1.64%	-		↑8.15%	-	-	-
Structural Consistency	BIO^{F1} Tagging	36	Model	89.56%	82.77%	30%	75.36%	2.19%	361.2	263.2
		37	PD	↑0.97%	↑0.04%		↑1.25%	0.99%	389.1	-
		38	SampL	↓0.17%	↑0.73%		↑2.62%	0.16%	429.8	-
		39	ILP	↑0.61%	↑2.96%		↑3.01%	-	-	312
		40	SampL ILP	↑0.08%	↑2.43%		↑2.80%	-	-	-
		41	PD ILP	↑1.07%	↑1.83%		↑2.73%	-	-	-
		42	A^* search	↑0.59%	↑2.97%		↑3.03%	-	-	-
Constraints in (Un/Semi)Supervision	Arithmetic Supervision for Digit Classification^{A}	43	Model	9.01%	-	5%	10.32%	96.92%	13.6	-
		44	PD	↑89.39%	-		↑85.01%	3.18%	197	-
		45	SampL	↑89.55%	-		↑85.60%	2.86%	90.2	-
		46	ILP	↓2.11%	-		0.00%	-	-	-
		47	Supervised	↑89.53%	-		↑84.30%	2.86%	12.5	-
	Sudoku^{CS}	48	PD	96.00%	-	$6*6$ Table	100%	3.7%	-	-
		49	SampL	87.00%	-		100%	18.88%	-	-
		50	ILP	100%	-		100%	-	-	-

Table 5.3 Impact of constraint integration. F1, A, and CS are F1-measure, accuracy, and constraint satisfaction metrics, respectively, showing model performance. The full data of the Sudoku task is a $9*9$ table. *: The 'Constraint Violation' values are calculated for the strong baselines trained with full data. ms: Run-Time is computed per example/batch and is reported in milliseconds. ↑ indicates improvement and ↓ indicates a drop in the performance compared to the initial Model. Run times are recorded on a machine with Intel Core i9-9820X (10 cores, 3.30 GHz) CPU and Titan RTX with NVLink as GPU. [KEYS: EK= external knowledge.]

data/ Performance'. This is because such constraints can inject additional information into the models, compensating for the lack of training data. However, when constraints are built over the self-consistency of decisions, they are less helpful in low-data regimes (rows #12 to #29), though a positive impact is still visible in many cases. This observation can be justified since there are fewer applicable global constraints in-between examples in the low-data regime. Typically, batches of the full data may contain tens of relationships leading to consistency constraints over their output, while batches of the low data may contain fewer relationships. The same observation is also seen as batch sizes for training are smaller.

Does constraint integration reduce the constraint violation? Since our inference-time integration methods are searching for a solution consistent with the constraints, they always have a constraint violation rate of 0%. However, training-time integration methods cannot fully guarantee consistency. However, it is worth noting these methods have successfully reduced the constraint violation in our experiments even when the performance of the models is not substantially improved or is even slightly hurt (see rows #18 and #20, rows #24 and #26, and rows #30 to #32). In general, SampL had a more significant impact than PD on making models consistent with the available task knowledge (compare rows with '+PD' and '+SampL' in column 'Constraint Violation').

How do the integration methods perform on simpler baselines? According to our experiments, there is a significant difference between the performance of the integration methods applied to simple and strong baselines when the source of constraint was external (BIO tagging, EMR, Simple Image Cls, and Hierarchical Image Cls tasks). Moreover, we find that ILP applied to a simple baseline can sometimes achieve a better outcome than a strong model without constraints. This is, in particular, seen in the two cases of EMR and Causal Reasoning, where the difference between the simple and strong baselines is in using a pre-trained model. Thus, explicitly integrating knowledge can reduce the need for pre-training. In such settings, constraint integration compensates for pre-training a network with vast amounts of data for injecting domain knowledge for specific tasks. Additionally, the substantial influence of integration methods on simple baselines compared to strong ones in these specific tasks indicates that constraint integration is more effective when knowledge is not presumably learned (at some level) by available patterns in historical data used in the pre-raining of large language models.

How much time overhead is added through the integration process? While the inference-time method (ILP) has a computational overhead during inference, we have shown that this overhead can be minimized if a proper tool is used to solve the optimization problem (here, we use Gurobi[8]. It should be noted that training-time integration methods do not introduce additional overhead during inference; however, they typically have a high computational cost during training. In the case of our baselines, SampL has shown to be relatively more expensive than PD. This is because SampL has an additional cost for forming samples and evaluating the satisfaction of each sample.

What is the effect of combining inference-time and training-time integration methods? Our results show that combining inference-time and training-time methods mainly yields the highest performance on multiple tasks. For example, the performance on the NLI task on low-data can yield over 10% improvement with the combination of PD and ILP, while ILP on its own can only improve around 7%. The rationale behind these observations needs to be further investigated. However, this can be attributed to better local predictions of the training-time integration methods that make the inference-time prediction more accurate. A more considerable improvement is achieved over the initial models when these predictions are paired with global constraints during ILP (see rows #16, #28, #29, and #41).

What type of constraints can be integrated using each method? Table 5.2 summarizes the limitations of each constraint integration method to encode a specific type of knowledge. We have included "Softmax" in this table since it can be used to support mutual exclusivity directly in DNN. However, "Softmax" or similar functions are not extendable to more general forms of constraints. SampL is the most powerful method that is capable of encoding any arbitrary program as a constraint. This is because it only needs to evaluate each constraint based on its satisfaction or violation. A linear constraint can be directly imposed by PD and ILP methods. However, first-order logic constraints must be converted to linear constraints before they can be directly applied. Still,

[8]https://www.gurobi.com/

PD and ILP methods fail to generalize to any arbitrary programs as constraints. The A^* search can generally be used for mutual exclusivity and sequential constraints, but it cannot provide a generic solution for complex constraints as it requires finding a corresponding heuristic. [23] show A^* with constraints can be applied under certain conditions and when the feature function is decomposable.

CHAPTER 6

PROCEDURAL REASONING WITH LOGICAL CONSTRAINTS

This chapter discusses integrating constraints in procedural reasoning tasks, focusing on entity tracking. This task is a structured prediction challenge, where constraints help guide predictions about entities' actions and locations. Our goal is to improve the accuracy of model predictions by using constraint integration techniques.

Previous methods for tracking entities mainly used sequential inference, depending on hand-coded algorithms or CRF (Conditional Random Field) decoders [165] along with the Viterbi algorithm [54] to identify important sequences of actions. These approaches adjust entity locations to match the sequences of actions identified.

Our work introduces a new way to consider constraints globally in structured prediction tasks through the use of DomiKnowS [47]. Our main contribution is a global consistency approach that ensures predictions about actions and locations are coherent during entity tracking. We use an Integer Linear Programming (ILP) [139] method to enforce this consistency during inference.

We also use a generative model as a baseline to predict actions and locations. This model can predict locations not directly mentioned in the text and tackle different types of questions within a single architecture, like yes/no questions and extracting phrases. The generative model also allows for the consideration of more types of questions derived from the dataset's labels. For example, by looking at location annotations, we can determine if an entity is an input or output in a process. Taking into account these additional questions and the original dependencies between action and location queries, our evaluation framework checks how well the models understand the processes. We aim to improve both the coherence of the models and, possibly, their overall performance.

Applying ILP to a large set of question types with various difficulty levels and potential output space further motivates us to investigate the success of such inference techniques in properly resolving inconsistencies between inherently different decisions. We further use the procedural reasoning task to investigate the potential limitations of the current formulation for applying the ILP method to the model-generated probabilities and propose enhancements to the formulation by

Process	Participants			
Sentences	**plant**	**animal**	**bone**	**oil**
Before the process begins	?	?	-	-
1. Plants and animals die in a watery environment	watery environment	watery environment	-	-
2. Over time, sediments build over	sediment	sediment	-	-
3. The body decomposes	sediment	-	sediment	-
4. Gradually buried material becomes oil	-	-	-	sediment

Figure 6.1 Example of a procedural reasoning task in the Propara dataset. '-' indicates the entity is absent, and '?' indicates an unknown location.

considering additional factors such as the decision's prior probability, confidence (entropy), and accuracy. We evaluate the performance of the newly proposed inference method with a series of toy tasks regarding hierarchical classification and the close to real-world task of entity tracking tasks.

6.1 Procedural Reasoning Constraints

In this chapter, we examine the task of procedural reasoning with a focus on entity tracking. Our study uses the Propara dataset [33], which provides a series of procedural steps, a list of entities of interest, and the locations of these entities after each process step. An illustration from the dataset is shown in figure 6.1.

Understanding how entity locations change requires models to grasp the actions occurring at each step and their impact on each entity. Previous research [33] has explored this task with the dual objective of predicting both the actions affecting entities and their locations at every step. The models can predict an action as 'No Change', 'Move', 'Create', or 'Destroy', alongside a location drawn from the text. These action-location decisions follow a logical dependency, expressed as Action-Action and Action-Location constraints. We list these constraints below:

Action-Location Constraints

- Move: If an entity moves at step i, its location at step $i-1$ and step i must differ.
- Create: If an entity is created at step i, its location must be 'None' at step $i-1$ and either 'Unknown' or a valid phrase at step i.
- Destroy: If an entity is destroyed at step i, it must exist (not 'None') at step $i-1$, but its location at step i should be 'None'.
- No Change: If there's no change to an entity, its location at steps $i-1$ and i should match.

'No Change' can further be divided into 'Exists' and 'Does not Exist', indicating whether the location should be 'None'.

Action-Action Constraints

- Move: If an item moves at step i, it must have existed at step $i-1$. A 'destroy' action should not precede step i without an intervening 'create'.
- Create: An item created at step i should not exist at step $i-1$. There shouldn't be a 'create' or 'Move' before step i without a 'destroy' in between.
- Destroy: An item destroyed at step i must have been present at step $i-1$. A 'destroy' should not precede this without a 'create' in between.
- Exists: If an item is marked as existing at step i, it must have been present at step $i-1$, with no 'destroy' before step i without an intervening 'create.'
- Does not Exist: An item marked as non-existent at step i should not have been present at step $i-1$. There should be no 'create' or 'Move' before step i without a 'destroy' in between.

To model these action-action constraints in a sequential model like Conditional Random Field (CRF), previous work has differentiated the 'Does not Exist' action into 'O_C' (not created yet) and ‘O_D‘ (already destroyed). This distinction aids in modeling the dependencies between consecutive steps without examining the entire history of actions. The probabilities of moving between nodes in the CRF (actions) are based on dataset statistics (frequency of action sequences in the training data). Invalid sequences are assigned zero probability, preventing their selection as the final output by the inference (Viterbi algorithm). However, this might be unnecessary if an inference model can effectively search through possible sequences and apply constraints globally across non-consecutive steps.

6.2 Consistent Procedural Reasoning within Generative Models

In recent years, there’s been a notable increase in the use of generative models across various fields such as natural language processing [157], computer vision [29], robotics [134, 20], and more [190]. A key strength of these models, particularly when they’re fine-tuned with specific instructions, lies in their flexibility to tackle different tasks within a single framework by simply altering the input prompt to include task-specific metadata [155].

However, despite their remarkable success in numerous applications, Generative Models, including Large Language Models (LLMs), face challenges in tasks that require complex, multi-step reasoning [114, 91] and exhibit inconsistencies in their outputs [58, 24].

This section aims to assess how well generative models can process and reason about procedural information, focusing particularly on tracking the status of entities throughout a narrative. Successful tracking demands an understanding of the physical world, entity attributes, the impact of various actions, and the ability to reason across past, present, and future events. We propose that a model’s understanding of a process is incomplete if it cannot maintain consistency across related tasks. For example, recognizing an entity’s location change should inherently mean understanding that the entity has been moved, even when phrased differently.

Assessing a model’s proficiency in procedural reasoning requires a set of interconnected tasks. Previous research in entity tracking has seen models predict both actions and locations of entities at each step. Dependencies between actions have been managed in sequential models like CRFs to ensure consistency and leverage statistical patterns in action changes [33]. Without resorting to CRFs, some studies have introduced an inference stage that processes decisions sequentially, adjusting subsequent decisions based on earlier ones [193, 49].

Although entity tracking datasets traditionally only mark the status of entities at each step, this information can be enriched to infer actions. Previous studies have expanded the datasets to teach models to predict actions [33] explicitly and tested them on more complex concepts like identifying input and output entities [167]. Building on these foundations, we generate interconnected questions about entities and the overall process, forming consistency sets for evaluating LLMs' ability to reason with consistency. We broaden the scope of questions from mere location and action identification to more detailed inquiries like detecting location changes relative to time, determining input/output entities, offering multiple-choice action predictions, identifying entity conversions, pinpointing creation times, and recognizing boolean state changes.

By utilizing this enriched dataset, we aim to scrutinize the model's competency in procedural reasoning through its consistency in answering these interconnected questions. Additionally, we explore the effects of increased supervisory labels (varied question types and phrasings) and the application of a neuro-symbolic approach to enforce output constraints on the generative model's reasoning processes. Our contributions are summarized as follows:

- Extension of the Propara benchmark to incorporate consistency sets for enhanced procedural reasoning evaluation.
- Analysis of a generative model's consistent reasoning capabilities using the expanded consistency sets.
- Development and assessment of a neuro-symbolic method to directly apply constraints on the generative model's outputs during procedural reasoning.

6.2.1 Datasets and Consistency Protocols

6.2.1.1 Dataset

We employ the Propara dataset to construct sets for consistency checking. The consistency set would include various question types, where their answers have correlated reasoning steps. Utilizing the question-answering format enables us to standardize the structure of different reasoning tasks. Previous research [49] has explored the adaptation of the location detection task into question forms such as "What is the location of the entity '$entity$' at step i?". We expand upon these initial questions regarding entity locations with an extended assortment of questions.

- **After Location:** "Where is ent located right after the completion of step i?"
- **Prior Location:** "Where is ent located right before the commencement of step i?"
- **Multiple Actions:** "What is the condition of ent at step i? (options listed here)?"
- **Boolean Actions (move, create, destroy):** "Is the entity ent action-verb at step i?", encompassing 4 variations
- **Location Change:** "Does the location of entity ent alter during step i?"
- **Input and Alternate Input:** "Is ent contributing to the subsequent process?" and "Does ent exist before the upcoming process?"
- **Output and Alternate Output:** "Is ent a result of the following process?" and "Does ent remain after the process concludes?"
- **Timing (Create/Destroy):** "During which step is ent action-verb? (step options including 'Never')"
- **Existence Before/After:** "Does entity ent exist immediately (before/after) step i (begins/ends)?"
- **Specific Location Change:** "What becomes the location of ent after undergoing (action-verb) at step i?"
- **Combination of Action & Location:** "What is the status, preceding location, and subsequent location of ent at step i?"

The complexity of each question type may vary based on the level of reasoning needed for resolution. For example, identifying an entity as a process's outcome might necessitate an analysis across the entire process and its steps (the model could seek a shortcut by pinpointing the entity's last mention and determining the relevant action). This set of additional questions is intended to facilitate the assessment of the model's comprehension of the described processes.

6.2.1.2 Consistency Rules

Given that the augmented question sets are derived from a singular annotation source, several dependencies exist among their anticipated responses. Below, we enumerate several rules that should be adhered to among the responses:

- **Locational Continuity:** The location following step i should align with the location prior to step $i \ 1$.
- **Input Consistency:** Both input queries should yield identical responses.
- **Output Consistency:** Both output queries should yield identical responses.
- **Action Consistency:** Boolean actions should concur with the multi-action responses.
- **Timing and Action Validity:** The timing query should identify the step where the corresponding boolean action query affirms the action.
- **Input Validation and Initial State:** An input is valid only if the preceding location for step 1 is either known or unknown.
- **Output Verification and Final State:** An output is valid only if the final subsequent location is identified as either known or unknown.
- **Initial Action Restriction for Inputs:** If an entity is an input, its initial action cannot be labeled as "create."
- **Existential Continuity and Location:** If an entity exists, it must possess a definable location, whether known or unknown.

- **Implications of Action on Location:** The action at step i should correlate with its effects on the after locations for steps $i-1$ and i.

6.2.2 Experimental Evaluations

To evaluate the capacity of generative models for performing consistent reasoning within procedural narratives, we conduct two primary sets of experiments. Initially, we measure the model's performance in its base state—without any adjustments or modifications to its original predictions—under two scenarios: 1) training with the original dataset (comprising Location and Action information) and 2) training with a dataset that has been expanded to include additional information. These tests aim to determine whether the model's consistency and overall performance can be enhanced through the implicit cues gained during the training phase. Additionally, we utilize the rate of constraint satisfaction as a measure of the consistency of decisions. In a second series of experiments, we examine the performance of the model after implementing strategies designed to achieve 100% consistency in its decisions. We explore three methods to ensure decisional consistency: 1) a sequential consistency enforcement approach crafted by experts, which adjusts sequences of actions and corresponding locations; 2) a global inference strategy (Integer Linear Programming, ILP) that addresses inconsistencies by solving an optimization problem within the task's constraints, using only decisions from the original question types; and 3) the application of this global inference approach to rectify inconsistencies across all decision types, considering a comprehensive set of constraints.

Baseline (T5): The T5 model, known for its powerful encoder-decoder capabilities and pretraining on various downstream tasks with specific prefixes, serves as our baseline. Within this framework, we evaluate both the base and large variants of the T5 model, whether employing pre-trained parameters or after fine-tuning on the UnifiedQA—a general question-answering dataset. Our evaluation extends to how this benchmark model performs when fine-tuned on both the original and augmented versions of the Propara dataset.

Size	Pretrain	Augmented Set	Consistency Rate (%)	Action	Loc Total	Macro Avg	Micro Avg
Base	-	-	**56.76%**	70.25	63.19	66.72	66.49
	UnifiedQA	-	56.61%	70	63.61	66.30	66.13
	-	Yes	56.32%	68.89	63.56	66.72	66.52
	UnifiedQA	Yes	56.35%	71.32	63.24	67.28	67.02
Large	-	-	56.44%	72.1	65.34	68.72	68.50
	UnifiedQA	-	56.39%	70.96	65.81	68.39	68.22
	-	Yes	56.32%	72.28	**66.49**	**69.38**	**69.20**
	UnifiedQA	Yes	56.62%	**73.65**	64.50	69.07	68.77

Table 6.1 The results for the fine-tuned version of various T5 models on the original and augmented set of the Propara dataset.

Model	Method	Action	Loc Total	Macro Avg	Micro Avg
Base	Expert-Designed	69.05	63.925	66.487	66.32
	ILP	70.13	64.707	67.418	67.24
Base - UnifiedQA	Expert-Designed	65.83	62.979	64.405	64.31
	ILP	68.51	63.401	65.956	65.79
Large	Expert-Designed	70.96	65.022	67.991	67.80
	ILP	**72.34**	**65.029**	**68.684**	**68.44**
Large - UnifiedQA	Expert-Designed	70.43	63.243	66.836	66.60
	ILP	70.97	64.607	67.789	67.58

Table 6.2 The results of applying various inference techniques on the T5 baseline, comparing the traditional ILP method and the Expert-Designed inference methods.

6.2.2.1 Initial Model Performance Evaluation

This phase focuses on assessing the baseline model's inherent performance without applying consistency enforcement strategies.

Table 6.1 details the outcomes from testing the model under all configurations without modifying its decision-making post hoc. It's noteworthy that the models were supervised either with the original dataset, incorporating only action and location queries, or with the enhanced dataset.

From Table 6.1, we deduce four main observations: 1) performance escalates with an increase in model size, 2) contrary to expectations, leveraging pre-training on the Unified-QA dataset adversely affects performance on this specific task, 3) incorporating the enhanced dataset markedly boosts the model's efficacy, and 4) additional pre-training, fine-tuning for consistency, or enlarging the model size does not compromise the model's ability to render consistent reasoning across the original set of questions.

Model	Method	Action	Action - F1	Loc Total	Macro Avg	Micro Avg
Base	Expert-Designed	69.05	53.69	61.937	65.493	65.26
	ILP	70.20	56.19	60.633	65.417	65.10
	ILP (aug)	70.25	57.02	61.622	65.936	65.65
Base - UnifiedQA	Expert-Designed	69.77	53.71	64.189	66.979	66.80
	ILP	71.86	58.38	64.555	68.207	67.97
	ILP (aug)	69.23	57.08	61.043	65.136	64.87
Large	Expert-Designed	70.97	55.85	63.713	67.341	67.10
	ILP	71.56	59.26	63.145	67.352	67.08
	ILP (aug)	72.16	60.01	**64.608**	**68.384**	**68.14**
Large - UnifiedQA	Expert-Designed	72.94	60.15	63.399	68.169	67.86
	ILP	**73.54**	**61.2**	63.355	68.447	68.11
	ILP (aug)	73.24	60.65	63.610	68.425	68.11

Table 6.3 The results of applying inference methods on the T5 base models when trained on augmented training sets. ILP (aug) refers to the optimization problem when considering the decisions from the augmented question types in the constraint satisfaction process.

6.2.2.2 Logical Post-Processing Evaluation

Table 6.2 illustrates the outcomes of implementing the three aforementioned inference methodologies on the model's decisions. We assess the impact of these inference strategies on the precision of responses to individual question types and the F1 scores for action questions regarding creation, destruction, and movement. The rationale behind incorporating the F1 score is to mitigate the inflated accuracy rates for actions, which could arise if the model predominantly forecasts "No Change" or "Exists" across most steps. The essence of this task is to accurately forecast concrete actions, including creation, destruction, and movement. The F1 score thus helps highlight the significance of these actions while also accounting for any false positives.

As indicated in Table 6.2, the global inference method (ILP) generally equals or surpasses the expert-designed strategy in effectiveness, occasionally outdoing it by more than 1%. Table 6.3 further demonstrates that in most cases (three out of four), incorporating all question types into the logical reasoning phase substantially improves model performance compared to other inference approaches.

Our experiments with the procedural reasoning task indicated that applying the ILP method to decisions that have various output sizes mostly alters the decisions made in larger output spaces in favor of preserving the original model decisions made in lower output spaces. For instance, the

ILP method applied to the two question types, Actions and Locations, heavily updated the location decisions while rarely changing the decisions regarding actions. This can be seen by comparing the base performance of models present in Table 6.1 with their corresponding ILP results in Table 6.3.

6.3 Enhanced Inference toward consistent Procedural Reasoning

To comprehend procedural text, multiple neural models collaborate to establish temporal relationships between actions, reveal semantic relations, and discern entity properties like location and temperature [50, 16, 70]. Each model responsible for these decisions exhibits distinct decision characteristics, output sizes, uncertainty levels, and varying excepted accuracy levels. Resolving inconsistencies and aligning these diverse neural decisions is crucial for comprehensively understanding the underlying process.

In many instances, raw model outputs lack usability without enforcing consistency. In tasks like hierarchical image classification, with independent models for each hierarchy level, outputs should adhere to the known hierarchical relationships. For example, the combination "Plant, Chair, Armchair" lacks validity and requires post-processing for downstream applications. A similar requirement extends to generative models in text summarization [98] and image captioning [8].

Prior studies have proposed techniques for handling inconsistencies in correlated decisions during both inference [55, 148, 31, 23, 59] and training [67, 118, 182] of neural models. This proposal focuses on resolving these inconsistencies at inference, where the goal is to ensure that outputs align with task constraints while preserving or enhancing the original model performance without training.

Integer Linear Programming (ILP) [140] stands out as a robust approach to addressing decision inconsistencies. ILP is a global optimization framework that seeks to find the best configuration of variables while meeting specified constraints. It is known for its efficiency and capability to produce globally optimal solutions, distinguishing it from alternatives like beam search. The ILP formulation is as follows:

$$\begin{gathered}\text{Objective : Maximize} \quad P^\top y \\ \text{subject to} \quad \mathcal{C}(y) \leq 0,\end{gathered} \tag{6.1}$$

where constraints are denoted by $\mathcal{C}(\cdot) \leq 0$, decision variables are denoted by $y \in \mathcal{R}^n$, and the vector containing the local weights of variables are denoted by P. To apply ILP to resolve conflicts from decisions of neural models, prior work [137, 127, 121, 60] has defined P to be the vector of raw probabilities of local decisions, $P = p^1, ..., p^n$, where p^i corresponds to the probability generated from a certain model for the ith decision variable (y_i). The global inference is modeled to maximize the combination of probabilities subject to constraints.

Previous use of ILP has proven effective in ensuring decision consistency in certain cases [51] but did not address model heterogeneity. This problem becomes more dominant in scenarios where output probabilities come from independent models, making them less directly comparable. We extend the ILP formulation to address this limitation beyond just considering the raw model probabilities. Instead, we map these raw scores into globally comparable values, facilitating a more balanced global optimization. We achieve this by incorporating additional information, such as decision confidence, expected model accuracy, and estimated prior probabilities. While previous studies have explored the integration of uncertainty in modeling the training objective [181, 56, 197], our work represents a novel effort in systematically incorporating multiple factors of this nature into the inference process for interrelated decisions to leverage external knowledge effectively.

6.3.1 Method

Our objective is to devise an improved scoring system, generating new local variable weights (importance) W in the ILP formulation. Thus, we modify the original objective function as follows:

$$\text{Maximize} \quad W^\top y, \tag{6.2}$$

where $W = w^1, ..., w^n$. To determine the new weights, we aim to find the scoring function G, which normalizes the local predictions of each model and maps them into globally comparable values. For each model m with multi-class decisions, we denote the output probabilities after applying a SoftMax layer as $P_m \subset P$. The scoring function G transforms these raw probabilities into new weights $W_m \subset W$ to indicate the importance of the variables within the ILP objective, i.e., $W_m = GP_m, m$. This subsection explores different options for the function G and provides an intuitive understanding of their rationale.

6.3.1.1 Prior Probability (Output Size)

We consider a normalization factor based on prior probabilities to facilitate fair comparison among decisions with varying output sizes. For an N-class output, the prior probability for each label is $\frac{1}{N}$ (assuming uniform distribution). This implies an inherent disadvantage for decisions made in larger output spaces. Thus, we normalize the raw probabilities by dividing them by the inverse of their respective priors and define $GP_m, m = P_m \times N$. This factor adjusts the weight of decisions relative to the size of the output space.

6.3.1.2 Entropy and Confidence

The outputs generated from models often exhibit varying levels of confidence. While raw probabilities alone may adequately indicate the model's confidence in individual Boolean decisions, a more sophisticated approach is required for assessing the models' confidence in multi-classification. We propose incorporating the entropy of the label distribution as an additional factor to assess the model's decision-making confidence. As lower entropy corresponds to higher confidence, we use the reverse of the entropy, normalized by the output size N, as a factor in forming the decision weight function $GP_m, m = P_m * \frac{N}{EntropyP_m}$.

6.3.1.3 Expected Models' Accuracy

Assigning higher weights to the probabilities generated by more accurate models aligns the optimal solution with the overall underlying models' performance. This approach mitigates the influence of poor-quality decisions, which can negatively impact others in the global setting. We define the decision weight function G as $GP_m, m = P_m * Acc_m$, where Acc_m represents the accuracy of the corresponding model, measured in isolation. To mimic the real-world settings where test labels are unavailable during inference, we utilize the models' accuracies on a probe/dev set.

6.3.2 Empirical Study

We assess the impact of integrating proposed factors into the ILP formulation on structured prediction tasks. Our approach is particularly suited for hierarchical structures encompassing multiple classes at different granularity levels, such as classical hierarchical classification problems. Additionally, we are the first to investigate the influence of enforcing global consistency on the procedural reasoning task, a complex real-world problem. To implement our method, we rely on the DomiKnowS framework [130], offering a versatile platform that enables implementing and evaluating techniques to leverage external logical knowledge with minimal effort on structured output prediction tasks.

6.3.2.1 Metrics and Evaluation

We compare our method against two inference-time approaches: sequential decoding and basic ILP (ILP without our refinement). In contrast to ILP, sequential decoding, which relies on expert-designed rules or programs to enforce consistency, is unique to each dataset. In addition to conventional metrics (e.g., accuracy/F1), we include measurements that evaluate changes applied by the inference techniques: (1) total changes (**C**), (2) the percentage of incorrect-to-correct changes (**+C**), (3) the percentage of correct-to-incorrect changes (**-C**). We further evaluate all the baselines and inference methods on (1) the percentage of decisions satisfying task constraints and (2) Set Correctness, the percentage of correct sets of interrelated decisions (i.e., predictions of all levels in the hierarchy must be correct for an image).

Number of Changes This metric quantifies the post-inference changes in decisions, specifically assessing the extent to which original decisions are altered due to inference constraints. It serves as a crucial indicator of whether the optimization method treats all decisions equally or prefers certain decisions over others. A genuinely global optimization method will result in multiple decision changes, promoting a more balanced distribution of alterations across all decisions. In contrast, expert-written strategies tend to favor specific decisions. This metric is straightforward to calculate by comparing the differences between decisions before and after applying the inference mechanism.

Ratio of In-Correct to Correct Changes (+C) This metric reveals the proportion of post-inference changes that are deemed favorable. While this metric may not carry substantial standalone significance, it is a valuable means to compare different inference techniques. A higher ratio signifies that the inference method has been more successful in deducing accurate labels based on the imposed constraints.

Ratio of Correct to In-Correct Changes (-C) This number shows the extent of undesirable changes made after inference. A lower ratio means the inference method has done a better job of preventing errors while ensuring the output adheres to the constraints.

Satisfaction Rate This metric shows how well predictions align with constraints. We calculate it by generating constraint instances from related decisions and counting the satisfying cases against all possible instances. Inference techniques guarantee that modified decisions always adhere to the constraints, resulting in a satisfaction rate of 100%.

Correctly Predicated Sets of Interrelated Decisions This metric is crucial for assessing the practical usefulness of the output from inference techniques or the original network decisions in downstream applications. The primary objective of inference mechanisms is to boost the percentage of these fully satisfying cases compared to the model's original performance, all while ensuring that the decisions align with the task's constraints. For instance, in a hierarchical classification task, we consider one instance to be correct only when the decisions at all levels are simultaneously accurate.

6.3.2.2 Tasks

Procedural Reasoning

Task: Procedural reasoning tasks entail tracking entities within a narrative. Following [48], we formulate this task as Question-Answering (QA). Two key questions are addressed for each entity e and step i: (1) *Where is e located in step i?* and (2) *What action is performed on e at step i?*. The decision output of this task exhibits heterogeneity, encompassing a diverse range of possible actions (limited multi-class) and varied locations derived from contextual information (spans). The task constraints establish relationships between action and location decisions as well as among action decisions at different steps. For instance, the ‘Destroy, Move’ sequence represents an invalid assignment for action predictions at steps i and $i \;\; 1$.

Dataset: We utilize the **Propara** dataset [33], a small dataset focusing on natural events. This dataset provides annotations for involved entities and their corresponding location changes. The label set is further expanded to include information on actions, which can be inferred from the sequence of locations.

Baseline: We employ a modified version of the MeeT architecture [156] as our baseline for this task. The MeeT model is designed to ask the two aforementioned questions at each step and employs a generative model (T5-large) to answer those questions. The **Sequential Decoding** baseline resolves action inconsistencies in a sequential stepwise manner (first to last), followed by the selection of locations accordingly.

Propara Dataset: The Propara dataset serves as a procedural reasoning benchmark, primarily devised to assess the ability of models to track significant entities across a series of events effectively. The stories within this dataset revolve around natural phenomena, such as photosynthesis. The annotation process involves capturing crucial entities and their corresponding locations at each step of the process, which are obtained through crowd-sourcing efforts. Figure 6.1 depicts an illustrative example of this dataset.

The sequence of locations pertaining to each entity can be further extended to infer the actions or status of the entity at each step. Previous studies [34] have proposed six possible actions for each entity at each step, namely ’Create,’ ’Move,’ ’Exist,’ ’Destroy,’ ’Prior,’ and ’Post.’ In this context, ’Prior’ signifies an entity that has not yet been created, while ’Post’ denotes an entity that has already been destroyed.

As for the baseline, we employ a modified version of the MeeT [156] architecture. The architecture utilizes T5-Large [129] as the backbone and employs a Question-Answering framework to extract the location and action of each entity at each step. The format of the input to the model is as follows for entity e and step i: "Where is e located in sent i? Sent 1: ..., Sent 2: ..., ...". For extracting the action, the set of options is also passed as input, resulting in the modification of the question to "What is the status of entity e in sent i? (a) Create (b) Move (c) Destroy (d) Exist (e) Prior (f) Post".

Although the original model of MeeT incorporates a Conditional Random Field (CRF) [80] layer during inference to ensure consistency among action decisions, we exclude this layer from our baseline. This decision is motivated by two reasons. Firstly, the use of CRF in this context is not generalizable as it relies on training data statistics for defining transitional scores. Secondly, we intend to impose consistency using various inference mechanisms and consider a joint framework to ensure both locations and actions exhibit consistency. Additionally, while the MeeT baseline employs two independent T5-Large models for each question type (location and action), our baseline utilizes the same model for both question types. For the sequential decoding technique to enforce sequential consistency among the series of interrelated action and location decisions, we utilize the post-processing code presented in [50].

Hierarchical Classification

Task: This task involves classifying inputs into various categories at distinct levels of granularity, establishing parent-child relationships between the classes where those follow a hierarchical structure.

Datasets: We employ three different datasets. (1) A subset of the Flickr dataset [188] with two hierarchical levels for the classification of images with types of *Animal, Flower, and Food*, (2) 20News dataset for text classification, where the label set is divided into two levels, and (3) The OK-VQA benchmark [104], a subset of the COCO dataset [92]. OK-VQA establishes the hierarchical relations between labels into four levels based on ConceptNet triplets and the dataset's knowledge base.

Baselines: ResNet [63] and BERT [44] are used to obtain representations for the image and text modalities, respectively. Linear classification layers are applied to convert obtained representations into decisions. The **Sequential Decoding** is top-down, bottom-up, and a two-stage (1) top-down on ‘None’ values and (2) bottom-up on labels for Animal/Flower/Food, 20 News, and VQA tasks, respectively.

Animal/Flower/Food Dataset: The dataset[1] employed in this study is sourced from the online platform ’Flickr’ and encompasses a total of 5439 images classified into three primary categories, namely ’Flower,’ ’Animal,’ and ’Food.’ Without an officially designated test set, a random partitioning strategy is adopted to ensure comparability in the distribution of training and testing instances. Consequently, the resulting splits are utilized within the experimental framework. The training subset encompasses 4531 images, while the test set comprises 1088 images. The dataset further comprises various sub-categories, including ’cat,’ ’dog,’ ’monkey,’ ’squirrel,’ ’daisy,’ ’dandelion,’ ’rose,’ ’sunflower,’ ’tulip,’ ’donuts,’ ’lasagna,’ ’pancakes,’ ’pizza,’ ’risotto,’ and ’salad.’ It should be noted that the data distribution across labels is not balanced, posing a more challenging classification task. This dataset is employed as a simplified scenario to illustrate the benefits of the proposed inference approach.

As the baseline for this task, we use ResNet-50 to represent the images and add a single layer MLP on top for each level. The model is further trained by Cross-Entropy objective and AdamW as optimizer.

The sequential decoding strategy for this dataset propagates labels in a top-down manner, where the most likely children of the selected Level1 decisions are chosen as the prediction at Level2.

20News Dataset: This dataset comprises a collection of diverse news articles classified into 23 distinct categories. In order to capture the hierarchical structure inherent in the dataset’s labels, we partition these categories into two levels. It should be noted that certain higher-level concepts lack corresponding lower-level labels, necessitating the inclusion of a ’None’ label at level 2. Furthermore, we perform a removal process on the initially annotated data containing the ’None’ labels, as this subset primarily consists of noisy documents that do not align with any categories present within the dataset. It is crucial to differentiate this removal process from the intentional addition of the ’None’ label at level 2, which we manually introduced.

[1] https://github.com/kaustubh77/Multi-Class-Classification

As the baseline for this task, we initially employed the Bert-Base encoder to generate representations for each news story. Due to the limited context size of Bert, which is constrained to a maximum of 512 tokens, we truncate the news articles accordingly and utilize the CLS token as the representative embedding for the entire article. For Level 1, a 2-layer Multilayer Perceptron (MLP) architecture is employed, with LeakyReLU as the chosen activation function. Additionally, Level 2 decisions are made using a single-layer MLP. The model is optimized using the AdamW optimizer during the training process, with the Cross-Entropy loss function employed.

The sequential decoding strategy in this dataset is a bottom-up strategy. Here, the model's decision from Level 2 is propagated into Level 1 without looking further into the initial probabilities generated by the model at that level.

OK-VQA (COCO) Dataset: The OK-VQA dataset is primarily introduced as a means to propose an innovative task centered around question-answering utilizing external knowledge. A subset of the COCO dataset is employed to construct this dataset, with augmented annotations obtained through crowdsourcing. While the main objective of the dataset revolves around question answering, it is important to note that it encompasses two levels of annotation. These annotations indicate the answer to the given question and provide additional clarifications regarding the types of objects depicted in the corresponding images. To leverage knowledge pertaining to image type relationships, the label set is expanded to include supplementary high-level concepts. A knowledge base is also provided, delineating parent-child relationships between these labels. The dataset comprises a total of 500 object labels. To enhance the breadth of knowledge encompassed by the dataset, we incorporate additional information from ConceptNet to establish comprehensive relationships among the labels. Notably, both the new information and the original knowledge base may contain noisy information. In conjunction with the original knowledge base, this forms a four-level hierarchical dependency among the initial 500 labels. Consequently, certain labels within each level may not possess corresponding children at lower levels, necessitating the introduction of 'None' labels at levels 2, 3, and 4.

In this study, we employ the Faster R-CNN framework [133] along with ResNet-110 as the

Model	Level 1 (3)				Level 2 (15)				Average
	Acc	C	+ C	- C	Acc	C	+ C	- C	Acc
Baseline	86.12	-	-	-	**54.85**	-	-	-	70.48
Sequential	86.12	-	-	-	54.39	32	**15.625**	37.5	70.25
ILP	86.07	16	43.75	43.75	54.43	16	12.5	37.5	70.25
+ Acc	86.14	3	33.33	**33.33**	54.41	29	13.79	37.93	70.27
+ Prior	86.30	24	50	41.67	54.78	8	12.5	**25**	70.54
+ Ent + Acc	86.09	12	33.33	50	54.41	20	10	40	70.25
+ Ent + Prior	**86.42**	25	**52**	40	**54.82**	7	14.29	28.57	**70.62**
+ All	86.17	16	43.75	43.75	54.50	16	12.5	37.5	70.33

Table 6.4 Results on *Animal/Flower/Food* dataset on four random seeds. Reported values are the average scores of runs with close variances for all techniques (Level1: ±1.6 and Level2: ±0.5). **C** values are derived from the best run. n in **Level** (n) denotes the number of output space classes. **Prior:** Prior Probability, and **Ent:** Entropy.

chosen methodology to represent individual objects within images. Subsequently, a one-layer Multilayer Perceptron (MLP) architecture is utilized to classify the images at each level of the hierarchical structure. It should be noted that the number of positive examples (i.e., labels that are not denoted as 'None') decreases as we move toward lower levels of the hierarchy. We perform subsampling on the 'None' labels for the corresponding classifiers at those levels to address this. The models are trained with the Cross-Entropy loss function and the AdamW optimizer.

The sequential decoding strategy for this dataset is a two-stage top-down and then bottom-up process. Here, 'None' labels are first propagated from Level 1 to Level 4, then the selected label (if not None) from Level 4 is propagated bottom-up to Level 1. Since each label at level$n$ only has one parent in Level$n-1$, this process does not need to look into the original model probabilities for propagation.

6.3.2.3 Results

Tables 6.4, 6.5, 6.6 and 6.7 display results for *Animal/Flower/Food*, *Ok-VQA*, *20 News*, and *Propara* datasets. 6.3.2.2. For close results, we use multiple seeds to validate reliability. Across experiments, the basic ILP technique favors decisions in smaller output spaces due to higher probability magnitudes (e.g., more changes in Actions than Locations in Table 6.7). Our new proposed variations can effectively mitigate this problem and perform a more balanced optimization.

Animal/Flower/Food: The sequential decoding establishes that the enforcement of the decisions

Model	Level 1 (274)	Level 2 (158)	Level 3 (63)	Level 4 (8)	Average
Baseline	56.73	54.45	43.43	17.68	**54.64**
Sequential	55.81	53.17	43.44	24.18	53.72
ILP	52.38	46.33	**49.66**	**28.43**	50.17
+ Acc	55.65	**54.67**	48.15	23.73	54.23
+ Prior	56.35	53.36	48.11	23.86	54.54
+ Ent + Acc	56.43	53.25	48.1	24.02	54.56
+ Ent + Prior	56.79	52.93	47.53	23.75	**54.61**
+ All	**56.84**	52.66	46.98	22.63	54.5

Table 6.5 The results on the Ok-VQA dataset. The values represent the F1 measure. Levels 2, 3, and 4 contain 'None' labels. The low F1 measure of lower levels is due to a huge number of False Positives.

Model	Level 1 (16)				Level 2 (8)			Average
	F1	C	+ C	- C	F1	C	+ C	F1
Baseline	73.62	-	-	-	75.13	-	-	74.01
Sequential	72.99	330	20.6	46.36	75.13	0	0.00	73.55
ILP	73.53	225	25.78	39.55	75.46	68	63.24	74.03
+ Acc	73.57	212	**26.89**	39.62	75.45	73	64.39	74.05
+ Prior	73.35	161	25.46	39.13	75.35	94	65.96	74.01
+ Ent + Acc	73.54	205	26.34	40	75.39	75	**64**	74.02
+ Ent + Prior	73.63	125	26.4	36	75.49	112	68.75	74.12
+ All	**73.64**	131	25.95	**35.11**	**75.52**	111	68.47	**74.13**

Table 6.6 The results on the 20News dataset, comparing various inference settings and the initial performance. Here, the *-C* of level 2 is 0 in all cases.

Model	Actions (6)				Locations (*)				Average
	Acc	C	+ C	- C	Acc	C	+ C	- C	Acc
Baseline	**73.05**	-	-	-	68.21	-	-	-	**70.47**
Sequential	71.56	75	13.33	46.66	67.63	255	27.8	32.2	69.47
ILP	**73**	63	**36.5**	38.1	66.38	217	19.8	35.9	69.47
+ Acc	**73**	63	**36.5**	38.1	66.43	217	19.8	35.9	69.50
+ Prior	72.88	119	31.93	**34.45**	67.54	138	23.2	32.6	**70.03**
+ Ent + Acc	72.93	63	34.92	38.1	66.38	219	19.6	35.6	69.44
+ Ent + Prior	71.62	209	25.83	37.32	68.16	53	26.4	28.3	69.78
+ All	71.74	198	25.75	36.86	**68.27**	72	**29.2**	**27.8**	69.89

Table 6.7 Results on Propara dataset. The dataset comprises 1910 location decisions and 1674 action decisions. *The output size of location decisions depends on the context of each procedure.

originating from a model with better accuracy and with a smaller output size (Level 1) on other decisions may even hurt them (Level 2). In such scenarios, the inclusion of *Expected Accuracy* favors dominant decisions and adversely affects performance. However, the inclusion of *Prior Probability* effectively achieves a balanced comparison among decisions. In this task, despite the basic ILP formulation being detrimental, some of the new variations can even surpass the original baseline performance.

Ok-VQA: The baseline exhibits lower accuracy in lower-level decisions with smaller output sizes. When applying the basic ILP method under these circumstances, a significant decline in results is observed, even below that of sequential decoding. However, incorporating any of our proposed factors leads to substantial improvements compared to the basic ILP formulation (over 4% improvement) and can surpass the performance of sequential decoding. Particularly, combining *Entropy* and *Prior Probability* achieves the best performance. Notably, although the baseline model has higher overall performance, its inconsistent outputs are unreliable for determining the object label (see Table 6.8).

20News: The baseline performance is similar across different decisions. Thus, in isolation, considering either the *Expected Accuracy* or the *Prior Probability* does not substantially impact the global optimization process. However, including all proposed factors *(Entropy, Accuracy, and Prior Probability)* leads to a balanced and optimal solution. Although the overall task performance in this experiment does not show significant improvements, this is mainly because the initial decision inconsistencies are minimal. Nevertheless, evaluating the positive and negative changes provides valuable insights into the significance of incorporating the proposed factors.

Propara: This is an example of a real-world task that involves hundreds of constraints and thousands of variables when combining decisions across entities and steps. Once again, basic ILP and *Expected Accuracy* factor prioritize decisions from the smaller output size (Actions). However, the *Prior probability* factor enables a more comparable space for resolving inconsistencies. Notably, the higher baseline performance is attributed to inconsistencies and cannot be used when reasoning about the process (See Table 6.8).

Dataset	Model	Satisfaction	Set Correctness
Animal/Flower	Baseline	96.4	53.40
	Sequential	100	**54.50**
	Ent + Prior	100	**54.50**
VQA	Baseline	38.99	54.43
	Sequential	100	57.11
	Ent + Prior	100	**58.92**
Propara	Baseline	45.12	23.30
	Sequential	100	28.81
	Prior	100	**30.93**

Table 6.8 Results of our proposed technique, baselines, and expert-written decoding strategies in terms of constraint satisfaction and set correctness. The **Set Correctness** metric reflects the practical usability of sets of dependent decisions in downstream applications.

Constraints: Table 6.8 presents the satisfaction results and set correctness metrics across various datasets. It is evident that our newly proposed method significantly outperforms the baseline in both of these metrics. Notably, the degree of improvement in set correctness is more pronounced when the initial consistency of the baseline is lower. This observation underscores the substantial significance of our proposed technique in ensuring the practical utility of model decisions in downstream applications by substantially increasing the proportion of correct interrelated decision sets. Furthermore, compared to sequential decoding, our proposed solutions demonstrate even greater performance enhancements, particularly in scenarios where the task complexity is higher, and global inference can exert its maximum effectiveness.

6.3.3 Discussion

Here, we address some of the key questions about this work.

Q1: Which metric is most important among the ones evaluated in this paper? All the metrics assessed in this paper provide insights into the model's performance. Among these, the **Set Correctness** score offers a comprehensive evaluation that combines constraint satisfaction and correctness, indicating the proportion of output decisions suitable for safe use in downstream tasks.

When comparing different ILP variations, the primary focus should be on the original task performance since they all share the same high satisfaction score of 100%. Additionally, the **Change** metric helps reveal whether an ILP variation conducts truly global optimization or exhibits a bias towards specific prediction classes.

In comparing the baseline method with inference techniques, it is essential to consider both the **satisfaction** and **set correctness** scores. This is because the raw model predictions, as initially generated, may not be directly acceptable. For instance, if a model predicts a "Move" action for entity A at step 4, but the location prediction does not indicate a change in location, it becomes unclear whether entity A changed locations.

Q2: Why utilize the model's overall accuracy in the score function instead of its accuracy for a specific decision variable? In our context, we assume that each decision type corresponds to a specific model. Therefore, assessing the model's accuracy is the same as evaluating the accuracy of a particular decision type. If a single model supplies multiple decision types, we can easily expand this concept to evaluate the accuracy of each decision type individually within the same framework.

Q3: What is the main difference between the sequential decoding strategy and the ILP formulation? The sequential decoding strategy is a domain-specific, expert-crafted technique employed for addressing decision inconsistencies in accordance with task constraints. In contrast, the ILP (Integer Linear Programming) formulation offers a more general, non-customized approach that isn't tailored to individual tasks.

Sequential decoding strategies typically involve rules or programs that often prefer a specific decision while adjusting other decisions to align with it. This approach prioritizes decision alignment over considering the probabilities associated with these decisions. On the other hand, the ILP optimization process seeks the most optimized solution by considering the raw probabilities from the models and the imposed constraints.

6.4 Conclusion

This chapter introduced an expanded benchmark aimed at assessing a model's ability to maintain consistency while reasoning through procedural texts in the context of entity tracking. Our enhanced benchmark encompasses a set of interrelated questions, all of which necessitate a fundamental comprehension of the described process to deduce accurate answers. We appraised the state-of-the-art model's efficacy in this domain, utilizing our evaluation framework. Our observations revealed a notable inconsistency in the model's reasoning, particularly concerning initial queries related to actions and locations. Furthermore, our findings demonstrated that incorporating a set of consistency-focused questions into the model's training regimen—despite these questions being derived from the same original dataset—improves its performance when addressing the initial question types. Moreover, we developed and assessed various strategies to enforce logical consistency within the model's outputs. The outcomes from these experiments suggest that adopting a global optimization approach for inference tends to be more advantageous than implementing sequential logical corrections, particularly when applying a comprehensive suite of constraints to the decision-making process.

We further proposed an approach for considering the uncertainty and confidence measures, including the decisions' prior probability, entropy, and expected accuracy, alongside raw probabilities when making globally consistent decisions based on diverse models. We demonstrated the effectiveness of incorporating our idea within the ILP formulation through experiments on four datasets including the procedural reasoning task. This contribution represents a significant advancement in integrating large models in a unified decision-making framework for conducting complex tasks requiring interrelated decisions.

CHAPTER 7

CONCLUSION & FUTURE WORK

This chapter will summarize our contributions and conclusions regarding the two directions that we took in this research: exploiting semantic structures in procedural reasoning and integrating logical constraints into deep neural models. Moreover, we will outline a series of ideas and directions for extending our current techniques and addressing the remaining challenges in procedural reasoning.

7.1 Summary of Contributions

Reasoning over procedural text presents a significant challenge for machine learning models due to the dynamic nature of the described world. Despite recent advances and promising performances achieved by building on Pretrained Language Models (PLMs), a notable performance gap between neural models and human capabilities persists. In this thesis, we introduced various strategies to improve language models' proficiency in complex reasoning over procedural contexts, particularly focusing on entity tracking and procedural summarization tasks. We also proposed a new framework and benchmark to advance research on integrating domain knowledge with deep neural networks.

7.1.1 Entity Tracking

Our first proposition aimed at enhancing understanding and reasoning over the temporal dynamics of events in procedural contexts. We introduced the Time-Stamped Language Model (TSLM), which augments PLMs with an additional embedding layer representing the temporal dimension of actions (past, present, and future). This enhancement allows PLMs to effectively tackle entity tracking by comprehensively reasoning across the process context and dynamically shifting focus between steps. This capability is crucial for neural models to utilize preceding and forthcoming contexts to resolve ambiguities and fill information gaps in procedural instructions.

Furthermore, recognizing the importance of understanding actions, their relationships with objects, and their impacts on entities for entity tracking, we proposed integrating semantic parsers with procedural reasoning modules. This approach allows neural models to directly reason about actions and entities, thereby abstracting and interpreting the significance of actions more accurately. We developed a symbolic model based on semantic parsers, achieving superior performance over several neural models. Additionally, we extended this approach by encoding semantic parses into graph representations for procedural reasoning modules. This result of such integration with various base neural models proved effective in improving the state-of-the-art models.

7.1.2 Procedural Abstraction

We introduced a novel method that trains a similarity matrix to correlate critical actions in the summary with procedural instructions. A set of constraints was designed to ensure the summary's sequential integrity accurately reflects the process flow. This innovative approach and modeling effort yielded state-of-the-art results on the RecipeQA benchmark for textual cloze tasks involving multi-modal recipes. We also demonstrated the advantages of utilizing text and image modalities in summarization, leveraging entangled representations from both to enhance task performance.

7.1.3 Integration of Logical Constraints with Deep Neural Networks

We developed DomiKnowS, a novel declarative framework for incorporating domain knowledge into deep neural networks. DomiKnowS offers a modular interface for defining knowledge and computational units and facilitates various methods for integrating human knowledge with data-derived learning.

Utilizing DomiKnowS's flexibility, we established a new benchmark (GLUECons) for evaluating the effectiveness of integrating constraints with deep neural networks. GLUECons comprises nine benchmarks across five task categories, expanding the evaluative scope to assess the general applicability of constraint integration methods across diverse scenarios rather than focusing solely on task-specific performance.

7.1.4 Procedural Reasoning Considering Logical Constraints

We pioneered applying a global constraint integration technique to complex tasks such as procedural reasoning. Given the extensive constraint space associated with procedural reasoning, we enriched the dataset by creating consistency sets (various relevant question/task types) to probe the consistency of neural model reasoning and explore the potential of further constraints to enhance model performance.

Acknowledging the diverse decision-making requirements inherent in procedural reasoning, we proposed an enhancement to integer linear programming (ILP) to ensure decision consistency across heterogeneous outputs. This novel approach accounts for additional factors such as confidence, prior probability expectations, and decision accuracy, demonstrating improved performance across multiple tasks, including procedural reasoning.

7.2 Future Directions

This section proposes several avenues for addressing the remaining challenges in procedural reasoning and integrating constraints with neural models.

Constraints and Generative AI Our research explored strategies for integrating constraints with computational units (neural models) by treating them as black-box entities and utilizing model-generated probabilities but was mostly limited to the scope of structured prediction tasks with the assumption that changing the probability of one decision does not directly affect the raw probabilities generated by the model on other decisions. However, the generative AI paradigm, especially auto-regressive techniques, produces dependent probabilities for each output variable, complicating ad-hoc inference adjustments during runtime. This scenario necessitates techniques for approximating or considering the effects of such adjustments during the inference phase.

Model Consistency in Complex Reasoning Tasks We highlighted the inconsistency of language models in reasoning over interconnected decisions in procedural reasoning. While inference techniques can address inconsistencies, simply training models on interrelated decisions does not inherently enhance consistency. Improving model consistency during training without reliance on inference techniques remains an open challenge. Integrating a soft interpretation of logical constraint violations as loss objectives may enhance consistent model performance.

Interpretability in Procedural Reasoning The detailed annotation of tasks like entity tracking provides valuable step-by-step insights into entity evolution during processes. Yet, the underlying reasoning steps for each decision are often not explicitly documented in current datasets. Extending datasets to include these reasoning steps could enable models to perform more comprehensive reasoning, from textual analysis to abstract procedural understanding.

Utilizing Entity Tracking in Story Understanding In this research, we have investigated the underlying models for entity tracking. An interesting future direction is to consider the effect of such underlying knowledge in story completion and understanding, where entity tracking plays a vital role.

BIBLIOGRAPHY

[1] Jordi Adell, Antonio Bonafonte, Antonio Cardenal, Marta R. Costa-Jussà, José A. R. Fonollosa, Asunción Moreno, Eva Navas, and Eduardo R. Banga. BUCEADOR, a multi-language search engine for digital libraries. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 1705–1709, Istanbul, Turkey, May 2012. European Language Resources Association (ELRA).

[2] Kareem Ahmed, Tao Li, Thy Ton, Quan Guo, Kai-Wei Chang, Parisa Kordjamshidi, Vivek Srikumar, Guy Van den Broeck, and Sameer Singh. Pylon: A pytorch framework for learning with constraints. In *NeurIPS 2021 Competitions and Demonstrations Track*, pages 319–324. PMLR, 2022.

[3] Alan Akbik, Duncan Blythe, and Roland Vollgraf. Contextual string embeddings for sequence labeling. In *COLING 2018, 27th International Conference on Computational Linguistics*, pages 1638–1649, 2018.

[4] James F Allen and Choh Man Teng. Broad coverage, domain-generic deep semantic parsing. In *2017 AAAI Spring Symposium Series*, 2017.

[5] Mustafa Sercan Amac, Semih Yagcioglu, Aykut Erdem, and Erkut Erdem. Procedural reasoning networks for understanding multimodal procedures. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 441–451, 2019.

[6] Aida Amini, Antoine Bosselut, Bhavana Dalvi Mishra, Yejin Choi, and Hannaneh Hajishirzi. Procedural reading comprehension with attribute-aware context flow. In *Proceedings of the Conference on Automated Knowledge Base Construction (AKBC)*, 2020.

[7] Saeed Amizadeh, Hamid Palangi, Alex Polozov, Yichen Huang, and Kazuhito Koishida. Neuro-symbolic visual reasoning: Disentangling. In *ICML*, pages 279–290. PMLR, 2020.

[8] Peter Anderson, Basura Fernando, Mark Johnson, and Stephen Gould. Guided open vocabulary image captioning with constrained beam search. In *EMNLP*, pages 936–945, 2017.

[9] Akari Asai and Hannaneh Hajishirzi. Logic-guided data augmentation and regularization for consistent question answering. *arXiv preprint arXiv:2004.10157*, 2020.

[10] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[11] Stephen H. Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. Hinge-loss markov random fields and probabilistic soft logic. *Journal of Machine Learning Research (JMLR)*, 18:1–67, 2017.

[12] Stephen H. Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. Hinge-loss markov random fields and probabilistic soft logic. *JMLR*, 18:1–67, 2017.

[13] Jonathan Berant, Vivek Srikumar, Pei-Chun Chen, Abby Vander Linden, Brittany Harding, Brad Huang, Peter Clark, and Christopher D Manning. Modeling biological processes for reading comprehension. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1499–1510, 2014.

[14] Marcus D Bloice, Peter M Roth, and Andreas Holzinger. Performing arithmetic using a neural network trained on digit permutation pairs. In *ISMIS*, pages 255–264. Springer, 2020.

[15] Sebastian Borgeaud and Guy Emerson. Leveraging sentence similarity in natural language generation: Improving beam search using range voting. In *4th WNGT*, pages 97–109, 2020.

[16] Antoine Bosselut, Omer Levy, Ari Holtzman, Corin Ennis, Dieter Fox, and Yejin Choi. Simulating action dynamics with neural process networks. In *Proceedings of the 6th International Conference for Learning Representations (ICLR)*, 2018.

[17] Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In *EMNLP*, pages 632–642, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[18] Matthias Broecheler, Lilyana Mihalkova, and Lise Getoor. Probabilistic similarity logic. In *Conference on Uncertainty in Artificial Intelligence*, 2010.

[19] Miles Brundage, Shahar Avin, Jasmine Wang, Haydn Belfield, Gretchen Krueger, Gillian Hadfield, Heidy Khlaaf, Jingying Yang, Helen Toner, Ruth Fong, et al. Toward trustworthy ai development: mechanisms for supporting verifiable claims. *arXiv preprint arXiv:2004.07213*, 2020.

[20] Judith Bütepage, Ali Ghadirzadeh, Özge Öztimur Karadağ, Mårten Björkman, and Danica Kragic. Imitating by generating: Deep generative models for imitation of interactive tasks. *Frontiers in Robotics and AI*, 7:47, 2020.

[21] Oana-Maria Camburu, Brendan Shillingford, Pasquale Minervini, Thomas Lukasiewicz, and Phil Blunsom. Make up your mind! adversarial generation of inconsistent natural language explanations. *arXiv preprint arXiv:1910.03065*, 2019.

[22] Bob Carpenter, Andrew Gelman, Matthew Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. Stan: A probabilistic programming language. *Journal of Statistical Software, Articles*, 76(1):1–32, 2017.

[23] Ming-Wei Chang, Lev Ratinov, and Dan Roth. Structured learning with constrained conditional models. *Machine learning*, 88(3):399–431, 2012.

[24] Angelica Chen, Jason Phang, Alicia Parrish, Vishakh Padmakumar, Chen Zhao, Samuel R Bowman, and Kyunghyun Cho. Two failures of self-consistency in the multi-step reasoning of llms. *arXiv preprint arXiv:2305.14279*, 2023.

[25] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar, October 2014. Association for Computational Linguistics.

[26] Peter Clark, Oyvind Tafjord, and Kyle Richardson. Transformers as soft reasoners over language. *arXiv preprint arXiv:2002.05867*, 2020.

[27] James Clarke, Dan Goldwasser, Ming-Wei Chang, and Dan Roth. Driving semantic parsing from the world's response. In *14th CoNLL*, pages 18–27, 2010.

[28] Anthony Costa Constantinou, Norman Fenton, and Martin Neil. Integrating expert knowledge with data in bayesian networks: Preserving data-driven expectations when the expert variables remain unobserved. *Expert systems with applications*, 56:197–208, 2016.

[29] Florinel-Alin Croitoru, Vlad Hondru, Radu Tudor Ionescu, and Mubarak Shah. Diffusion models in vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[30] Lei Cui, Shaohan Huang, Furu Wei, Chuanqi Tan, Chaoqun Duan, and Ming Zhou. SuperAgent: A customer service chatbot for E-commerce websites. In *Proceedings of ACL 2017, System Demonstrations*, pages 97–102, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[31] Daniel Dahlmeier and Hwee Tou Ng. A beam-search decoder for grammatical error correction. In *EMNLP 2012*, pages 568–578, 2012.

[32] Bhavana Dalvi, Lifu Huang, Niket Tandon, Wen-tau Yih, and Peter Clark. Tracking state changes in procedural text: a challenge dataset and models for process paragraph comprehension. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1595–1604, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[33] Bhavana Dalvi, Lifu Huang, Niket Tandon, Wen-tau Yih, and Peter Clark. Tracking state changes in procedural text: a challenge dataset and models for process paragraph comprehension. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1595–1604, 2018.

[34] Bhavana Dalvi, Niket Tandon, Antoine Bosselut, Wen-tau Yih, and Peter Clark. Everything

happens for a reason: Discovering the purpose of actions in procedural text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4496–4505, Hong Kong, China, November 2019. Association for Computational Linguistics.

[35] Rajarshi Das, Tsendsuren Munkhdalai, Xingdi Yuan, Adam Trischler, and Andrew McCallum. Building dynamic knowledge graphs from text using machine reading comprehension. In *International Conference on Learning Representations*, 2018.

[36] Tirtharaj Dash, Sharad Chitlangia, Aditya Ahuja, and Ashwin Srinivasan. A review of some techniques for inclusion of domain-knowledge into deep neural networks. *Scientific Reports*, 12(1):1–15, 2022.

[37] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: a probabilistic Prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 2468–2473. AAAI Press, 2007.

[38] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *IJCAI*, volume 7, pages 2462–2467. Hyderabad, 2007.

[39] Luc De Raedt, Robin Manhaeve, Sebastijan Dumancic, Thomas Demeester, and Angelika Kimmig. Neuro-symbolic= neural+ logical+ probabilistic. In *NeSy'19 workshop @ IJCAI*, 2019.

[40] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[41] Daniel Deutsch, Shyam Upadhyay, and Dan Roth. A general-purpose algorithm for constrained sequential inference. In *Proceedings of the 23rd CoNLL*, pages 482–492, 2019.

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

[43] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[44] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[45] Perdo Domingos and Matthew Richardson. Markov logic: A unifying framework for statistical relational learning. In *ICML'04 Workshop on Statistical Relational Learning and its Connections to Other Fields*, pages 49–54, 2004.

[46] Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Lucas Morales, Luke Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B Tenenbaum. Dreamcoder: Bootstrapping inductive program synthesis with wake-sleep library learning. In *Proceedings of the 42nd ACM SIGPLAN PLDI*, pages 835–850, 2021.

[47] Hossein Rajaby Faghihi, Quan Guo, Andrzej Uszok, Aliakbar Nafar, and Parisa Kordjamshidi. Domiknows: A library for integration of symbolic domain knowledge in deep learning. In *EMNLP: System Demonstrations*, pages 231–241, 2021.

[48] Hossein Rajaby Faghihi and Parisa Kordjamshidi. Time-stamped language model: Teaching language models to understand the flow of events. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4560–4570, 2021.

[49] Hossein Rajaby Faghihi and Parisa Kordjamshidi. Time-stamped language model: Teaching language models to understand the flow of events. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4560–4570, 2021.

[50] Hossein Rajaby Faghihi, Parisa Kordjamshidi, Choh Man Teng, and James Allen. The role of semantic parsing in understanding procedural text. *arXiv preprint arXiv:2302.06829*, 2023.

[51] Hossein Rajaby Faghihi, Aliakbar Nafar, Chen Zheng, Roshanak Mirzaee, Yue Zhang, Andrzej Uszok, Alexander Wan, Tanawan Premsri, Dan Roth, and Parisa Kordjamshidi. Gluecons: A generic benchmark for learning under constraints. *arXiv preprint arXiv:2302.10914*, 2023.

[52] Haoqi Fan and Jiatong Zhou. Stacked latent attention for multimodal reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1072–1080, 2018.

[53] George Ferguson, James F Allen, et al. Trips: An integrated intelligent problem-solving assistant. In *Aaai/Iaai*, pages 567–572, 1998.

[54] G.D. Forney. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

[55] Markus Freitag and Yaser Al-Onaizan. Beam search strategies for neural machine translation. *ACL 2017*, page 56, 2017.

[56] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages

1050–1059. PMLR, 2016.

[57] Matt Gardner, Yoav Artzi, Victoria Basmova, Jonathan Berant, Ben Bogin, Sihao Chen, Pradeep Dasigi, Dheeru Dua, Yanai Elazar, Ananth Gottumukkala, Nitish Gupta, Hannaneh Hajishirzi, Gabriel Ilharco, Daniel Khashabi, Kevin Lin, Jiangming Liu, Nelson F. Liu, Phoebe Mulcaire, Qiang Ning, Sameer Singh, Noah A. Smith, Sanjay Subramanian, Reut Tsarfaty, Eric Wallace, Ally Zhang, and Ben Zhou. Evaluating nlp models via contrast sets. *ArXiv*, abs/2004.02709, 2020.

[58] Yuling Gu, Bhavana Dalvi, and Peter Clark. Do language models have coherent mental models of everyday things? In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1892–1913, 2023.

[59] Quan Guo, Hossein Rajaby Faghihi, Yue Zhang, Andrzej Uszok, and Parisa Kordjamshidi. Inference-masked loss for deep structured output learning. In *29th IJCAI*, pages 2754–2761, 2021.

[60] Quan Guo, Hossein Rajaby Faghihi, Yue Zhang, Andrzej Uszok, and Parisa Kordjamshidi. Inference-masked loss for deep structured output learning. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 2754–2761. International Joint Conferences on Artificial Intelligence Organization, 7 2020. Main track.

[61] Aditya Gupta and Greg Durrett. Tracking discrete and continuous entity state for process understanding. In *Proceedings of the Third Workshop on Structured Prediction for NLP*, pages 7–12, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[62] James Hargreaves, Andreas Vlachos, and Guy Emerson. Incremental beam manipulation for natural language generation. In *16th EACL*, pages 2563–2574, 2021.

[63] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[64] Mikael Henaff, Jason Weston, Arthur Szlam, Antoine Bordes, and Yann LeCun. Tracking the world state with recurrent entity networks. In *5th International Conference on Learning Representations, ICLR 2017*, 2017.

[65] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Advances in neural information processing systems*, pages 1693–1701, 2015.

[66] Jack Hessel, Lillian Lee, and David Mimno. Unsupervised discovery of multimodal links in multi-image, multi-sentence documents. In *EMNLP*, 2019.

[67] Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, and Eric Xing. Harnessing deep neural networks with logic rules. In *54th ACL*, pages 2410–2420, 2016.

[68] Hao Huang, Xiubo Geng, Jian Pei, Guodong Long, and Daxin Jiang. Reasoning over entity-action-location graph for procedural text understanding. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5100–5109, Online, August 2021. Association for Computational Linguistics.

[69] Jiani Huang, Ziyang Li, Binghong Chen, Karan Samel, Mayur Naik, Le Song, and Xujie Si. Scallop: From probabilistic deductive databases to scalable differentiable reasoning. *Neurips*, 2021.

[70] Yifan Jiang, Filip Ilievski, and Kaixin Ma. Transferring procedural knowledge across commonsense tasks. *arXiv preprint arXiv:2304.13867*, 2023.

[71] Nora Kassner, Oyvind Tafjord, Hinrich Schütze, and Peter Clark. Beliefbank: Adding memory to a pre-trained language model for a systematic notion of belief. *arXiv preprint arXiv:2109.14723*, 2021.

[72] Phil Kim and Phil Kim. Convolutional neural network. *MATLAB deep learning: with machine learning, neural networks and artificial intelligence*, pages 121–147, 2017.

[73] P. Kordjamshidi, D. Roth, and H. Wu. Saul: Towards declarative learning based programming. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 7 2015.

[74] Parisa Kordjamshidi, Daniel Khashabi, Christos Christodoulopoulos, Bhargav Mangipudi, Sameer Singh, and Dan Roth. Better call saul: Flexible programming for learning and inference in nlp. In *Proc. of the International Conference on Computational Linguistics (COLING)*, 2016.

[75] Parisa Kordjamshidi, Daniel Khashabi, Christos Christodoulopoulos, Bhargav Mangipudi, Sameer Singh, and Dan Roth. Better call Saul: Flexible programming for learning and inference in NLP. In *COLING*, pages 3030–3040, Osaka, Japan, December 2016. COLING.

[76] Parisa Kordjamshidi, Dan Roth, and Hao Wu. Saul: Towards declarative learning based programming. In *2015 AAAI Fall Symposium Series*, 2015.

[77] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical Report 0, University of Toronto, Toronto, Ontario, 2009.

[78] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *NeurIPS*, 25, 2012.

[79] Igor Labutov, Shashank Srivastava, and Tom Mitchell. LIA: A natural language programmable personal assistant. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 145–150, Brussels, Belgium, November 2018. Association for Computational Linguistics.

[80] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[81] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[82] Celine Lee, Justin Gottschlich, and Dan Roth. Toward code generation: A survey and lessons from semantic parsing. *arXiv preprint arXiv:2105.03317*, 2021.

[83] Jay Yoon Lee, Sanket Vaibhav Mehta, Michael Wick, Jean-Baptiste Tristan, and Jaime Carbonell. Gradient-based inference for networks with output constraints. In *AAAI*, volume 33, pages 4147–4154, 2019.

[84] Jay Yoon Lee, Michael L Wick, Jean-Baptiste Tristan, and Jaime G Carbonell. Enforcing output constraints via sgd: A step towards neural lagrangian relaxation. In *AKBC@ NIPS*, 2017.

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

[86] Liunian Harold Li, Mark Yatskar, Da Yin, Cho-Jui Hsieh, and Kai-Wei Chang. Visualbert: A simple and performant baseline for vision and language. *arXiv preprint arXiv:1908.03557*, 2019.

[87] Rumeng Li, Xun Wang, and Hong Yu. Metamt, a meta learning method leveraging multiple domain data for low resource machine translation. In *AAAI*, volume 34, pages 8245–8252, 2020.

[88] Tao Li, Vivek Gupta, Maitrey Mehta, and Vivek Srikumar. A logic-driven framework for consistency of neural models. In *EMNLP-IJCNLP*, pages 3924–3935, 2019.

[89] Tao Li and Vivek Srikumar. Augmenting neural networks with first-order logic. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 292–302, 2019.

[90] Tao Li and Vivek Srikumar. Augmenting neural networks with first-order logic. In Anna Korhonen, David R. Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Conference*

of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers, pages 292–302. Association for Computational Linguistics, 2019.

[91] Xingxuan Li, Liying Cheng, Qingyu Tan, Hwee Tou Ng, Shafiq Joty, and Lidong Bing. Unlocking temporal question answering for large language models using code execution. *arXiv preprint arXiv:2305.15014*, 2023.

[92] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.

[93] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019.

[94] Xiaoxu Liu, Haoye Lu, and Amiya Nayak. A spam transformer model for sms spam detection. *IEEE Access*, 9:80253–80263, 2021.

[95] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.

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

[97] Reginald Long, Panupong Pasupat, and Percy Liang. Simpler context-dependent logical forms via model projections. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1456–1465, Berlin, Germany, August 2016. Association for Computational Linguistics.

[98] Ximing Lu, Peter West, Rowan Zellers, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Neurologic decoding:(un) supervised neural text generation with predicate logic constraints. In *NAACL*, pages 4288–4299, 2021.

[99] Kaixin Ma, Filip Ilievski, Jonathan Francis, Eric Nyberg, and Alessandro Oltramari. Coalescing global and local information for procedural text understanding. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 1534–1545, 2022.

[100] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[101] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. *NeurIPS*, 2018. Code&nbsp;is available.

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

[103] Vikash K. Mansinghka, Daniel Selsam, and Yura N. Perov. Venture: a higher-order probabilistic programming platform with programmable inference. *CoRR*, abs/1404.0099, 2014.

[104] Kenneth Marino, Mohammad Rastegari, Ali Farhadi, and Roozbeh Mottaghi. Ok-vqa: A visual question answering benchmark requiring external knowledge. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3195–3204, 2019.

[105] Lluís Màrquez, Xavier Carreras, Kenneth C Litkowski, and Suzanne Stevenson. Semantic role labeling: an introduction to the special issue, 2008.

[106] Sherin Mary Mathews. Explainable artificial intelligence applications in nlp, biomedical, and malware classification: a literature review. In *Intelligent computing:proceedings of the computing conference*, pages 1269–1292. Springer, 2019.

[107] Larry R Medsker and LC Jain. Recurrent neural networks. *Design and Applications*, 5:64–67, 2001.

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

[109] Brian Milch, Bhaskara Marthi, Stuart Russell, David Sontag, Daniel L. Ong, and Andrey Kolobov. BLOG: Probabilistic models with unknown objects. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2005.

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

[111] Pasquale Minervini and Sebastian Riedel. Adversarially regularising neural nli models to integrate logical background knowledge. *arXiv preprint arXiv:1808.08609*, 2018.

[112] Tom Minka, John M. Winn, John P. Guiver, and David A. Knowles. Infer.NET 2.5, 2012. Microsoft Research Cambridge. http://research.microsoft.com/infernet.

[113] Roshanak Mirzaee, Hossein Rajaby Faghihi, Qiang Ning, and Parisa Kordjamshidi. Spartqa: A textual question answering benchmark for spatial reasoning. In *NAACL*, pages 4582–4598,

2021.

[114] Roshanak Mirzaee and Parisa Kordjamshidi. Disentangling extraction and reasoning in multi-hop spatial reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3379–3397, 2023.

[115] Nikhil Muralidhar, Mohammad Raihanul Islam, Manish Marwah, Anuj Karpatne, and Naren Ramakrishnan. Incorporating prior domain knowledge into deep neural networks. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 36–45. IEEE, 2018.

[116] Hyeonseob Nam, Jung-Woo Ha, and Jeonghee Kim. Dual attention networks for multimodal reasoning and matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 299–307, 2017.

[117] Yatin Nandwani, Abhishek Pathak, Mausam, and Parag Singla. A primal dual formulation for deep learning with constraints. In *NeurIPS*, 2019.

[118] Yatin Nandwani, Abhishek Pathak, and Parag Singla. A primal dual formulation for deep learning with constraints. *NeurIPS*, 32, 2019.

[119] Usman Naseem, Imran Razzak, Katarzyna Musial, and Muhammad Imran. Transformer based deep intelligent contextual embedding for twitter sentiment analysis. *Future Generation Computer Systems*, 113:58–69, 2020.

[120] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 427–436, 2015.

[121] Qiang Ning, Zhili Feng, Hao Wu, and Dan Roth. Joint reasoning for temporal and causal relations. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2278–2288, 2018.

[122] Kuntal Kumar Pal, Kazuaki Kashihara, Pratyay Banerjee, Swaroop Mishra, Ruoyu Wang, and Chitta Baral. Constructing flow graphs from procedural cybersecurity texts. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3945–3957, Online, August 2021. Association for Computational Linguistics.

[123] Raghavendra Pappagari, Piotr Zelasko, Jesús Villalba, Yishay Carmiel, and Najim Dehak. Hierarchical transformers for long document classification. In *2019 IEEE automatic speech recognition and understanding workshop (ASRU)*, pages 838–844. IEEE, 2019.

[124] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[125] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[126] Avi Pfeffer. *Practical Probabilistic Programming*. Manning Publications, 2016.

[127] Vasin Punyakanok, Dan Roth, Wen-tau Yih, and Dav Zimak. Semantic role labeling via integer linear programming inference. In *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*, pages 1346–1352, 2004.

[128] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[129] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

[130] Hossein Rajaby Faghihi, Quan Guo, Andrzej Uszok, Aliakbar Nafar, and Parisa Kordjamshidi. DomiKnowS: A library for integration of symbolic domain knowledge in deep learning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 231–241, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

[131] Hossein Rajaby Faghihi, Roshanak Mirzaee, Sudarshan Paliwal, and Parisa Kordjamshidi. Latent alignment of procedural concepts in multimodal recipes. In *Proceedings of the First Workshop on Advances in Language and Vision Research*, pages 26–31, 2020.

[132] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics.

[133] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[134] Francesco Riccio, Roberto Capobianco, and Daniele Nardi. Guess: Generative modeling of unknown environments and spatial abstraction for robots. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1978–1980, 2020.

[135] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine learning*, 62(1):107–136, 2006.

[136] N. Rizzolo and D. Roth. Learning based Java for rapid development of NLP systems. In *Proceedings of the Seventh Conference on International Language Resources and Evaluation*, 2010.

[137] Nick Rizzolo and Dan Roth. Integer linear programming for coreference resolution. *Anaphora Resolution: Algorithms, Resources, and Applications*, pages 315–343, 2016.

[138] D. Roth and W. Yih. Integer linear programming inference for conditional random fields. In *ICML*, pages 737–744, 2005.

[139] Dan Roth and Vivek Srikumar. Integer linear programming formulations in natural language processing. In *15th EACL: Tutorial Abstracts*, 2017.

[140] Dan Roth and Wen-tau Yih. Integer linear programming inference for conditional random fields. In *Proceedings of the 22nd international conference on Machine learning*, pages 736–743, 2005.

[141] Alexander Rush. Torch-struct: Deep structured prediction library. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 335–342, Online, July 2020. Association for Computational Linguistics.

[142] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[143] Mohammed Saeed, Naser Ahmadi, Preslav Nakov, and Paolo Papotti. Rulebert: Teaching soft rules to pre-trained language models. In *EMNLP*, pages 1460–1476, 2021.

[144] Erik F Sang and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. *arXiv preprint cs/0306050*, 2003.

[145] Marvin C Santillan and Arnulfo P Azcarraga. Poem generation using transformers and doc2vec embeddings. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2020.

[146] Adam Santoro, Ryan Faulkner, David Raposo, Jack Rae, Mike Chrzanowski, Theophane Weber, Daan Wierstra, Oriol Vinyals, Razvan Pascanu, and Timothy Lillicrap. Relational recurrent neural networks. In *Advances in neural information processing systems*, pages 7299–7310, 2018.

[147] Taisuke Sato and Yoshitaka Kameya. Prism: A language for symbolic-statistical modeling. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages

1330–1339, 1997.

[148] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. Picard: Parsing incrementally for constrained auto-regressive decoding from language models. In *EMNLP*, pages 9895–9901, 2021.

[149] Moritz Schubotz, Philipp Scharpf, Kaushal Dudhat, Yash Nagar, Felix Hamborg, and Bela Gipp. Introducing mathqa: a math-aware question answering system. *IDD*, 2018.

[150] Karin Kipper Schuler. Verbnet: A broad-coverage, comprehensive verb lexicon. 2005.

[151] Min Joon Seo, Sewon Min, Ali Farhadi, and Hannaneh Hajishirzi. Query-reduction networks for question answering. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[152] Peng Shi and Jimmy Lin. Simple bert models for relation extraction and semantic role labeling. *arXiv preprint arXiv:1904.05255*, 2019.

[153] Qi Shi, Qian Liu, Bei Chen, Yu Zhang, Ting Liu, and Jian-Guang Lou. Lemon: Language-based environment manipulation via execution-guided pre-training. *arXiv preprint arXiv:2201.08081*, 2022.

[154] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. *arXiv preprint arXiv:2009.03509*, 2020.

[155] Manli Shu, Jiongxiao Wang, Chen Zhu, Jonas Geiping, Chaowei Xiao, and Tom Goldstein. On the exploitability of instruction tuning. *Advances in Neural Information Processing Systems*, 36, 2024.

[156] Janvijay Singh, Fan Bai, and Zhen Wang. Entity tracking via effective use of multi-task learning model and mention-guided decoding. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 1255–1263, Dubrovnik, Croatia, May 2023. Association for Computational Linguistics.

[157] Shashi Kant Singh, Shubham Kumar, and Pawan Singh Mehra. Chat gpt & google bard ai: A review. In *2023 International Conference on IoT, Communication and Automation Technology (ICICAT)*, pages 1–6. IEEE, 2023.

[158] Robyn Speer, Joshua Chin, and Catherine Havasi. Conceptnet 5.5: An open multilingual graph of general knowledge. *CoRR*, abs/1612.03975, 2016.

[159] Russell Stewart and Stefano Ermon. Label-free supervision of neural networks with physics and domain knowledge. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[160] Shane Storks, Qiaozi Gao, Yichi Zhang, and Joyce Chai. Tiered reasoning for intuitive physics: Toward verifiable commonsense language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4902–4918, 2021.

[161] Weijie Su, Xizhou Zhu, Yue Cao, Bin Li, Lewei Lu, Furu Wei, and Jifeng Dai. Vl-bert: Pre-training of generic visual-linguistic representations. In *Eighth International Conference on Learning Representations (ICLR)*, April 2020.

[162] Sainbayar Sukhbaatar, arthur szlam, Jason Weston, and Rob Fergus. End-to-end memory networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28, pages 2440–2448. Curran Associates, Inc., 2015.

[163] Haitian Sun, Bhuwan Dhingra, Manzil Zaheer, Kathryn Mazaitis, Ruslan Salakhutdinov, and William Cohen. Open domain question answering using early fusion of knowledge bases and text. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4231–4242, 2018.

[164] Jiankai Sun, De-An Huang, Bo Lu, Yun-Hui Liu, Bolei Zhou, and Animesh Garg. Plate: Visually-grounded planning with transformers in procedural tasks. *IEEE Robotics and Automation Letters*, 7(2):4924–4930, 2022.

[165] Charles Sutton and Andrew McCallum. An introduction to conditional random fields. *Foundations and Trends in Machine Learning*, 4(4):267–373, 2012.

[166] Hao Tan and Mohit Bansal. LXMERT: Learning cross-modality encoder representations from transformers. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5100–5111, Hong Kong, China, November 2019. Association for Computational Linguistics.

[167] Niket Tandon, Bhavana Dalvi, Joel Grus, Wen-tau Yih, Antoine Bosselut, and Peter Clark. Reasoning about actions and state changes by injecting commonsense knowledge. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 57–66, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[168] Niket Tandon, Bhavana Dalvi, Keisuke Sakaguchi, Peter Clark, and Antoine Bosselut. WIQA: A dataset for "what if..." reasoning over procedural text. In *EMNLP*, 2019.

[169] Niket Tandon, Bhavana Dalvi, Keisuke Sakaguchi, Peter Clark, and Antoine Bosselut. Wiqa: A dataset for "what if..." reasoning over procedural text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6076–6085, 2019.

[170] Niket Tandon, Keisuke Sakaguchi, Bhavana Dalvi, Dheeraj Rajagopal, Peter Clark, Michal Guerquin, Kyle Richardson, and Eduard Hovy. A dataset for tracking entities in open domain procedural text. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6408–6417, Online, November 2020. Association for Computational Linguistics.

[171] Niket Tandon, Keisuke Sakaguchi, Bhavana Dalvi, Dheeraj Rajagopal, Peter Clark, Michal Guerquin, Kyle Richardson, and Eduard Hovy. A dataset for tracking entities in open domain procedural text. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6408–6417, 2020.

[172] Makarand Tapaswi, Yukun Zhu, Rainer Stiefelhagen, Antonio Torralba, Raquel Urtasun, and Sanja Fidler. Movieqa: Understanding stories in movies through question-answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4631–4640, 2016.

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

[174] Laura von Rueden, Sebastian Mayer, Katharina Beckh, Bogdan Georgiev, Sven Giesselbach, Raoul Heese, Birgit Kirsch, Julius Pfrommer, Annika Pick, Rajkumar Ramamurthy, et al. Informed machine learning–a taxonomy and survey of integrating knowledge into learning systems. *arXiv preprint arXiv:1903.12394*, 2019.

[175] Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M. Rush, Bart van Merriënboer, Armand Joulin, and Tomas Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. 2015. cite arxiv:1502.05698.

[176] Thomas Winters, Giuseppe Marra, Robin Manhaeve, and Luc De Raedt. Deepstochlog: Neural stochastic logic programming. *arXiv preprint arXiv:2106.12574*, 2021.

[177] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 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, Online, October 2020. Association for Computational Linguistics.

[178] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771, 2019.

[179] Hao Wu, Jiayuan Mao, Yufeng Zhang, Yuning Jiang, Lei Li, Weiwei Sun, and Wei-Ying Ma. Unified visual-semantic embeddings: Bridging vision and language with structured meaning representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6609–6618, 2019.

[180] Te-Lin Wu, Alex Spangher, Pegah Alipoormolabashi, Marjorie Freedman, Ralph Weischedel, and Nanyun Peng. Understanding multimodal procedural knowledge by sequencing multimodal instructional manuals. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4525–4542, 2022.

[181] Yijun Xiao and William Yang Wang. Quantifying uncertainties in natural language processing tasks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 7322–7329, 2019.

[182] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Broeck. A semantic loss function for deep learning with symbolic knowledge. In *ICML*, pages 5502–5511. PMLR, 2018.

[183] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5502–5511. PMLR, 10–15 Jul 2018.

[184] Semih Yagcioglu, Aykut Erdem, Erkut Erdem, and Nazli Ikizler-Cinbis. Recipeqa: A challenge dataset for multimodal comprehension of cooking recipes. In *EMNLP*, 2018.

[185] Bishan Yang and Tom Mitchell. Leveraging knowledge bases in lstms for improving machine reading. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1436–1446, 2017.

[186] Tsung-Yen Yang, Michael Y Hu, Yinlam Chow, Peter J Ramadge, and Karthik Narasimhan. Safe reinforcement learning with natural language constraints. *NeurIPS*, 34:13794–13808, 2021.

[187] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pages 5753–5763, 2019.

[188] Peter Young, Alice Lai, Micah Hodosh, and Julia Hockenmaier. From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions. *Transactions of the Association for Computational Linguistics*, 2:67–78, 2014.

[189] Zhou Yu, Yuhao Cui, Jun Yu, Dacheng Tao, and Qi Tian. Multimodal unified attention networks for vision-and-language interactions. *arXiv preprint arXiv:1908.04107*, 2019.

[190] Peng Zhang and Maged N Kamel Boulos. Generative ai in medicine and healthcare: Promises, opportunities and challenges. *Future Internet*, 15(9):286, 2023.

[191] Xiao Zhang, Maria Leonor Pacheco, Chang Li, and Dan Goldwasser. Introducing DRAIL – a step towards declarative deep relational learning. In *Proceedings of the Workshop on Structured Prediction for NLP*, pages 54–62, Austin, TX, November 2016. Association for Computational Linguistics.

[192] Zhihan Zhang, Xiubo Geng, Tao Qin, Yunfang Wu, and Daxin Jiang. Knowledge-aware procedural text understanding with multi-stage training. In *Proceedings of the Web Conference 2021*, 2021.

[193] Zhihan Zhang, Xiubo Geng, Tao Qin, Yunfang Wu, and Daxin Jiang. Knowledge-aware procedural text understanding with multi-stage training. In *Proceedings of the Web Conference 2021*, pages 3512–3523, 2021.

[194] Chen Zheng and Parisa Kordjamshidi. Srlgrn: Semantic role labeling graph reasoning network. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8881–8891, 2020.

[195] Chen Zheng and Parisa Kordjamshidi. Relevant commonsense subgraphs for "what if..." procedural reasoning. In *Findings of ACL 2022*, pages 1927–1933, 2022.

[196] Da Zheng, Minjie Wang, Quan Gan, Zheng Zhang, and George Karypis. Learning graph neural networks with deep graph library. WWW '20, New York, NY, USA, 2020. Association for Computing Machinery.

[197] Lingxue Zhu and Nikolay Laptev. Deep and confident prediction for time series at uber. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 103–110. IEEE, 2017.

[198] Barret Zoph, Deniz Yuret, Jonathan May, and Kevin Knight. Transfer learning for low-resource neural machine translation. In *EMNLP*, pages 1568–1575, 2016.