UNIVERSITY OF POTSDAM

MASTER'S THESIS

# SoPa++: Leveraging explainability from hybridized RNN, CNN and weighted finite-state neural architectures

*Author:*
Atreya SHANKAR
799227

*1st Supervisor:*
Dr. Sharid LOÁICIGA
University of Potsdam

*2nd Supervisor:*
Mathias MÜLLER
University of Zurich

*A thesis submitted in fulfillment of the requirements for the degree of Cognitive Systems: Language, Learning, and Reasoning (M.Sc.)*

*in the*

Foundations of Computational Linguistics Research Group
Department of Linguistics

April 11, 2021

# Contents

**1 Introduction** **1**
1.1 Motivation . . . 1
1.2 Research questions . . . 2
1.3 Thesis structure . . . 2

**2 Background concepts** **3**
2.1 Explainable artificial intelligence . . . 3
2.1.1 Transparency . . . 3
2.1.2 Explainability and XAI . . . 4
2.1.3 Terminology clarification . . . 4
2.1.4 Post-hoc explainability techniques . . . 5
2.1.5 Performance-interpretability tradeoff . . . 6
2.1.6 Explainability metrics . . . 7
2.2 Straight-through estimator . . . 8
2.3 Finite-state automata . . . 8
2.3.1 Finite-state automaton . . . 9
2.3.2 Regular expressions . . . 10
2.3.3 Weighted finite-state automaton . . . 10
2.4 SoPa . . . 12
2.4.1 Linear-chain WFAs . . . 12
2.4.2 Document score . . . 13
2.4.3 Computational graph . . . 13
2.4.4 Patterns hyperparameter $P$ . . . 14
2.4.5 Transparency . . . 14
2.4.6 Explainability . . . 14

**3 Data and methodologies** **16**
3.1 Facebook Multilingual Task Oriented Dialog . . . 16
3.1.1 Preprocessing . . . 16
3.1.2 Summary statistics . . . 18
3.1.3 Performance range . . . 19
3.2 SoPa++ . . . 19
3.2.1 Strict linear-chain WFA-$\omega$'s . . . 19
3.2.2 Document score . . . 21
3.2.3 TauSTE . . . 21
3.2.4 Computational graph . . . 22
3.2.5 Transparency . . . 24
3.3 RE proxy . . . 25
3.3.1 Path-augmented document score . . . 25
3.3.2 RE lookup layer . . . 26
3.3.3 Assembling RE proxy . . . 27
3.3.4 Computational graph . . . 28
3.3.5 Transparency . . . 28

3.3.6 Explainability . . . 28
3.4 Differences between SoPa and SoPa++ . . . 29
3.5 RQ1: Evaluating performance of SoPa++ . . . 29
3.5.1 Training . . . 30
3.5.2 Evaluation . . . 30
3.6 RQ2: Evaluating explanations by simplification . . . 31
3.6.1 Softmax distance norm . . . 31
3.6.2 Binary misalignment rate . . . 32
3.7 RQ3: Interesting and relevant explanations . . . 32
3.7.1 Relative linear weights . . . 32
3.7.2 Samples from RE lookup layer . . . 32

**4 Results** **33**
4.1 RQ1: Evaluating performance of SoPa++ . . . 33
4.2 RQ2: Evaluating explanations by simplification . . . 33
4.3 RQ3: Interesting and relevant explanations . . . 36

**5 Discussion** **40**
5.1 RQ1: Evaluating performance of SoPa++ . . . 40
5.2 RQ2: Evaluating explanations by simplification . . . 40
5.3 RQ3: Interesting and relevant explanations . . . 41

**6 Conclusions** **43**

**7 Further work** **44**
7.1 Efficiency . . . 44
7.2 Generalization . . . 44
7.3 Modeling . . . 45
7.4 Explainability . . . 45

**Bibliography** **46**

# List of Figures


1 Parameter counts of recently released pre-trained language models which showed competitive or SOTA performance when fine-tuned over a range of NLP tasks; figure taken from Sanh et al. (2019) . . . . . 2

2 Examples of various target audiences in XAI; figure taken from Arrieta et al. (2020) . . . . . 5
3 Qualitative visualization of the performance-interpretability tradeoff; figure taken from Arrieta et al. (2020) . . . . . 7
4 Visualization of the vanilla STE's forward and backward passes . . . . 8
5 Regular expression '`how are (you|they) doing ?`' converted into a strict linear-chain NFA; double circles on state 5 indicate an accepting state . . . . . 10
6 Visualization of a (non-strict) linear-chain NFA with self-loop (blue), $\epsilon$ (red) and main-path (black) transitions; figure adapted from Schwartz, Thomson, and Smith (2018) . . . . . 12
7 Visualization of the SoPa model framework with two constituent WFAs highlighted in blue and red; figure taken from Schwartz, Thomson, and Smith (2018) . . . . . 14

8 Frequency distribution of the preprocessed FMTOD data set by classes and partitions . . . . . 17
9 Visualization of a strict linear-chain NFA with $\omega$ (blue) and main-path (black) transitions . . . . . 20
10 Visualization of the TauSTE's forward and backward passes . . . . . 22
11 Visualization of the SoPa++ computational graph . . . . . 23
12 Visualization of the RE proxy computational graph . . . . . 27

13 Visualization of validation accuracy against number of training updates grouped by patterns $P$ and $\tau$-threshold hyperparameters; the different coloured lines represent random seed iterations with a specific initial random seed . . . . . 34
14 Visualization of accuracy and model-pair distance metrics against the $\tau$-threshold hyperparameter grouped by the patterns hyperparameter $P$ 35
15 Relative linear layer weights applied to TauSTE neurons for the best performing small RE proxy model with a test accuracy of 97.4% . . . . 36
16 Ten sampled regular expressions from the RE lookup layer corresponding to TauSTE neuron 19 for the best performing small RE proxy model 37
17 Ten sampled regular expressions from the RE lookup layer corresponding to TauSTE neuron 25 for the best performing small RE proxy model 38
18 Ten sampled regular expressions from the RE lookup layer corresponding to TauSTE neuron 17 for the best performing small RE proxy model 39

# List of Tables

1 Frequency of the preprocessed FMTOD data set classes grouped by partitions; Σ signifies the cumulative frequency statistic . . . . . 17
2 Utterance length summary statistics and examples for the preprocessed FMTOD data set; $\mu$ signifies the cumulative summary statistic . . . . . 18
3 Studies that addressed the FMTOD English language intent classification task along with their relevant summaries and accuracy ranges . . 18
4 Summarized differences for SoPa vs. SoPa++ . . . . . 29
5 Three different SoPa++ model sizes used during training with corresponding patterns hyperparameter $P$ and parameter counts . . . . . 30
6 Test accuracies of the SoPa++ models grouped by model sizes and $\tau$-threshold hyperparameters; mean accuracies and standard deviations were calculated across random seed iterations; bold scores show best performing model for each model size . . . . . 34
7 Test accuracies and model-pair distances metrics of the SoPa++ and RE proxy models grouped by model sizes and $\tau$-thresholds; accuracies and standard deviations were calculated across random seed iterations; bold test accuracies show model pairs where mean accuracies are closest to one another; bold distance metrics indicate mean metrics that are closest to zero . . . . . 35

# Chapter 1

# Introduction

In this chapter, we introduce this thesis by presenting its motivation, describing our research questions and providing an overview of the thesis structure. We start off by considering the current trend of utilizing increasingly large deep learning models to address Machine Learning (ML) and Natural Language Processing (NLP) tasks.

## 1.1 Motivation

With the recent trend of increasingly large deep learning models achieving State-Of-The-Art (SOTA) performance on a myriad of ML tasks including in NLP as shown in Figure 1, several studies argue for focused research into Explainable Artificial Intelligence (XAI) to address emerging concerns such as security risks and inductive biases associated with black-box models (Doran, Schulz, and Besold, 2017; Townsend, Chaton, and Monteiro, 2019; Danilevsky et al., 2020; Arrieta et al., 2020). Of these studies, Arrieta et al. (2020) provide an extensive overview of XAI and related concepts based on a thorough literature review of ∼400 XAI research contributions published to date. In particular, Arrieta et al. (2020) explore and classify a variety of machine-learning models into transparent and black-box categories depending on their degrees of transparency. Furthermore, they explore taxonomies of post-hoc explainability techniques aimed at effectively explaining black-box models. Of high relevance to this thesis are the local explanations, feature relevance and explanations by simplification post-hoc explainability techniques.

Through our own survey of recent literature on explainability techniques used in NLP, we came across several interesting studies employing the local explanations, feature relevance and explanations by simplification explainability techniques to better explain black-box models; particularly deep neural networks. (Schwartz, Thomson, and Smith, 2018; Peng et al., 2018; Suresh et al., 2019; Wang and Niepert, 2019; Jiang et al., 2020). Drawing inspiration from the work of Schwartz, Thomson, and Smith (2018), we focus this thesis on further developing their **So**ft **Pa**tterns (SoPa) model; which represents a hybridized RNN, CNN and Weighted Finite-State Automaton (WFA) neural network architecture. We modify the SoPa model by changing key aspects of its architecture which ultimately allows us to conduct effective explanations by simplification and abbreviate this modified model as **SoPa++**, which signifies an improvement or major modification over the SoPa model. Finally, we evaluate both the performance and explanations by simplification of the SoPa++ model on the Facebook Multilingual Task Oriented Dialog data set (FMTOD; Schuster et al. 2019); focusing on the English-language intent classification task.

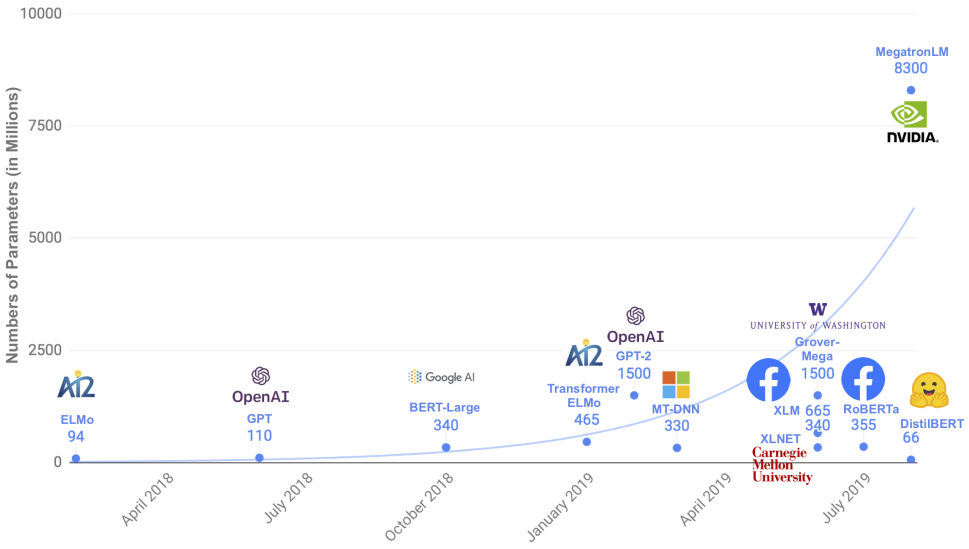


FIGURE 1: Parameter counts of recently released pre-trained language models which showed competitive or SOTA performance when fine-tuned over a range of NLP tasks; figure taken from Sanh et al. (2019)

## 1.2 Research questions

With the introduction of our SoPa++ model, we aim to answer the following three research questions:

1. Does SoPa++ contribute to competitive performance[1] on the FMTOD English language intent classification task?
2. To what extent does SoPa++ contribute to effective explanations by simplification on the FMTOD English language intent classification task?
3. What interesting and relevant explanations can SoPa++ provide on the FMTOD English language intent classification task?

## 1.3 Thesis structure

We now summarize the contents and structure of this thesis.

**Chapter 1:** Introduce this thesis, its contents and our research questions.

**Chapter 2:** Describe the background concepts utilized in this thesis.

**Chapter 3:** Describe the FMTOD data set and methodologies pursued in this thesis.

**Chapter 4:** Describe the results obtained from our methodologies.

**Chapter 5:** Interpret the results and discuss their implications.

**Chapter 6:** Summarize and conclude the findings of this thesis.

**Chapter 7:** Describe future work to expand on our research questions.

[1] We define competitive performance as the scenario where a mean performance metric on a certain task falls within the range obtained from other recent studies on the same task

# Chapter 2

# Background concepts

In this chapter, we describe the various background concepts which are necessary in order to support the methodologies pursued in this thesis. These background concepts range from conceptualizations and definitions in XAI, variants of finite-state automata, straight-through estimators and finally the SoPa model framework. We start off by introducing the most important concepts related to XAI.

## 2.1 Explainable artificial intelligence

In this section, we lay out background concepts for Explainable Artificial Intelligence (XAI) which have been largely adopted from Arrieta et al. (2020). The study is particularly helpful for us since it summarizes the findings of ∼400 XAI contributions and presents these findings in the form of well-defined concepts and taxonomies. In addition, the study discusses the many possible future directions of XAI research.

### 2.1.1 Transparency

One of the key contributions of Arrieta et al. (2020) to the field of XAI is the introduction of the concept of transparency, as well as the classification of various Machine Learning (ML) models into transparent and black-box categories. In this section, we provide an adapted definition of transparency and list examples of transparent and black-box ML models.

**Definition 1** (Transparency; Arrieta et al. 2020; Page 4, Section 2.1)**.** A model is considered to be transparent if it is understandable on its own without any external techniques to assist its understandability. Since a model can provide different degrees of understandability, transparent models are split into three possibly mutually-inclusive categories; specifically simulatable models, decomposable models and algorithmically transparent models.

*Remark* 1.1. *Simulatability* refers to the ability of a model's inner-mechanisms being simulated strictly by a human. Therefore, model simplicity plays a major role in this class.

*Remark* 1.2. *Decomposability* refers to the ability to clearly understand the individual parts of a model; such as its inputs, parameters and basic computational mechanisms.

*Remark* 1.3. *Algorithmic transparency* refers to the ability of a human to understand the algorithmic processes used in the model to produce any given output from any given input.

*Remark* 1.4. A model is considered transparent if it falls into one or more of the aforementioned transparency categories. If a model cannot satisfy any of the requirements of being transparent, then it is classified as a *black-box* model.

*Remark* 1.5. As recommended by Arrieta et al. (2020, Page 3, Section 2.1), we use the terms *transparency* and *interpretability* to refer to the same feature. Therefore, we use these terms equivalently and interchangeably.

Examples of well-known transparent ML models are linear/logistic regressors, decision trees and rules-based learners. Similarly, common examples of black-box ML models are tree ensembles and deep neural networks. Arrieta et al. (2020) provide extensive justifications using the aforementioned three criteria in conducting model classifications into the transparent and black-box categories. We would direct the reader to their study for a full analysis and justification of these classifications.

### 2.1.2 Explainability and XAI

Another key contribution of Arrieta et al. (2020) is a unified conceptualization of explainability and XAI based on an extensive literature review. Drawing inspiration from their study, we provide adapted definitions of explainability and XAI, as well as comment on interesting aspects related to XAI.

**Definition 2** (Explainability; Arrieta et al. 2020; Page 4, Section 2.1)**.** Explainability refers to the interface between humans and a model, such that this interface is both an accurate proxy of the model and comprehensible to humans.

**Definition 3** (Explainable Artificial Intelligence; Arrieta et al. 2020; Page 4, Section 2.2)**.** An explainable artificial intelligence is one that produces details to make its functioning understandable to a given target audience.

Arrieta et al. (2020) observe that black-box ML models are increasingly being employed to make important predictions in critical contexts, citing high-risk areas such as precision medicine and autonomous vehicles. Of particular relevance to the field of Natural Language Processing (NLP), the study notes a myriad of issues related to inductive biases within training data sets and the ethical issues involved in using black-box models trained on such data sets. As a result, they describe an increased demand for transparency in black-box ML models from the various stakeholders in Artificial Intelligence (AI). In addition, Arrieta et al. (2020) emphasize the presence of a target audience for XAI; implying that different XAI techniques should be employed for different target audiences. In their study, they provide examples of target audiences such as domain experts, end-users and managers as shown in Figure 2.

### 2.1.3 Terminology clarification

Based on their extensive literature review, Arrieta et al. (2020) observe that many studies tend to misuse the terms interpretability and explainability by using them interchangeably. To address this, they provide clear conceptual differences between the terms. For example, Arrieta et al. (2020, Page 3, Section 2.1) state that *"interpretability refers to a passive characteristic of a model referring to the level at which a given model makes sense for a human observer."* In contrast, Arrieta et al. (2020, Page 3, Section 2.1) state that *"explainability can be viewed as an active characteristic of a model, denoting any action or procedure taken by a model with the intent of clarifying or detailing its internal functions."*

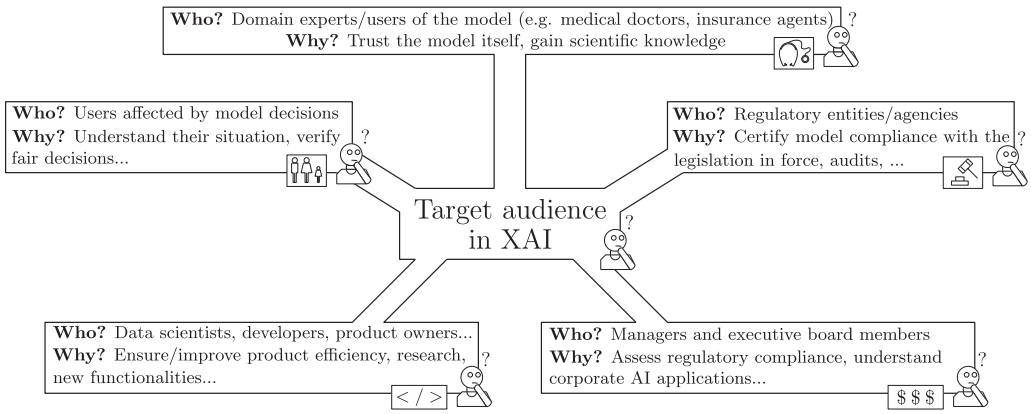


FIGURE 2: Examples of various target audiences in XAI; figure taken from Arrieta et al. (2020)

In summary, we gather that interpretability (or transparency as per Remark 1.5) refers to an inherent or passive feature of a model. On the other hand, explainability refers to an active characteristic undertaken by the model and its developers to explain the model's inner mechanisms. In addition, explainability entails the presence of a target audience; which may not necessarily be the case for interpretability or transparency.

### 2.1.4 Post-hoc explainability techniques

Based on the aforementioned classification of ML models into transparent and black-box models, Arrieta et al. (2020) expound on explainability techniques for each of these model types. Due to their transparent nature, the study states that transparent ML models are usually explainable in themselves to most target audiences and therefore usually do not require any external technique to extract explanations. The study does however highlight some target audiences, such as non-expert users, who may require external explainability techniques such as model output visualizations in order to explain the inner workings of transparent ML models.

For the case of black-box models, Arrieta et al. (2020) argue that separate or external techniques must be utilized in order to reasonably explain these models. Such explainability techniques are referred to in the study as post-hoc explainability techniques; which is derived from the idea that explanations for such models are usually extracted post-modeling. Notable examples of post-hoc explainability techniques include local explanations, feature relevance and explanations by simplification; for which we provide adapted definitions below.

**Definition 4** (Local explanations; Arrieta et al. 2020; Page 11, Section 4.1)**.** Local explanations operate by segmenting a model's solution space into subspaces and provide explanations for the less complex model subspaces.

*Remark* 4.1. Local explanations are commonly used in XAI research and function by using differentiating properties on model solution space subsets.

*Remark* 4.2. Two well-known examples of local explainability techniques are Local Interpretable Model-Agnostic Explanations (LIME; Ribeiro, Singh, and Guestrin 2016) and G-REX (Konig, Johansson, and Niklasson, 2008).

**Definition 5** (Feature relevance; Arrieta et al. 2020; Page 11, Section 4.1)**.** Feature relevance explanation methods operate by computing an importance score for the

model's input features over the model's outputs. These scores typically quantify how sensitive the model's output is to perturbations in the model's inputs or internal features.

*Remark* 5.1. Feature relevance methods can be considered as indirect methods of explaining a model.

*Remark* 5.2. Well-known feature relevance explainability techniques include the Shapley Additive Explanations (SHAP; Lundberg and Lee 2017) and the occlusion sensitivity method (Zeiler and Fergus, 2014).

**Definition 6** (Explanations by simplification; Arrieta et al. 2020; Page 11, Section 4.1)**.** Explanations by simplification refer to techniques where a simplified proxy model is built to approximate and explain a more complex model. The simplified proxy model usually has to fulfil the the joint criteria of reducing its complexity compared to its antecedent model while maximizing its resemblance to its antecedent and keeping a similar performance score.

*Remark* 6.1. In this thesis, we refer to the original black-box model as an *antecedent* model and the simplified model as the *proxy* model. Furthermore, we qualify that all proxy models must be designed to globally approximate their respective antecedent models.

*Remark* 6.2. Bastani, Kim, and Bastani (2017) and Tan et al. (2018) are examples of studies that extract and distill simpler proxy models from complex antecedent models.

Through our own survey of recent literature on explainability techniques used in NLP, we came across several interesting studies employing the local explanations, feature relevance and explanations by simplification explainability techniques to better explain black-box models; particularly deep neural networks. (Schwartz, Thomson, and Smith, 2018; Peng et al., 2018; Suresh et al., 2019; Wang and Niepert, 2019; Jiang et al., 2020). We expound more on these in Sections 2.3 and 2.4 respectively.

### 2.1.5 Performance-interpretability tradeoff

An interesting and insightful contribution of Arrieta et al. (2020) is their conceptualization of the performance-interpretability tradeoff; which in its essence states that the interpretability or transparency of models is negatively correlated with performance on ML tasks. Arrieta et al. (2020, Page 18, Section 5.1) introduce the performance-interpretability tradeoff with the caveat that *"the matter of interpretability versus performance is one that repeats itself through time, but as any other big statement, has its surroundings filled with myths and misconceptions."* To address some of the aforementioned myths and misconceptions, Arrieta et al. (2020) first disprove the generic statement that black-box models are *always* more performant by pointing to case studies which show that transparent models can perform on-par or better than black-box models when the function to be modeled is simple, data is well-structured and input features are available with high quality.

With these exceptions clarified, Arrieta et al. (2020) then provide cases which show that black-box models perform better than transparent models; which ultimately gives rise to the performance-interpretability tradeoff as reflected in Figure 3. According to them, this is usually the case when the function to be modeled is sufficiently complex and where the input data has high diversity or variance; and possibly contains significant noise.

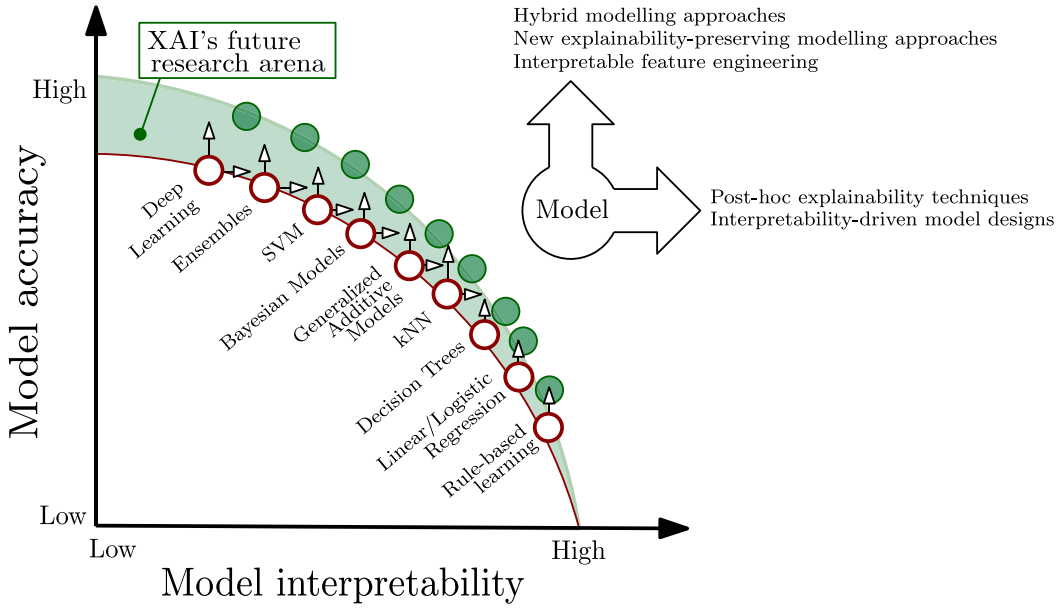


FIGURE 3: Qualitative visualization of the performance-interpretability tradeoff; figure taken from Arrieta et al. (2020)

### 2.1.6 Explainability metrics

Towards the end of their study, Arrieta et al. (2020) note two major limitations of the current state of XAI research. Firstly, they observe the lack of a unified conceptualization of explainability and transparency between various studies. To address this, they acknowledge that their study could provide a good unified starting point for other XAI studies to branch out from. Secondly, Arrieta et al. (2020) note the lack of a unified metric that denotes how explainable any given model is. They explain why developing such a metric has been a difficult process for many XAI studies; particularly because such a metric would entail incorporating psychological, sociological and cognitive elements to accommodate the goodness of fit of an explainability technique to a certain target audience. Furthermore, incorporating such elements might involve significant amounts of subjectivity in the desired metric.

To reduce some of the aforementioned subjectivity involved, Miller (2019) and Arrieta et al. (2020) provide three guidelines of what could constitute a good explanation based on human psychology, sociology and cognitive sciences:

1. Firstly, they observe that explanations are better when *constrictive*; meaning an explanation is good if it not only explains why a model made decision X, but also why it made decision X over decision Y.

2. Next, they suggest that good explanations should be able to communicate *causal* links over probabilities; which could be a challenge for black-box models which generally compute aggregate probabilities without necessarily considering causal links.

3. Finally, they recommend that explanations are better when *selective*; meaning that a good explanation should be able to selectively provide the most important causal links instead of all possible causal links as these might be irrelevant or confusing to the target audience.

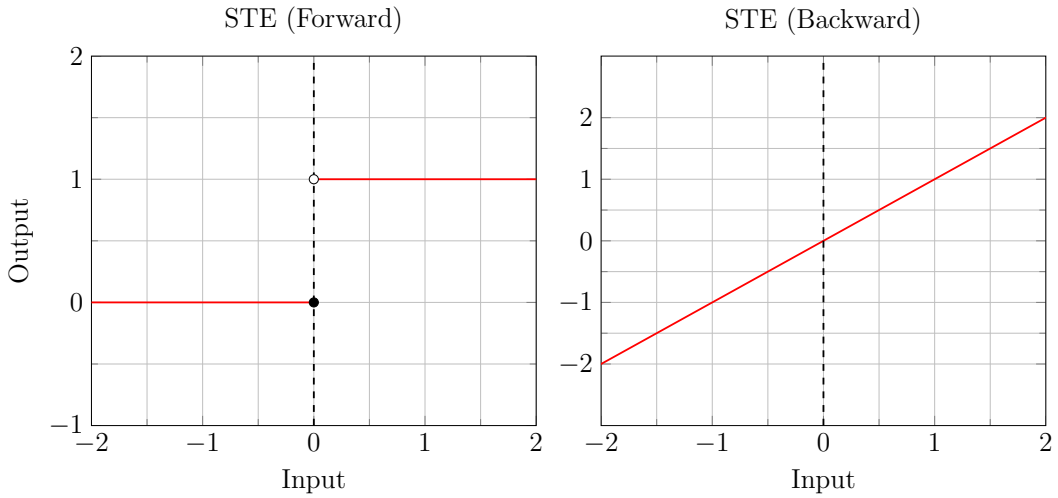


FIGURE 4: Visualization of the vanilla STE's forward and backward passes

## 2.2 Straight-through estimator

Quantized neural networks refer to neural networks that contain layers which transmit piecewise discrete signals and have been an object of active research in regards to low-precision and low-resource computing. One of the main challenges in training such quantized neural networks is that their gradients tend to vanish almost everywhere because the derivatives of such piecewise discrete activation functions typically default to zero (Bengio, Léonard, and Courville, 2013; Courbariaux and Bengio, 2016; Yin et al., 2019).

One significant workaround for this issue was proposed by Bengio, Léonard, and Courville (2013) through the introduction of the Straight-Through Estimator (STE). In the vanilla version of the STE, the STE neuron emits a signal of 1 when its input is strictly positive, and emits a zero signal in all other cases (Equation 1). The STE then uses a simple identity function to estimate the gradient during the backward pass (Equation 2). The vanilla STE is visualized in Figure 4.

$$\mathrm{STE}(x) = \begin{cases} 1 & x \in (0, +\infty) \\ 0 & x \in (-\infty, 0] \end{cases} \tag{1}$$

$$\mathrm{STE}'(x) = x \tag{2}$$

Aside from the vanilla STE, several other flavors of STEs have been proposed by studies such as Courbariaux and Bengio (2016) and Yin et al. (2019). In general, these studies have shown that quantized neural networks perform competitively with their non-quantized counterparts; while providing performance-related benefits related to lower precision computing as well as enabling further research into threshold driven neural activation functions.

## 2.3 Finite-state automata

As mentioned in Section 2.1.4, our survey of explainability in NLP yielded several studies employing a post-hoc explainability techniques on deep neural networks. One common factor among several of these studies was the simplification of black-box models to Finite-state Automata (FAs). As this will eventually become a key

focus of this thesis, we define key concepts related to the FAs; as well as extensions to FAs.

### 2.3.1 Finite-state automaton

Finite-state automata is a collective term for two sub-categories of models; namely deterministic and nondeterministic finite-state automata. Here we provide definitions and descriptions for these mutually exclusive model categories.

**Definition 7** (Deterministic finite-state automaton; Sipser 1996)**.** A deterministic finite-state automaton is a 5-tuple $\mathcal{M} = \langle \Sigma, \mathcal{Q}, \delta, q_0, F \rangle$, with:

- a finite input alphabet $\Sigma$;
- a finite state set $\mathcal{Q}$;
- a transition function $\delta : \mathcal{Q} \times \Sigma \to \mathcal{Q}$;
- an initial state $q_0 \in \mathcal{Q}$;
- and a set of final or accepting states $F \subseteq Q$.

**Definition 8** (Nondeterministic finite-state automaton; Sipser 1996)**.** A nondeterministic finite-state automaton is a 5-tuple $\mathcal{M} = \langle \Sigma, \mathcal{Q}, \delta, q_0, F \rangle$, with:

- a finite input alphabet $\Sigma$;
- a finite state set $\mathcal{Q}$;
- a transition function $\delta : \mathcal{Q} \times \{\Sigma \cup \{\epsilon\}\} \to \mathcal{P}(\mathcal{Q})$;
- an initial state $q_0 \in \mathcal{Q}$;
- and a set of final or accepting states $F \subseteq Q$.

*Remark* 8.1. $\epsilon \notin \Sigma$ refers to an empty-string transition and results in a change of state without consuming any input token. These transitions are unique to nondeterministic finite-state automata.

*Remark* 8.2. Self-loop transitions refer to special transitions which consume an input token while staying at the same state. These are allowed in both deterministic and nondeterministic finite-state automata.

*Remark* 8.3. $\mathcal{P}(\mathcal{Q})$ refers to the power set of $\mathcal{Q}$, or otherwise the collection of all subsets of $\mathcal{Q}$.

**Definition 9** (Linear-chain finite-state automaton; Schwartz, Thomson, and Smith 2018)**.** Any finite-state automaton is considered linear-chain if there exists only one set of consecutive transition states leading from the start to the accepting state. Linear-chain finite-state automata furthermore possess only one start and accepting state and only allow transitions to the same or next state which is closer to the accepting state.

*Remark* 9.1. A linear-chain finite-state automaton can be considered as *strict* if it does not permit self-loop transitions and therefore only permits transitions to adjacent states which are strictly closer to the accepting state. An example of a strict linear-chain NFA can be seen in Figure 5.

*Remark* 9.2. The presence of the linear-chain terminology is effectively absent in theoretical computer science literature but is prevalent in practical research related to Hidden Markov Models (HMMs) and Conditional Random Fields (CRFs) such as in Tsuruoka, Tsujii, and Ananiadou (2009). As a result, we are only able to provide a text-based definition here.

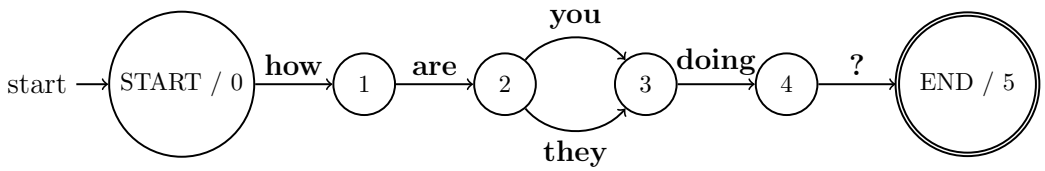


FIGURE 5: Regular expression ‘‘`how are (you|they) doing ?`’’ converted into a strict linear-chain NFA; double circles on state 5 indicate an accepting state

A string $\boldsymbol{x}$ is said to be *accepted* by any FA $\mathcal{M}$ if the current state after consuming string $\boldsymbol{x}$ is the accepting state. Similarly, a string $\boldsymbol{x}$ is said to be *rejected* by any FA $\mathcal{M}$ if the string $\boldsymbol{x}$ cannot be consumed or the current state after consuming $\boldsymbol{x}$ is a non-accepting state. The key difference between DFAs and NFAs is present in their transition functions; specifically that the former guarantees that each state allows for a unique transition to another state given a non-empty input string. Contrastingly, NFAs allow for arbitrary transitions without necessarily consuming an input token.

### 2.3.2 Regular expressions

Regular expressions (REs) and FAs have been shown to be equivalent in their expressive power in that they both recognize regular languages (Sipser, 1996). Historically, several algorithms have been studied and optimized for converting FAs to REs and vice-versa (McNaughton and Yamada, 1960; Thompson, 1968). Figure 5 shows a simple example of converting the regular expression ‘‘`how are (you|they) doing ?`’’ into a strict linear-chain NFA. The aforementioned regular expression is written using the Perl-compatible regular expression syntax with the ‘‘`|`’’ character indicating alternative possibilities.

### 2.3.3 Weighted finite-state automaton

Weighted finite-state automata (WFAs) are extensions of finite-state automata which allow for the assignment of numerical weights to transitions given a semiring to govern the algebra of the transitions. Compared to the aforementioned FAs that only accept or reject strings, WFAs numerically score strings which has made them useful for several applications in NLP such as tokenization and part-of-speech tagging (Maletti, 2017). Here, we provide extensive definitions of a semiring, WFA, path score and string score.

**Definition 10** (Semiring; Kuich and Salomaa 1986)**.** A semiring is a set $\mathbb{K}$ along with two binary associative operations $\oplus$ (addition) and $\otimes$ (multiplication) and two identity elements: $\bar{0}$ for addition and $\bar{1}$ for multiplication. Semirings require that addition is commutative, multiplication distributes over addition, and that multiplication by $\bar{0}$ annihilates, i.e., $\bar{0} \otimes a = a \otimes \bar{0} = \bar{0}$.

*Remark* 10.1. Semirings follow the following generic notation: $\langle \mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1} \rangle$.

*Remark* 10.2. A simple and common semiring is the real or sum-product semiring: $\langle \mathbb{R}, +, \times, 0, 1 \rangle$. Two important semirings for this thesis are shown below.

*Remark* 10.3. **Max-sum** semiring: $\langle \mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0 \rangle$

*Remark* 10.4. **Max-product** semiring: $\langle \mathbb{R}_{>0} \cup \{-\infty\}, \max, \times, -\infty, 1 \rangle$

**Definition 11** (Weighted finite-state automaton; Peng et al. 2018)**.** A weighted finite-state automaton over a semiring $\mathbb{K}$ is a 5-tuple $\mathcal{A} = \langle \Sigma, \mathcal{Q}, \boldsymbol{\Gamma}, \boldsymbol{\lambda}, \boldsymbol{\rho} \rangle$, with:

- a finite input alphabet $\Sigma$;
- a finite state set $\mathcal{Q}$;
- transition matrix $\mathbf{\Gamma} : \mathcal{Q} \times \mathcal{Q} \times (\Sigma \cup \{\epsilon\}) \to \mathbb{K}$;
- initial vector $\boldsymbol{\lambda} : \mathcal{Q} \to \mathbb{K}$;
- and final vector $\boldsymbol{\rho} : \mathcal{Q} \to \mathbb{K}$.

*Remark* 11.1. $\Sigma^*$ refers to the (possibly infinite) set of all strings over the alphabet $\Sigma$, where * represents the Kleene star operator.

*Remark* 11.2. As with finite-state automata, it is also possible to construct (strict) linear-chain weighted finite-state automata. Similarly, it is possible to extract a FA from a WFA given a particular set of valid transitions.

**Definition 12** (Path score; Peng et al. 2018)**.** Let $\boldsymbol{\pi} = \langle \pi_1, \pi_2, \ldots, \pi_n \rangle$ be a sequence of adjacent transitions in $\mathcal{A}$, with each $\pi_i = \langle q_i, q_{i+1}, z_i \rangle \in \mathcal{Q} \times \mathcal{Q} \times (\Sigma \cup \{\epsilon\})$. The path $\boldsymbol{\pi}$ derives the $\epsilon$-free string $\boldsymbol{x} = \langle x_1, x_2, \ldots, x_m \rangle \in \Sigma^*$; which is a substring of the $\epsilon$-containing string $\boldsymbol{z} = \langle z_1, z_2, \ldots, z_n \rangle \in (\Sigma \cup \{\epsilon\})^*$. $\boldsymbol{\pi}$'s score in $\mathcal{A}$ is given by:

$$\mathcal{A}[\boldsymbol{\pi}] = \boldsymbol{\lambda}(q_1) \otimes \left( \bigotimes_{i=1}^{n} \mathbf{\Gamma}(\pi_i) \right) \otimes \boldsymbol{\rho}(q_{n+1}) \tag{3}$$

**Definition 13** (String score; Peng et al. 2018)**.** Let $\Pi(\boldsymbol{x})$ denote the set of all paths in $\mathcal{A}$ that derive $\boldsymbol{x}$. Then the string score assigned by $\mathcal{A}$ to string $\boldsymbol{x}$ is given by:

$$\mathcal{A}[\![\boldsymbol{x}]\!] = \bigoplus_{\boldsymbol{\pi} \in \Pi(\boldsymbol{x})} \mathcal{A}[\boldsymbol{\pi}] \tag{4}$$

*Remark* 13.1. Since $\mathbb{K}$ is a semiring, $\mathcal{A}[\![\boldsymbol{x}]\!]$ can be efficiently computed using the Forward algorithm (Baum and Petrie, 1966). Its dynamic program is summarized below without $\epsilon$-transitions for simplicity. $\Omega_i(q)$ gives the aggregate score of all paths that derive the substring $\langle x_1, x_2, \ldots, x_i \rangle$ and end in state $q$:

$$\Omega_0(q) = \boldsymbol{\lambda}(q) \tag{5a}$$

$$\Omega_{i+1}(q) = \bigoplus_{q' \in \mathcal{Q}} \Omega_i(q') \otimes \mathbf{\Gamma}(q', q, x_i) \tag{5b}$$

$$\mathcal{A}[\![\boldsymbol{x}]\!] = \bigoplus_{q \in \mathcal{Q}} \Omega_n(q) \otimes \boldsymbol{\rho}(q) \tag{5c}$$

*Remark* 13.2. The Forward algorithm can be generalized to any semiring (Eisner, 2002) and has a runtime of $O(|Q|^3 + |Q|^2|\boldsymbol{x}|)$ (Schwartz, Thomson, and Smith, 2018); notably with a linear runtime with respect to the length in tokens of the input string $\boldsymbol{x}$.

*Remark* 13.3. A special case of Forward is the Viterbi algorithm, where the addition $\oplus$ operation is constrained to the maximum operator (Viterbi, 1967). Viterbi therefore returns the highest scoring path $\boldsymbol{\pi}$ that derives the input string $\boldsymbol{x}$.

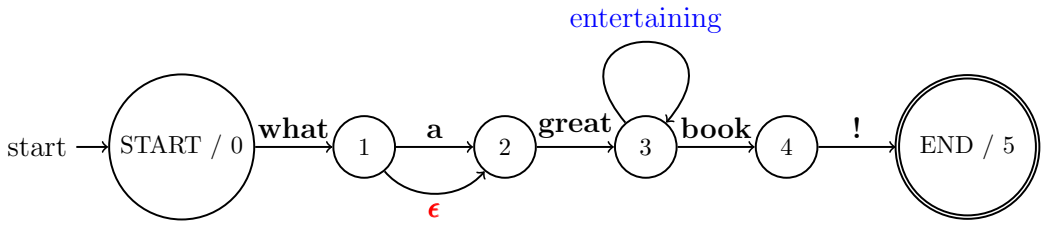


FIGURE 6: Visualization of a (non-strict) linear-chain NFA with self-loop (blue), $\epsilon$ (red) and main-path (black) transitions; figure adapted from Schwartz, Thomson, and Smith (2018)

## 2.4 SoPa

As mentioned previously in Chapter 1, a key focus of this thesis is in improving the **So**ft **Pa**tterns (SoPa) model presented in Schwartz, Thomson, and Smith (2018). In this section, we describe important aspects of the SoPa model which ultimately become relevant for our methodologies. Schwartz, Thomson, and Smith (2018) introduce SoPa as a novel and lightweight hybridized RNN, CNN and WFA-based neural architecture. SoPa resembles a RNN because it processes text sequentially and can encode strings of arbitrary lengths. Similarly, the architecture contains a variable number of constrained linear-chain WFAs with variable numbers of states or window sizes; which resembles both one-layer CNNs and an ensemble of linear-chain WFAs. The combination of the aforementioned features ultimately allows SoPa to learn soft versions of traditionally hard surface patterns.

In their study, Schwartz, Thomson, and Smith (2018) test the SoPa architecture on three separate sentiment classification tasks and compare the results with four baselines which include a bidirectional LSTM and a CNN. With their results, they show that SoPa performed on par or better than all baselines on all tasks. Additionally, they show that SoPa outperformed all baselines significantly in low-data settings. Finally, the authors present workflows to explain the SoPa model using both the local explanations and feature relevance explainability techniques.

### 2.4.1 Linear-chain WFAs

In regards to the WFAs used in SoPa, Schwartz, Thomson, and Smith (2018) utilize linear-chain WFAs where each linear-chain WFA is alloted a sequence of $|Q|$ states. Each state $i$ in the linear-chain WFA has three possible outgoing transitions; namely a **self-loop transition** which consumes a token but stays in the same state $i$, an **$\epsilon$-transition** which does not consume a token but transitions to state $i+1$ and a **main-path transition** which consumes a specific token and transitions to state $i+1$. Furthermore, Schwartz, Thomson, and Smith (2018) utilize only the max-sum and max-product semirings in their linear-chain WFAs. Figure 6 shows a sample NFA extracted from the aforementioned linear-chain WFA with all three aforementioned transitions, which could accept strings such as `"what a great book !"` and `"what a great entertaining book !"`.

To more concretely express these transitions, Schwartz, Thomson, and Smith (2018) provide the following formulation of the transition matrix $\mathbf{\Gamma}$ under the linear-chain structure. Here, $\mathbf{\Gamma}(x)$ represents a $|Q| \times |Q|$ matrix containing transition scores when consuming an input token $x$. $[\mathbf{\Gamma}(x)]_{i,j}$ corresponds to the cell value in $\mathbf{\Gamma}(x)$ for

row $i$ and column $j$ and represents the transition score when consuming token $x$ and transitioning from state $i$ to $j$.

$$[\mathbf{\Gamma}(x)]_{i,j} = \begin{cases} \boldsymbol{u}_i \cdot \boldsymbol{v}_x + a_i & \text{if } j = i \text{ (self-loop transition)}, \\ \boldsymbol{w}_i \cdot \boldsymbol{v}_x + b_i & \text{if } j = i + 1 \text{ (main-path transition)}, \\ \bar{0} & \text{otherwise.} \end{cases} \tag{6}$$

Here, $\boldsymbol{u}_i$ and $\boldsymbol{w}_i$ are learnable vectors and $a_i$ and $b_i$ are learnable scalar biases parameterizing transitions out of state $i$. $\boldsymbol{v}_x$ represents the word embedding for token $x$ and $\bar{0}$ represents the zero value in the semiring used as per Definition 10. Similarly, $\epsilon$-transitions are parameterized with the following representation in $\mathbf{\Gamma}$:

$$[\mathbf{\Gamma}(\epsilon)]_{i,j} = \begin{cases} c_i & \text{if } j = i + 1 \text{ ($\epsilon$-transition)} \\ \bar{0} & \text{otherwise.} \end{cases} \tag{7}$$

Here, $c_i$ represents a learnable scalar bias for $\epsilon$-transitions out of state $i$. Next, Schwartz, Thomson, and Smith (2018) fix the initial vector $\boldsymbol{\lambda} = [\bar{1}, \bar{0}, \ldots, \bar{0}]$ and the final vector $\boldsymbol{\rho} = [\bar{0}, \bar{0}, \ldots, \bar{1}]$, where $\bar{1}$ and $\bar{0}$ represent the one and zero values specified in the semiring as per Definition 10. Ultimately, these are formalisms to imply that there only exists one start and one accepting state and these are present at both extremes of the linear-chain WFA; which is ultimately consistent with the linear-chain definition as per Definition 9.

The aforementioned constraints in the linear-chain WFA structure result in the transition matrix $\mathbf{\Gamma}$ being reduced to a sparse diagonal matrix. Correspondingly, the runtime of the Forward algorithm under the linear-chain WFAs gets reduced to $O(|Q||\boldsymbol{x}|)$ compared to the original runtime in Remark 13.2. Finally, it is worth noting that Schwartz, Thomson, and Smith (2018, Page 3, Section 3.1) refer to the linear-chain WFAs simply as *"patterns"*. For brevity and consistency, we refer to linear-chain WFAs as patterns interchangeably.

### 2.4.2 Document score

Since SoPa was intended to compute scores for entire documents and not just short strings, Schwartz, Thomson, and Smith (2018, Page 3, Section 3.2) propose computing the string score, as per Definition 13, over all consecutive substrings in the document which ultimately returns a document score $s_{\text{doc}}(\boldsymbol{y})$ for an arbitrary document $\boldsymbol{y}$. The document score $s_{\text{doc}}(\boldsymbol{y})$ for a WFA would represent an aggregated score over all consecutive substrings and would therefore also depend on the semiring used in the WFAs. In the case of max-based semirings using the Viterbi algorithm, the document score $s_{\text{doc}}(\boldsymbol{y})$ would reflect the highest path score corresponding to a substring in document $\boldsymbol{y}$.

### 2.4.3 Computational graph

To explain the overall computational graph of the SoPa model, we refer to the visualization shown in Figure 7 with two WFAs highlighted in blue and red. The WFAs compute the aforementioned document scores as they traverse the document. Given max-based semirings, the max-pooled scores accumulate in an output layer as shown on the top of Figure 7. After traversing the full document, the max-pooled scores are passed through a Multi-Layer Perceptron (MLP) which conducts the final classification to an output label. It is worth noting that without $\epsilon$-transitions and

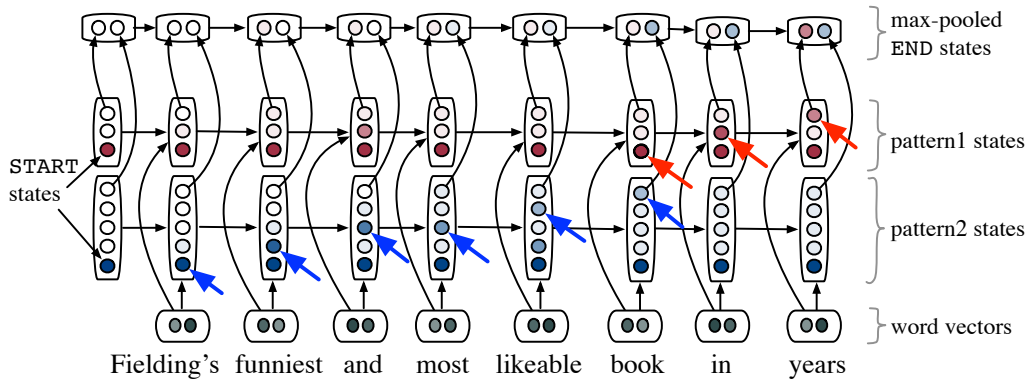


FIGURE 7: Visualization of the SoPa model framework with two constituent WFAs highlighted in blue and red; figure taken from Schwartz, Thomson, and Smith (2018)

self-loops, a linear-chain WFA with $|\mathcal{Q}|$ states should always consume $|\mathcal{Q}| - 1$ tokens. However, by allowing the aforementioned special transitions; it is possible for strings of variable lengths to be consumed since an $\epsilon$-transition can transition to the next state without consuming tokens while a self-loop can consume tokens without transitioning to the next state. This is indeed the case for Pattern 2 in Figure 7, when a self-loop is encountered in the transition from the token *"and"* to the token *"most"*.

### 2.4.4 Patterns hyperparameter $P$

Training the SoPa model requires both commonly used and special hyperparameters. Commonly used hyperparameters include the learning rate, neuron dropout and word dropout probabilities. A special hyperparameter unique to the SoPa model is the patterns hyperparameter $P$ which contains information on the number of linear-chain WFAs and the number of states allotted to each of them. The hyperparameter $P$ is encoded as a string with the following syntax: $\texttt{Length}_1\texttt{-Count}_1\texttt{\_}\ldots\texttt{\_Length}_k\texttt{-Count}_k$. An example of this hyperparameter $P$ is `5-15_4-10_3-5`, which would signify 15 patterns with 5 states each, 10 patterns with 4 states each and 5 patterns with 3 states each.

### 2.4.5 Transparency

Since we expounded on XAI in Section 2.1 and made a case for viewing ML models from the lens of XAI, it would only make sense to extend the same standards to the SoPa model. Based on the arguments made by Arrieta et al. (2020), we can classify the SoPa model as a black-box model since it closely resembles RNNs and CNNs; and a strong case has already been made in their study regarding the black-box natures of both RNNs and CNNs. Naturally, this would imply that post-hoc explainability techniques are required to explain the SoPa model.

### 2.4.6 Explainability

In their study, Schwartz, Thomson, and Smith (2018, Page 7, Section 7) describe two simple post-hoc explainability techniques for the SoPa model. One method involves the usage of back-pointers during the Viterbi computation to determine the patterns

and substrings in a document which contributed the highest pattern scores. Another method involves zeroing out the corresponding patterns scores via the occlusion sensitivity method to determine which pattern had the greatest impact on each classification decision.

Using the post-hoc explainability taxonomies described in Section 2.1.4, we can correspondingly classify the explainability techniques presented in Schwartz, Thomson, and Smith (2018) using terminology consistent with XAI research. The first explainability technique uses individual text samples to determine the highest scoring substrings in documents; as well as the patterns or WFAs corresponding to them. Since this analysis is conducted at an individual document level and is never synthesized to a more global context, we would classify this under the local explanations explainability technique. The next technique involves an occlusion or sensitivity analysis over all patterns and documents to determine which pattern had the greatest impact for each class. Since this involves systematic perturbation to determine the importance of pattern features, we would classify this technique as a feature relevance explainability technique. While the target audience(s) of these explainability techniques is not explicitly mentioned in their study, we infer that the target audience for these techniques is likely to be end-users since the outputs of these post-hoc explainability techniques are generally easy to understand.

In order to objectively probe the quality of the post-hoc explainability techniques proposed by Schwartz, Thomson, and Smith (2018), we utilize the three guidelines provided in Section 2.1.6; namely that a good explanation should be constrictive, causal and selective. For the constrictive quality, it is likely that the explainability techniques do not meet this criterion since they only highlight individual features that were important for SoPa, and do not necessarily go into detail regarding why these features superseded adjacent features. Next, the explainability techniques likely do not fulfill the criterion of providing causal links; since they generally provide explanations by aggregating probabilistic quantities inside SoPa. Finally and in contrast, the explainability techniques likely pass the criterion of being selective since they only provide the most important features for an explanation.

# Chapter 3

# Data and methodologies

In this chapter, we describe the FMTOD data set in greater detail and survey the performance of other studies that addressed this data set and its tasks. Following this, we introduce the core content of this thesis by describing the methodologies used for answering our three research questions. Comprehensive source code reflecting our methodologies can be found in our public GitHub repository[1].

## 3.1 Facebook Multilingual Task Oriented Dialog

Schuster et al. (2019) originally released the Facebook Multilingual Task Oriented Dialog (FMTOD) data set to encourage research in cross-lingual transfer learning for dialogue-oriented Natural Language Understanding (NLU) tasks; specifically from from high-resource to low-resource languages. The authors released the FMTOD data set with English as the high-resource language providing ∼43k utterances, and Spanish and Thai as low-resource languages providing a total of ∼14k utterances. Furthermore, they streamlined the data set towards two key tasks; namely intent classification and textual slot filling. In this thesis, we focus solely on the English language intent classification task in the FMTOD data set; which entails a multi-label sequence classification task with a total of 12 classes from alarm, reminder and weather-related domains. For brevity, we refer to the FMTOD English language intent classification data set as the FMTOD data set.

We chose to work with the FMTOD data set since it is both a recently released and well-studied data set (Schuster et al., 2019; Zhang et al., 2019; Zhang, Lyu, and Callison-Burch, 2020). We focus on the English language intent classification task since it is a relatively straightforward task which allows us to place a greater focus on performance and explainability. Furthermore, the English language subset entails the highest resources in the FMTOD data set. Finally, we find the FMTOD data set's intent classification task especially attractive because it allows us to test the SoPa++ model on a multi-class NLU problem; which is significantly different from the focus on binary classification sentiment detection tasks in SoPa.

### 3.1.1 Preprocessing

Given that we are handling text-based data in the FMTOD data set, it is necessary to preprocess this data first before proceeding with any modeling steps. We enumerate our preprocessing steps below:

1. We convert all FMTOD text samples into a lowercased format. This assists in simplifying and normalizing the textual data.

[1] https://github.com/atreyasha/spp-explainability

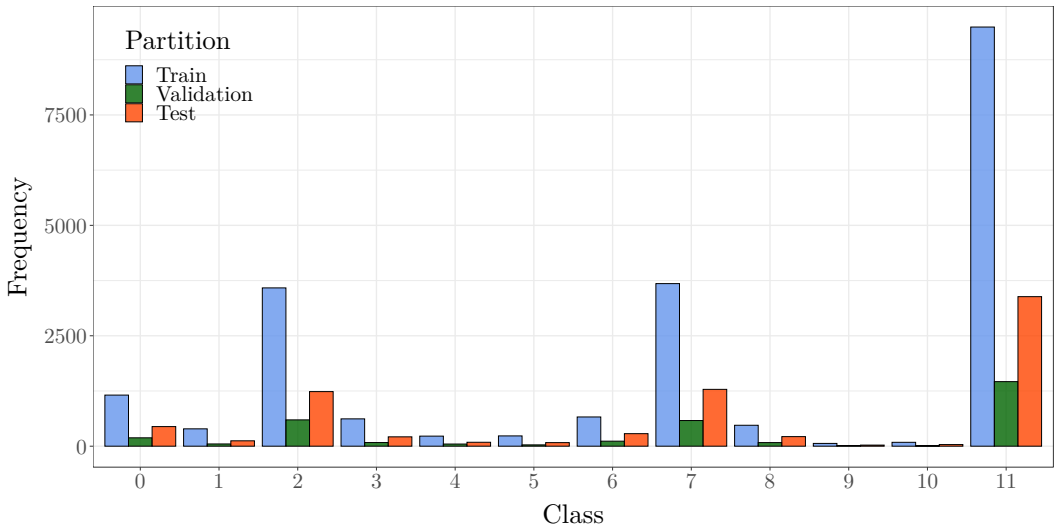


FIGURE 8: Frequency distribution of the preprocessed FMTOD data set by classes and partitions

| Class and description | Train | Validation | Test | Σ |
|---|---|---|---|---|
| 0: `alarm/cancel_alarm` | 1157 | 190 | 444 | 1791 |
| 1: `alarm/modify_alarm` | 393 | 51 | 122 | 566 |
| 2: `alarm/set_alarm` | 3584 | 596 | 1236 | 5416 |
| 3: `alarm/show_alarms` | 619 | 83 | 212 | 914 |
| 4: `alarm/snooze_alarm` | 228 | 49 | 89 | 366 |
| 5: `alarm/time_left_on_alarm` | 233 | 30 | 81 | 344 |
| 6: `reminder/cancel_reminder` | 662 | 114 | 284 | 1060 |
| 7: `reminder/set_reminder` | 3681 | 581 | 1287 | 5549 |
| 8: `reminder/show_reminders` | 474 | 82 | 217 | 773 |
| 9: `weather/check_sunrise` | 63 | 13 | 25 | 101 |
| 10: `weather/check_sunset` | 88 | 11 | 37 | 136 |
| 11: `weather/find` | 9490 | 1462 | 3386 | 14338 |
| Σ | 20672 | 3262 | 7420 | 31354 |

TABLE 1: Frequency of the preprocessed FMTOD data set classes grouped by partitions; Σ signifies the cumulative frequency statistic

2. Next, we remove data duplicates within the provided training, validation and test data partitions.

3. Finally, we remove data duplicates which overlap between partitions. During this step, we do not remove any cross-partition duplicates from the test partition in order to keep it as similar as possible to the original test partition. This comes into importance later when we compare performance metrics on the test set with other studies.

During the preprocessing phase, many data duplicates were encountered and correspondingly removed. Some of these duplicates observed were already present in the original FMTOD data set, with additional duplicates being created from the initial lowercasing step. After preprocessing, we obtain a lowercased variant of the

| Class and description | Utterance length† | Example‡ |
|---|---|---|
| 0: `alarm/cancel_alarm` | $5.6 \pm 1.9$ | cancel weekly alarm |
| 1: `alarm/modify_alarm` | $7.1 \pm 2.5$ | change alarm time |
| 2: `alarm/set_alarm` | $7.5 \pm 2.5$ | please set the new alarm |
| 3: `alarm/show_alarms` | $6.9 \pm 2.2$ | check my alarms. |
| 4: `alarm/snooze_alarm` | $6.1 \pm 2.1$ | pause alarm please |
| 5: `alarm/time_left_on_alarm` | $8.6 \pm 2.1$ | minutes left on my alarm |
| 6: `reminder/cancel_reminder` | $6.6 \pm 2.2$ | clear all reminders. |
| 7: `reminder/set_reminder` | $8.9 \pm 2.5$ | birthday reminders |
| 8: `reminder/show_reminders` | $6.8 \pm 2.2$ | list all reminders |
| 9: `weather/check_sunrise` | $6.7 \pm 1.7$ | when is sunrise |
| 10: `weather/check_sunset` | $6.7 \pm 1.7$ | when is dusk |
| 11: `weather/find` | $7.8 \pm 2.3$ | jacket needed? |
| $\mu$ | $7.7 \pm 2.5$ | — |

†Summary statistics follow the mean $\pm$ standard-deviation format
‡Short and simple examples were chosen for brevity and formatting purposes

TABLE 2: Utterance length summary statistics and examples for the preprocessed FMTOD data set; $\mu$ signifies the cumulative summary statistic

| Study | Summary | Accuracy |
|---|---|---|
| Schuster et al. (2019) | BiLSTM jointly trained on both the slot filling and intent classification tasks | 99.1% |
| Zhang et al. (2019) | BERT along with various decoders jointly fine-tuned on both the slot filling and intent classification tasks | 96.6–98.9% |
| Zhang, Lyu, and Callison-Burch (2020) | RoBERTa and XLM-RoBERTa fine-tuned on the English language and multilingual intent classification tasks along with WikiHow pre-training | 99.3–99.5% |

TABLE 3: Studies that addressed the FMTOD English language intent classification task along with their relevant summaries and accuracy ranges

FMTOD data set with strictly unique data partitions. In the next section, we describe the summary statistics of the preprocessed FMTOD data set.

### 3.1.2 Summary statistics

In regards to summary statistics of the preprocessed FMTOD data set, Figure 8 provides a visualization of the frequency distribution in the data set grouped by classes and partitions; while Table 1 shows the same summary statistics in a tabular form with explicit frequencies. Based on the summary statistics, we can observe that the preprocessed FMTOD data set is significantly imbalanced with $\sim$45% of samples falling into Class 11 alone. We take this observation into consideration in later sections and apply fixes to mitigate this data imbalance. In addition, we observe

from Table 2 that input utterances in the preprocessed FMTOD data set are generally short; with a mean input utterance length of 7.7 and a standard deviation of 2.5 tokens. Utterance length summary statistics were computed with the assistance of NLTK's default `Treebank` word tokenizer (Bird and Loper, 2004).

### 3.1.3 Performance range

Several recent studies have addressed the FMTOD English language intent classification task using a variety of deep neural networks such as BiLSTMs and large language models such as XLM-RoBERTa (Schuster et al., 2019; Zhang et al., 2019; Zhang, Lyu, and Callison-Burch, 2020). Table 3 summarizes these studies along with their reported accuracy scores on the FMTOD English language intent classification task. Based on the presented results, we can observe that the competitive accuracy range for the FMTOD English language intent classification task is 96.6-99.5%.

## 3.2 SoPa++

We now introduce the main contribution of this thesis: the SoPa++ model. Etymologically, SoPa++ derives its name from the variable increment operator "++" used in programming languages such as C and Java. In essence, the name SoPa++ signifies an improvement or major modification to the SoPa model. Some of the major modifications from SoPa to SoPa++ include the utility of strict linear-chain WFA-$\omega$'s over linear-chain WFAs, replacement of the MLP in SoPa with quantized and transparent hidden layers and the introduction of a new explanations by simplification post-hoc explainability technique which leverages on the aforementioned modifications in SoPa++'s neural architecture.

### 3.2.1 Strict linear-chain WFA-$\omega$'s

As mentioned in Section 2.4, Schwartz, Thomson, and Smith (2018) constructed the SoPa model with an ensemble of linear-chain WFAs which permitted both $\epsilon$ and self-loop transitions. As noted in Section 2.4.3, $\epsilon$ and self-loop transitions are useful constructs in abstracting WFAs and allowing them to consume variable length strings. However, based on experimentation during our development phase; we observed a key concern that the highest scoring substrings in the linear-chain WFAs in SoPa tended to have a large variation of string lengths due to the effect of both $\epsilon$-transitions and self-loops. We believe that this reduces the impact of SoPa's explainability techniques due to a lack of consistency in the lengths of highest scoring paths and substrings. As a result, the first change we decided for was to remove both $\epsilon$ and self-loop transitions in constituent WFAs or patterns. With this change, we could at least ensure that each WFA would always consume strings of fixed lengths.

However, consuming strings of fixed lengths could also be seen as a form of overfitting in the model; since a model could simply memorize short strings or phrases and would not necessarily incorporate any form of generalization. To address this concern, we include a wildcard transition which we define here as a $\omega$-transition. Allowing for such a transition was only natural since wildcards are already crucial parts of regular expressions; which as we mentioned are equivalent to FAs. To provide formalisms for this modification, we provide the following definition for a WFA-$\omega$.

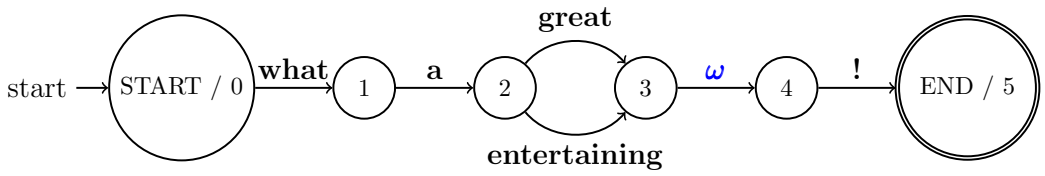


FIGURE 9: Visualization of a strict linear-chain NFA with $\omega$ (blue) and main-path (black) transitions

**Definition 14** (Weighted finite-state automaton-$\omega$)**.** A weighted finite-state automaton-$\omega$ over a semiring $\mathbb{K}$ is a 5-tuple $\mathcal{A} = \langle \Sigma, \mathcal{Q}, \boldsymbol{\Gamma}, \boldsymbol{\lambda}, \boldsymbol{\rho} \rangle$, with:

- a finite input alphabet $\Sigma$;
- a finite state set $\mathcal{Q}$;
- transition matrix $\boldsymbol{\Gamma} : \mathcal{Q} \times \mathcal{Q} \times (\Sigma \cup \{\omega\}) \to \mathbb{K}$;
- initial vector $\boldsymbol{\lambda} : \mathcal{Q} \to \mathbb{K}$;
- and final vector $\boldsymbol{\rho} : \mathcal{Q} \to \mathbb{K}$.

*Remark* 14.1. An $\omega$ transition is equivalent to a wildcard transition, which consumes an arbitrary token input and moves to the next state

*Remark* 14.2. Besides the inclusion of the $\omega$-transition and removal of the $\epsilon$-transition, a WFA-$\omega$ has all of the same characteristics as the WFA defined in Definition 11.

Comparing with the linear-chain WFAs used in Schwartz, Thomson, and Smith (2018) as per Section 2.4.1, our *strict* linear-chain WFA-$\omega$ is similarly alloted a sequence of $|\mathcal{Q}|$ states. However, each state $i$ in the linear-chain WFA-$\omega$ only has two possible outgoing transitions; namely a **$\omega$-transition** which consumes an arbitrary input token and transitions to state $i+1$ and a **main-path transition** which consumes a specific token and transitions to state $i+1$. Because of the elimination of self-loop transitions, we refer to our linear-chain WFA-$\omega$ as strict linear-chain WFA-$\omega$ as per Remark 9.1. Similar to Schwartz, Thomson, and Smith (2018), we utilize only the max-sum and max-product semirings in our strict linear-chain WFA-$\omega$'s.

Next, we provide a mathematical formulation of the modified transition matrix $\boldsymbol{\Gamma}$ in our strict linear-chain WFA-$\omega$. Here, $\boldsymbol{\Gamma}(x)$ represents a $|Q| \times |Q|$ matrix containing transition scores when consuming an input token $x$. $[\boldsymbol{\Gamma}(x)]_{i,j}$ corresponds to the cell value in $\boldsymbol{\Gamma}(x)$ for row $i$ and column $j$ and represents the transition score when consuming token $x$ and transitioning from state $i$ to $j$.

$$[\boldsymbol{\Gamma}(x)]_{i,j} = \begin{cases} \boldsymbol{w}_i \cdot \boldsymbol{v}_x + b_i & \text{if } j = i+1 \text{ (main-path transition)}, \\ \bar{0} & \text{otherwise.} \end{cases} \tag{8}$$

Here, $\boldsymbol{w}_i$ and $b_i$ are learnable vectors and scalar biases parameterizing transitions out of state $i$ to state $i+1$. $\boldsymbol{v}_x$ represents the word embedding for token $x$ and $\bar{0}$ represents the zero value in the semiring used as per Definition 10. Similarly, $\omega$-transitions are parameterized with the following representation in $\boldsymbol{\Gamma}$:

$$[\boldsymbol{\Gamma}(\omega)]_{i,j} = \begin{cases} c_i & \text{if } j = i+1 \text{ ($\omega$-transition)} \\ \bar{0} & \text{otherwise.} \end{cases} \tag{9}$$

Here, $c_i$ represents a learnable scalar bias for $\omega$-transitions out of state $i$ to state $i+1$. Finally as per Schwartz, Thomson, and Smith (2018), we fix the initial vector

**Algorithm 1** Strict linear-chain WFA-$\omega$ document score*

**Input(s):** Strict linear-chain WFA-$\omega$ (denoted as $\mathcal{A}$) and document $\boldsymbol{y}$
**Output(s):** Document score $s_{\text{doc}}(\boldsymbol{y})$

**function** DOCSCORE($\mathcal{A}, \boldsymbol{y}$)
  $\boldsymbol{h}_0 \leftarrow [\bar{1}, -\infty, \ldots, -\infty]$ ▷ Create initial hidden state vector $\boldsymbol{h}_0 : \mathcal{Q} \to \mathbb{K}$
  **for** $i \leftarrow 1, 2, \ldots, |\boldsymbol{y}|$ **do** ▷ Sequentially iterate over each token $y_i \in \boldsymbol{y}$
    $\boldsymbol{m} \leftarrow \left[[\boldsymbol{\Gamma}(y_i)]_{1,2}, [\boldsymbol{\Gamma}(y_i)]_{2,3}, \ldots, [\boldsymbol{\Gamma}(y_i)]_{|\mathcal{Q}|-1,|\mathcal{Q}|}\right]$ ▷ Main-path scores for token $y_i$
    $\boldsymbol{\omega} \leftarrow \left[[\boldsymbol{\Gamma}(\omega)]_{1,2}, [\boldsymbol{\Gamma}(\omega)]_{2,3}, \ldots, [\boldsymbol{\Gamma}(\omega)]_{|\mathcal{Q}|-1,|\mathcal{Q}|}\right]$ ▷ State-wise wildcard scores
    $\boldsymbol{m}' \leftarrow \boldsymbol{m} \otimes \boldsymbol{h}_{i-1}[:-1]$ ▷ Path score with main-path transitions†
    $\boldsymbol{\omega}' \leftarrow \boldsymbol{\omega} \otimes \boldsymbol{h}_{i-1}[:-1]$ ▷ Path score with $\omega$ transitions†
    $\boldsymbol{h}_i \leftarrow [\bar{1}] \,\|\, \max(\boldsymbol{m}', \boldsymbol{w}')$ ▷ Concatenate $\bar{1}$ to maximum of $\boldsymbol{m}'$ and $\boldsymbol{\omega}'$
  **end for**
  $s_{\text{doc}}(\boldsymbol{y}) \leftarrow \max_{i \in 1,2,\ldots,|\boldsymbol{y}|} \boldsymbol{h}_i[-1]$ ▷ Get maximum of hidden vector final states‡
  **return** $s_{\text{doc}}(\boldsymbol{y})$
**end function**

*$\otimes$ and $\bar{1}$ are derived from max-based semirings where all semiring operations are element-wise
†$\boldsymbol{h}_i[:-1]$ follows the Python indexing syntax and implies keeping all elements of $\boldsymbol{h}_i$ except the last
‡$\boldsymbol{h}_i[-1]$ follows the Python indexing syntax and implies retrieving the last element of the vector $\boldsymbol{h}_i$

$\boldsymbol{\lambda} = [\bar{1}, \bar{0}, \ldots, \bar{0}]$ and the final vector $\boldsymbol{\rho} = [\bar{0}, \bar{0}, \ldots, \bar{1}]$, where $\bar{1}$ and $\bar{0}$ represent the one and zero values specified in the semiring as per Definition 10. The time-complexity of the Viterbi algorithm to compute the string score for our linear-chain WFA-$\omega$'s is $O(|Q||\boldsymbol{x}|)$, where $|\mathcal{Q}|$ refers to the number of states and $|\boldsymbol{x}|$ refers to the length of the input string.

Ultimately, the introduction of a strict linear-chain WFA-$\omega$ allows us to attain fixed string length consumption with an added layer of generalization because of the introduction of wildcards. An example of a strict linear-chain NFA extracted from a strict linear-chain WFA-$\omega$ is shown in Figure 9. Interestingly, we can observe that this NFA corresponds to the Perl-compatible regular expression ‘`what a (great|entertaining) [^\s]+ !`’, where `[^\s]+` refers to any set of consecutive characters which are not separated by a space character. We can therefore infer that a $\omega$-transition in a FA corresponds to the `[^\s]+` regular expression term.

### 3.2.2 Document score

Similar to Schwartz, Thomson, and Smith (2018), SoPa++ was intended to compute scores for entire documents and not just fixed-length strings using the strict linear-chain WFA-$\omega$. To achieve this, we propose Algorithm 1 to compute the document score $s_{\text{doc}}(\boldsymbol{y})$ for an arbitrary document $\boldsymbol{y}$. Here, we score all consecutive substrings in a document $\boldsymbol{y}$ using either the max-sum or max-product semirings assisted with the Viterbi algorithm. Following from this, the document score $s_{\text{doc}}(\boldsymbol{y})$ for an arbitrary document $\boldsymbol{y}$ would reflect the highest path score which corresponds to a substring in document $\boldsymbol{y}$.

### 3.2.3 TauSTE

In Section 2.2, we described the concept of a STE in quantized neural networks and explained how STEs function in both their forward and backward passes. Furthermore, we provided a motivation as to why STEs and other quantized activation functions are of interest; for example in relation to computational savings linked to

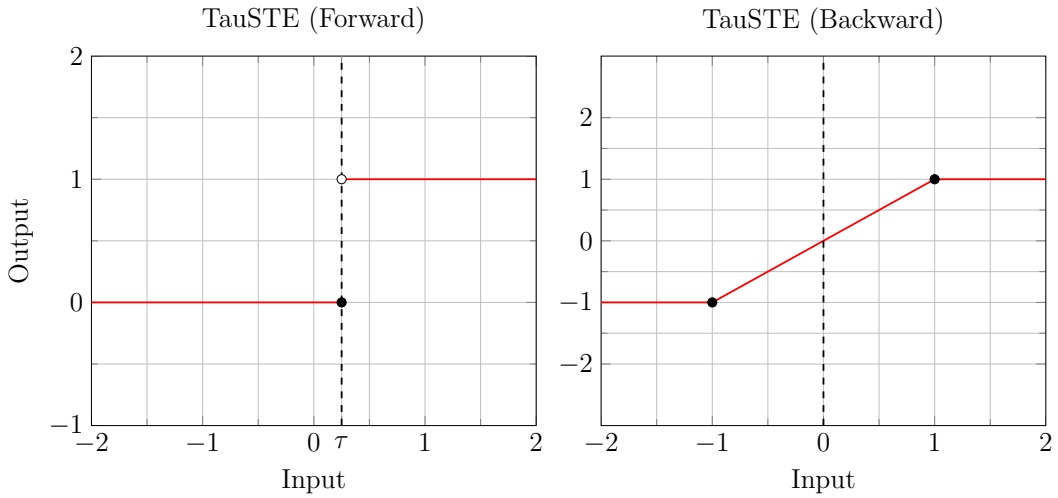


FIGURE 10: Visualization of the TauSTE's forward and backward passes

low-precision computing. In SoPa++, we make use of a variant of the STE activation function which we define here as the Tau Straight-through Estimator (TauSTE).

$$\text{TauSTE}(x) = \begin{cases} 1 & x \in (\tau, +\infty) \\ 0 & x \in (-\infty, \tau] \end{cases} \tag{10}$$

$$\text{TauSTE}'(x) = \begin{cases} 1 & x \in (1, +\infty) \\ x & x \in [-1, 1] \\ -1 & x \in (-\infty, -1) \end{cases} \tag{11}$$

A visualization of the TauSTE's forward and backward passes is shown in Figure 10. As we can see, there are two key changes from the vanilla STE to the TauSTE. Firstly, the threshold for activation in the forward function is now governed by a $\tau$-threshold such that $\tau \in \mathbb{R}$. This was done to allow for some degree of freedom in deciding the activation threshold. Secondly, the backward pass returns the identity function on $x \in [-1, 1]$ and remains fixed at -1 or +1 beyond these limits. This restriction was placed to ensure that gradients do not blow up in size.

We chose to use the TauSTE because we believed it could prove useful for the explainability purposes of our SoPa++ model. This is mainly because the TauSTE (or even the STE) activation function simplifies continuous inputs into discrete outputs; thereby reducing the information content of the signal it receives. In later sections, we show how we capitalize on this reduction in signal information in our SoPa++ model for our explanations by simplification post-hoc explainability technique.

### 3.2.4 Computational graph

With the core modifications in the SoPa++ model explained, we now shift to describing the computational graph or forward pass of the SoPa++ model by referring to its various neural components. This description is linked to the visualization of the computational graph of SoPa++ in Figure 11. Firstly, we utilize NLTK's default `Treebank` word tokenizer (Bird and Loper, 2004) to conduct tokenization of input utterances into word-level tokens. Next, we pad input utterances with special `[START]` and `[END]` tokens at the start and end indices of the utterances to signify the location where the utterance begins and ends. Finally, we utilize GloVe 6B 300-dimensional

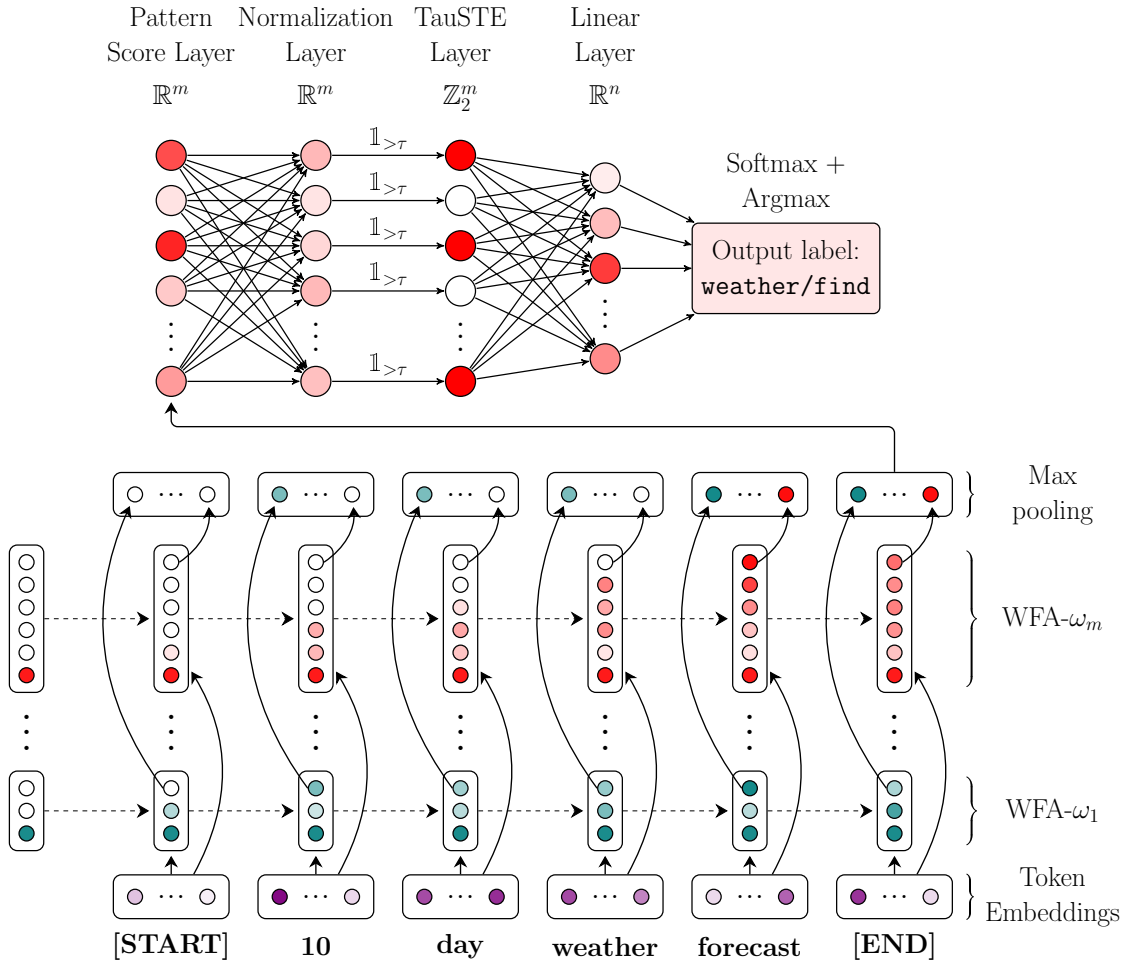


FIGURE 11: Visualization of the SoPa++ computational graph

uncased word-level embeddings (Pennington, Socher, and Manning, 2014) to project the input tokens to continuous numerical spaces.

Following this, we use our ensemble of $m \in \mathbb{N}$ strict linear-chain WFA-$\omega$'s to traverse the input utterance and provide individual document scores for this utterance; as prescribed by Algorithm 1. When processing each input token, we monitor the final states of each of the $m$ strict linear-chain WFA-$\omega$'s and max-pool the score present in this state. In the edge case that an input string was too short for the final state of a WFA-$\omega$ to register a non-$\bar{0}$ score, we simply discard this score in further analysis. This description so far corresponds to the lower half of Figure 11. After max-pooling scores from all the $m$ strict linear-chain WFA-$\omega$'s, we then pass this collection of pattern scores for further processing using SoPa++'s hidden neural components; which can be seen in the upper portion of Figure 11. Firstly, we apply layer normalization (Ba, Kiros, and Hinton, 2016) to all pattern scores without any additional affine transformations. We omit the affine transformation to not alter any of the pattern score information content and use layer normalization as an expedient means of projecting the values of pattern scores to a standard normal distribution. This normalization process guarantees that pattern scores would be small in size and have a roughly even distribution of positive and negative values around 0.

Layer normalization becomes very useful as we correspondingly encounter the TauSTE layer. Here, the TauSTE layer maps all inputs which are strictly larger than the $\tau \in \mathbb{R}$ threshsold to 1 and all others inputs to zero. Naturally, the TauSTE layer

is only useful when it is able to discriminate the inputs by mapping some of them to 1 and some to 0, instead of always mapping all inputs to either 1 or 0. Without layer normalization, the TauSTE layer would not be able to perform its function since pattern scores tend to be mostly positive with differing ranges. Layer normalization therefore helps to project these variations of pattern scores to a uniform range; which ultimately allows the TauSTE layer to discriminate inputs and produce diverse binary outputs.

A natural criticism of the TauSTE layer could be that it strongly limits the flow of information in the SoPa++ model. While the binarization present in this layer does indeed limit the rich flow of continuous numerical information, it is still worth noting that this layer can preserve significant information given a sufficiently large $m$ value, corresponding to the number of strict linear-chain WFA-$\omega$ and TauSTE neurons. For example, if we allow for $m = 40$ and therefore provide 40 WFA-$\omega$ and TauSTE neurons, we can have a total of $2^{40} \approx 1.1 \times 10^{12}$ binary state possibilities; which is slightly greater than one trillion possible binary states. To provide some context to this order of magnitude, the aforementioned number of possibilities is roughly equal to estimated number of stars present in the Andromeda Galaxy (Peñarrubia et al., 2014). This would imply that despite the reduction in information content on the TauSTE layer, there are still sufficient mappable states available to learn various representations relevant to output classes; given a large enough $m$ value or number of WFA-$\omega$.

After binarizing the hidden values in the TauSTE layer, we apply a simple linear transformation to the TauSTE binary outputs to modify their dimensionality from $m$ to $n \in \mathbb{N}$, where the $n$ represents the number of output classes. We specifically chose a linear transformation layer over a MLP because linear regressors are known to be transparent models (Arrieta et al., 2020) and this is a feature which ultimately assists us in our explanations by simplification post-hoc explainability technique, which we describe in greater detail in the next sections. Finally, we apply a softmax function over the linear outputs to project them to probabalistic spaces and then compute an argmax to extract the highest scoring index which represents the predicted class. In the case of Figure 11, the output class for the input preprcessed utterance ‘‘`[START] 10 day weather forecast [END]`’’ is the `weather/find` class which corresponds to class index 11.

### 3.2.5 Transparency

In regards to transparency, SoPa++ is a hybridized model consisting of RNN, CNN and weighted finite-state neural components similar to that of SoPa. Following the arguments of Arrieta et al. (2020) related to the black-box natures of RNNs and CNNs, we can conclude that SoPa++ would correspondingly fall into the black-box model category. Because of this classification, the SoPa++ model would require post-hoc explainability techniques in order to explain its inner mechanisms.

Reviewing the post-hoc explainability techniques offered by SoPa as per Section 2.4.6, we would opine that a major limitation of these techniques is that they do not fully capitalize on the rich theoretical foundations offered by WFAs; such as their possible conversions to NFA and ultimately regular expressions. In order to address this limitation, we propose a new explanations by simplification post-hoc explainability technique for simplifying a fully trained black-box SoPa++ model into a transparent regular expression proxy model. We describe this new explanations by simplification post-hoc explainability technique in the next section.

**Algorithm 2** Strict linear-chain WFA-$\omega$ path-augmented document score*

**Input(s):** Strict linear-chain WFA-$\omega$ (denoted as $\mathcal{A}$) and document $\boldsymbol{y}$
**Output(s):** Document score $s_{\text{doc}}(\boldsymbol{y})$ and its corresponding path $\pi_{\text{doc}}(\boldsymbol{y})$

**function** DOCSCORE_PATH($\mathcal{A}, \boldsymbol{y}$)
  $\boldsymbol{h}_0 \leftarrow [\bar{1}, -\infty, \ldots, -\infty]$ ▷ Create initial hidden state vector $\boldsymbol{h}_0 : \mathcal{Q} \to \mathbb{K}$
  **for** $i \leftarrow 1, 2, \ldots, |\boldsymbol{y}|$ **do** ▷ Sequentially iterate over each token $y_i \in \boldsymbol{y}$
    $\boldsymbol{m} \leftarrow [[\boldsymbol{\Gamma}(y_i)]_{1,2}, [\boldsymbol{\Gamma}(y_i)]_{2,3}, \ldots, [\boldsymbol{\Gamma}(y_i)]_{|\mathcal{Q}|-1,|\mathcal{Q}|}]$ ▷ Main-path scores for token $y_i$
    $\boldsymbol{\omega} \leftarrow [[\boldsymbol{\Gamma}(\omega)]_{1,2}, [\boldsymbol{\Gamma}(\omega)]_{2,3}, \ldots, [\boldsymbol{\Gamma}(\omega)]_{|\mathcal{Q}|-1,|\mathcal{Q}|}]$ ▷ State-wise wildcard scores
    $\boldsymbol{m}' \leftarrow \boldsymbol{m} \otimes \boldsymbol{h}_{i-1}[:-1]$ ▷ Path score with main-path transitions†
    $\boldsymbol{\omega}' \leftarrow \boldsymbol{\omega} \otimes \boldsymbol{h}_{i-1}[:-1]$ ▷ Path score with $\omega$ transitions†
    $\boldsymbol{h}_i \leftarrow [\bar{1}] \,\|\, \max(\boldsymbol{m}', \boldsymbol{w}')$ ▷ Concatenate $\bar{1}$ to maximum of $\boldsymbol{m}'$ and $\boldsymbol{\omega}'$
    $\pi_i \leftarrow \text{trace}(h_i[-1])$ ▷ Back-trace path $\pi_i$ corresponding to $h_i[-1]$‡
  **end for**
  $j \leftarrow \arg\max_{i \in 1,2,\ldots,|\boldsymbol{y}|} \boldsymbol{h}_i[-1]$ ▷ Get index of final states' maximum‡
  $s_{\text{doc}}(\boldsymbol{y}) \leftarrow \boldsymbol{h}_j[-1]$ ▷ Get maximum final state value‡
  $\pi_{\text{doc}}(\boldsymbol{y}) \leftarrow \pi_j$ ▷ Get path of maximum final state value
  **return** $[s_{\text{doc}}(\boldsymbol{y}), \pi_{\text{doc}}(\boldsymbol{y})]$
**end function**

*$\otimes$ and $\bar{1}$ are derived from max-based semirings where all semiring operations are element-wise
†$\boldsymbol{h}_i[:-1]$ follows the Python indexing syntax and implies keeping all elements of $\boldsymbol{h}_i$ except the last
‡$\boldsymbol{h}_i[-1]$ follows the Python indexing syntax and implies retrieving the last element of the vector $\boldsymbol{h}_i$

## 3.3 RE proxy

In this section, we describe the key processes required to convert a fully trained black-box SoPa++ model into a transparent RE proxy model. These include the introduction of a path-augmented document scoring algorithm and the creation of the RE lookup layer. Next, we describe the computational graph or forward pass of the RE proxy model. Finally, we provide some comments on explainability-related aspects of the simplified RE proxy model and its antecedent SoPa++ counterpart.

### 3.3.1 Path-augmented document score

One major advantage of the Viterbi algorithm, as per Definition 13, is its ability to return the highest path score which can ultimately allow for attribution to a certain path and substring in a document. In order to simplify SoPa++ into a RE proxy model, we first need to modify our document scoring algorithm to not only return the document score $s_{\text{doc}}(\boldsymbol{y})$ but also its corresponding path $\pi_{\text{doc}}(\boldsymbol{y})$. This process is described as a path-augmented document score in Algorithm 2, where we trace the exact path of each transition and return the best path in addition to the highest path score. Similar to Algorithm 1, this algorithm has a time-complexity of $O(|Q||\boldsymbol{x}|)$, where $|\mathcal{Q}|$ refers to the number of states in a strict linear-chain WFA-$\omega$ and $|\boldsymbol{x}|$ refers to the length of the input string. It is also worth mentioning that the highest scoring path returned ultimately reflects a set of transitions in the form of a strict linear-chain NFA; which can correspondingly be transformed into a regular expression. Therefore, it can be inferred that Algorithm 2 returns the best scoring regular expression corresponding to a certain strict linear-chain WFA-$\omega$.

**Algorithm 3** Extracting RE lookup layer from SoPa++

**Input(s):** Trained SoPa++ model $\mathcal{S}$ and training data $[\boldsymbol{y}_1, \ldots, \boldsymbol{y}_t]$
**Output(s):** RE lookup layer $[\{\mathbf{RE}\}_1, \ldots, \{\mathbf{RE}\}_m]$

**function** EXTRACT_LOOKUP($\mathcal{S}, [\boldsymbol{y}_1, \ldots, \boldsymbol{y}_t]$)
  $[\{\mathbf{RE}\}_1, \ldots, \{\mathbf{RE}\}_m] \leftarrow [\emptyset, \ldots, \emptyset]$ ▷ Initialize RE lookup layer
  **for** $i \leftarrow 1, 2, \ldots, t$ **do** ▷ Loop over training data
    **for** $j \leftarrow 1, 2, \ldots, m$ **do** ▷ Loop over WFA-$\omega$'s in $\mathcal{S}$
      $\mathcal{A}_j \leftarrow$ get_WFA($\mathcal{S}, j$) ▷ Get WFA-$\omega$ by index
      $s^j_{\text{doc}}(\boldsymbol{y}_i), \pi^j_{\text{doc}}(\boldsymbol{y}_i) \leftarrow$ DOCSCORE_PATH($\mathcal{A}_j, \boldsymbol{y}_i$) ▷ As per Algorithm 2
      **if** TauSTE($s^j_{\text{doc}}(\boldsymbol{y}_i)$) == 1 **then**
        $\{\mathbf{RE}\}_j \leftarrow \{\mathbf{RE}\}_j \cup \pi^j_{\text{doc}}(\boldsymbol{y}_i)$ ▷ Save $\pi^j_{\text{doc}}(\boldsymbol{y}_i)$ if it activates TauSTE
      **end if**
    **end for**
  **end for**
  **return** compress($[\{\mathbf{RE}\}_1, \ldots, \{\mathbf{RE}\}_m]$) ▷ Compress RE lookup layer
**end function**

### 3.3.2 RE lookup layer

The next step in simplifying a fully-trained SoPa++ to a RE proxy model is the extraction of a RE lookup layer. To motivate this process, we shortly revert to the SoPa++ computational graph in Figure 11. The TauSTE layer filters input signals from normalized pattern scores and provides +1 outputs for normalized pattern scores which exceed the $\tau$-threshold. Since pattern scores correspond to document scores and document scores can be augmented with best paths and regular expressions as per Algorithm 2, we can assign the activation of each TauSTE neuron for each input utterance with an "activating" regular expression. By iterating over all our training data, we can collect many such "activating" regular expressions which activate the various TauSTE neurons; such that each TauSTE neuron $\mathbf{N}_i$ is assigned a set of "activating" regular expressions $\{\mathbf{RE}\}_i$. The collection of all regular expressions $[\{\mathbf{RE}\}_1, \ldots, \{\mathbf{RE}\}_m]$ which activate $m$ TauSTE neurons is known as the RE lookup layer. Overall, the RE lookup layer represents a knowledge base of important regular expressions that cause the SoPa++ model to make weighted decisions. This process of extracting the RE lookup layer from the SoPa++ model is reflected in Algorithm 3.

This leads us to several interesting conclusions. Firstly, we observe here that the TauSTE layer is not only useful for reducing information complexity; but also for attributing causal links to SoPa++'s decision-making. In this case, the causal links are the "activating" regular expressions returned by the strict-linear chain WFA-$\omega$ when computing the path-augmented document score. Next, we can also observe the effect of the $\tau$-threshold on the RE lookup layer. If the $\tau$-threshold is low, we effectively allow more TauSTE neurons to be activated and therefore allow more regular expressions from the strict linear-chain WFA-$\omega$ to be saved in the RE lookup layer. Contrastingly, if the $\tau$-threshold is high; we allow fewer TauSTE neurons to be activated and therefore allow fewer regular expressions to be saved in the RE lookup layer. It is not necessarily clear whether small or large $\tau$-thresholds are better for performance; but a larger $\tau$-threshold and therefore smaller RE lookup layer could be beneficial for explainability purposes since the RE knowledge base would be smaller and possbily easier to comprehend for a human.

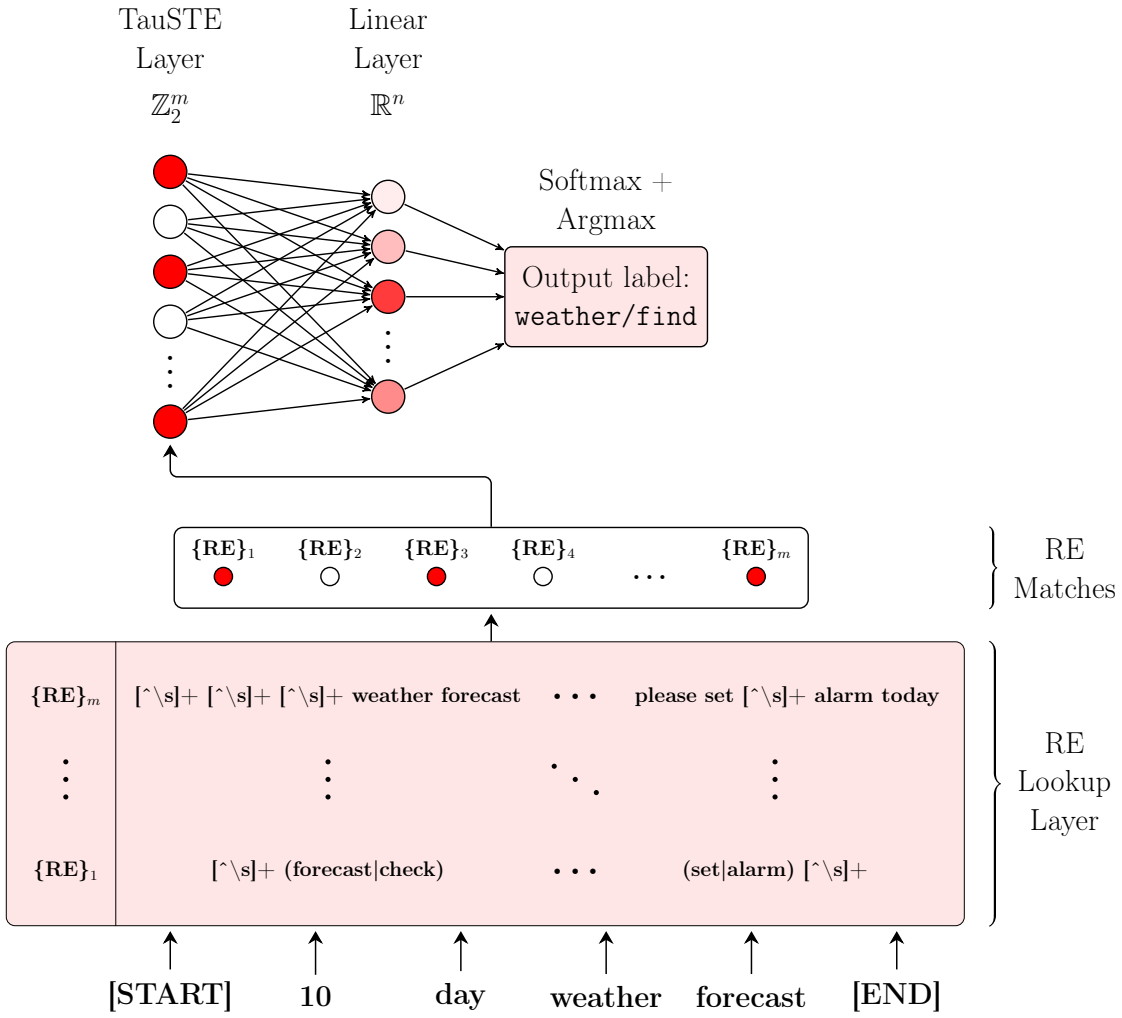


FIGURE 12: Visualization of the RE proxy computational graph

### 3.3.3 Assembling RE proxy

The final step in simplifying a fully-trained SoPa++ model into a RE proxy model is to assemble the RE proxy model using the RE lookup and SoPa++ linear layers. Specifically, for a given SoPa++ model; we firstly extract the RE lookup layer. Then, we combine the RE lookup layer and the SoPa++ linear layer in order to create the RE proxy model. A visualization of this is shown in Figure 12 and we can observe how the RE lookup layer essentially replaces most of the lower neural components of the SoPa++ model up until the TauSTE layer. The resulting RE proxy should ideally be a good approximator of the SoPa++ model; with the exact degree of approximation being reserved for empirical investigation. Given this process of assembly, it is important to note that each SoPa++ model allows for exactly one RE proxy model that can be assembled. Therefore, SoPa++ and RE proxy models occur in model-pairs.

One major limitation of the RE proxy model is in its RE lookup layer. Since the RE lookup layer is essentially a memorized knowledge base, it could contribute to overfitting on the training data and result in a RE proxy model that is not representative of the SoPa++ model on unseen data. While this limitation could theoretically be offset by the presence of sufficient variable-length regular expression samples with wildcards, it would ultimately boil down to an empirical investigation to quantify how similarly the RE proxy model performs compared to its antecedent SoPa++

model on previously unseen data.

### 3.3.4 Computational graph

We now provide a description of the computational graph or forward pass of the assembled RE proxy model with reference to Figure 12. Similar to the computational graph of SoPa++, we process our input text sequentially. However, instead of using token embeddings and neural network components; we pass our input utterance through our RE lookup layer which conducts substring matches over all sets of regular expressions in $[\{\mathbf{RE}\}_1, \ldots, \{\mathbf{RE}\}_m]$. If any regular expression set $\{\mathbf{RE}\}_i$ matches the input utterance, the index of this RE set if given a value of 1. If no regular expression in the set matches, the index of this RE set is given a value of 0. These values are assembled into a binary vector that mimics the outputs of the TauSTE layer in the antecedent SoPa++ model. This binary vector is then passed to the linear layer; which transforms the binary vector back into continuous space with the appropriate dimensionality. Following softmax and argmax operations, we arrive at our predicted output class. It is worth mentioning that while the execution of the SoPa++ forward pass can be accelerated due to tensor-based parallelization with hardware acceleration on a Graphics Processing Unit (GPU), the forward pass of the RE proxy model is significantly slower since we utilized an unoptimized single-threaded lookup process to match regular expressions in the RE lookup layer.

### 3.3.5 Transparency

So far, we have described the process of simplifying a fully-trained black-box SoPa++ model into a RE proxy model. We now investigate the RE proxy model and comment on its degree of transparency. In essence, a RE proxy model consists of a RE lookup layer followed by a linear regressor. The RE lookup layer can be seen as a rules-based learner component where inputs are processed via clear and interpretable RE matching rules to produce outputs. Since both rules-based learners and linear regressors can be considered as transparent models as per Arrieta et al. (2020, Page 7, Section 3), we could theoretically classify the RE proxy model as a transparent model. However, it is important to note that this is only a theoretical argument which could be disproven in given practical cases. For example as per Arrieta et al. (2020, Page 9, Table 2), rules-based learners and linear regressors could be viewed as black-box models if they require the handling of an incomprehensible number of rules or input features. This could also be the case for the RE proxy model depending most importantly on the size of the RE lookup layer.

### 3.3.6 Explainability

We now provide additional comments on the explanations by simplification post-hoc explainability technique to simplify a fully-trained black-box SoPa++ model into a transparent RE proxy model. Firstly, we would assign the target audience of this explainability technique as expert-users compared to the inferred target audience of average end-users for the SoPa model. We designate expert-users as our target audience mainly because the process of constructing and understanding the RE proxy model is not as straightforward as the local explanations and feature relevance techniques in SoPa as per Section 2.4.6.

| Characteristic | SoPa | SoPa++ |
|---|---|---|
| Text casing | True-cased | Lower-cased |
| Token embeddings | GloVe 840B 300-dimensions | GloVe 6B 300-dimensions |
| WFAs | Linear-chain WFAs with $\epsilon$, self-loop and main-path transitions | Strict linear-chain WFA-$\omega$'s with $\omega$ and main-path transitions |
| Hidden layers | Multi-layer perceptron after max-pooling | Layer normalization, TauSTE and linear transformation after max-pooling |
| Post-hoc explainability technique(s) | Local explanations, feature relevance | Explanations by simplification |

TABLE 4: Summarized differences for SoPa vs. SoPa++

Next, we evaluate the quality of explanations offered by using the explanations by simplification technique using the three guidelines offered in Section 2.1.6. Regarding the constrictive quality, it is likely that the RE proxy model meets this criterion since the linear layer contains easily interpretable weights which could explain which matched regular expressions were scored higher or lower. Regarding causal links, it is likely that the RE proxy model meets this criterion since the RE lookup layer matches fixed REs whose binary matching scores are then clearly propagated from the start to the end of the model. Therefore, a decision occurring at the end of the model could be causally attributed to features at the start of the model. Regarding the selective quality, it is possible that the RE proxy model meets this criterion since the linear weights applied to matched regular expressions can be easily ranked to understand which were the most important causal links influencing the model's decision.

## 3.4 Differences between SoPa and SoPa++

To wrap up this current segment on the SoPa++ model, we summarize the key differences between SoPa and SoPa++ as shown in Table 4. The most significant changes in SoPa++ include the modification of linear-chain WFAs to strict linear-chain WFA-$\omega$'s, the replacement of the MLP with layer normalization, TauSTE and linear layers and the introduction of the explanations by simplification post-hoc explainability technique to create RE proxy models. With the construction of SoPa++ and its RE proxy models established, we now proceed to describe the methodologies used to answer our three research questions.

## 3.5 RQ1: Evaluating performance of SoPa++

In this section, we describe the methodologies used to answer our first research question regarding the competitive performance of the SoPa++ model on the FMTOD data set. Specifically, we describe how we trained the SoPa++ model and afterwards, how we proceeded to evaluate and compare its performance to other studies.

| Model size | Patterns hyperparameter $P$ | Parameter count |
|---|---|---|
| Small | `6-10_5-10_4-10_3-10` | 1,260,292 |
| Medium | `6-25_5-25_4-25_3-25` | 1,351,612 |
| Large | `6-50_5-50_4-50_3-50` | 1,503,812 |

TABLE 5: Three different SoPa++ model sizes used during training with corresponding patterns hyperparameter $P$ and parameter counts

### 3.5.1 Training

First and foremost, we address the issue of data imbalance in the FMTOD data set as mentioned in Section 3.1.2. For this, we chose a simple but effective solution of upsampling all minority data classes such that they would have the same frequency as the majority class. We chose this approach over other approaches such as gradient-weighting since this is generally a model agnostic and straightforward approach.

Returning to the computational graph of SoPa++ in Figure 11, we compute the cross-entropy loss using the softmax output and target one-hot encoded classes. Since SoPa++ is in essence a deep neural network, we utilize the well-studied gradient descent technique to train and update our SoPa++ model with the objective of minimizing the aforementioned cross-entropy loss. To facilitate this learning process, we utilize PyTorch[2] to assist with gradient computations, backward passes and parallelized tensor computations. We utilize the Adam optimizer (Kingma and Ba, 2015) to stabilize the gradient descent learning technique with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. In terms of fixed training hyperparameters, we utilize a learning rate of 0.001, a batch size of 256 input utterances and neuron/word dropout probabilities of 0.2. We only apply neuron dropouts on the transition matrices of all the strict linear-chain WFA-$\omega$'s in SoPa++. We furthermore apply batch sorting based on input utterance lengths to ensure maximum efficiency when computing pattern scores. Based on our own experiments, we observed more stable training with the max-sum semiring compared to the max-product semiring. For simplicity, we therefore only use the max-sum semiring in all of our WFA-$\omega$'s.

To obtain some variation in our SoPa++ models, we decide to use a grid-search technique while varying the patterns $P$ and $\tau$-threshold hyperparameters. Our variations in the patterns hypereparameter $P$ result in three different SoPa++ model sizes which we define as small, medium and large as per Table 5. Correspondingly, we vary the $\tau$-threshold hyperparameter with the following five possible values: $\{0.00, 0.25, 0.50, 0.75, 1.00\}$. For each model configuration in our grid-search routine, we repeat a model run ten times with different initial random seeds in order to obtain a distribution of performances. In total, our grid-search routine produces a total of $3 \times 5 \times 10 = 150$ model runs. Finally, we train all models for a maximum of 50 epochs with 10 patience epochs for early stopping in case of a performance plateau or worsening. To trigger early stopping in the patience framework, we monitor the cross-entropy loss over the validation data set. We run all SoPa++ training experiments on a single NVIDIA GeForce GTX 1080 Ti GPU for $\sim$24 hours.

### 3.5.2 Evaluation

Given fully trained SoPa++ models from the previous training step, we now proceed to evaluate the performance of the SoPa++ models on the FMTOD data set's

[2]https://pytorch.org/

test partition. We simply run the preprocessed FMTOD test partition through the computational graph of the SoPa++ model and obtain the predicted classes. With the predicted and target classes, we compute the accuracy of the SoPa++ models and summarize these to obtain a distribution of performances over the random seed iterations. Finally, we compare our mean accuracies with the accuracy range of other recent papers on FMTOD as described in Section 3.1.3. If the mean accuracy of our SoPa++ models falls into the aforementioned competitive range, we can then conclude that our SoPa++ model performs competitively with other recent studies.

## 3.6 RQ2: Evaluating explanations by simplification

We now move on to describe the methodologies pursued to answer our second research question on evaluating the effectiveness of our explanations by simplification post-hoc explainability technique. As mentioned in Definition 6, the purpose of explanations by simplification is to create a less complex proxy model which can both keep a similar performance score and maximize its resemblance to the antecedent model. We already discussed how the RE proxy derived from SoPa++ is likely to be a transparent model compared to the black-box SoPa++ model; which already satisfies the first criterion that the proxy model should be less complex than the antecedent model.

To address the next criterion regarding the similarity of the performance scores of both SoPa++ and RE proxy model pairs, we compute and compare the accuracy scores of all model pairs on the FMTOD data set's test partition since this represents previously unseen data for both SoPa++ and RE proxy models. To address the final criterion of maximum resemblance between the antecedent and proxy models, we propose computing the softmax distance norm and binary misalignment rate distance metrics to quantify the distance between the SoPa++ and RE proxy model pairs. Naturally, a smaller distance metric would symbolize greater resemblance between model pairs. We describe these distance metrics with mathematical formalisms in the following sections.

### 3.6.1 Softmax distance norm

The softmax distance norm $\delta_\sigma(\boldsymbol{y})$ refers to the Euclidean norm of the difference in the softmax vectors of the SoPa++ and RE proxy models for a given document $\boldsymbol{y}$, which are represented by $\sigma_{\boldsymbol{\mathcal{S}}}(\boldsymbol{y})$ and $\sigma_{\boldsymbol{\mathcal{R}}}(\boldsymbol{y})$ respectively:

$$\delta_\sigma(\boldsymbol{y}) = \|\sigma_{\boldsymbol{\mathcal{S}}}(\boldsymbol{y}) - \sigma_{\boldsymbol{\mathcal{R}}}(\boldsymbol{y})\|_2 = \sqrt{\sum_{i=1}^{n}(\sigma_{\mathcal{S}_i}(\boldsymbol{y}) - \sigma_{\mathcal{R}_i}(\boldsymbol{y}))^2} \tag{12}$$

In order to measure the central tendency of the softmax distance norm across the FMTOD data set's test partition, we compute this distance metric for all instances and correspondingly compute the mean softmax distance norm; which we denote here as $\overline{\delta_\sigma}$. A low mean softmax distance norm norm would imply that the softmax distributions of SoPa++ and RE proxy models were similar and would imply, to a certain extent, that the models conducted similar weightings of input features. This method is not a perfect indicator of interpreting distances between models; but it is more refined than analyzing discrete classification outputs.

### 3.6.2 Binary misalignment rate

The binary misalignment rate $\delta_b(\boldsymbol{y})$ refers to the dimension normalized Manhattan norm of the difference in the binary TauSTE vectors of the SoPa++ and RE proxy models for a given document $\boldsymbol{y}$, which are represented by $\boldsymbol{b_S}(\boldsymbol{y})$ and $\boldsymbol{b_R}(\boldsymbol{y})$ respectively:

$$\delta_b(\boldsymbol{y}) = \frac{\|\boldsymbol{b_S}(\boldsymbol{y}) - \boldsymbol{b_R}(\boldsymbol{y})\|_1}{dim(\boldsymbol{b_S}(\boldsymbol{y}) - \boldsymbol{b_R}(\boldsymbol{y}))} = \frac{\sum_{i=1}^{n} |b_{\mathcal{S}_i}(\boldsymbol{y}) - b_{\mathcal{R}_i}(\boldsymbol{y})|}{dim(\boldsymbol{b_S}(\boldsymbol{y}) - \boldsymbol{b_R}(\boldsymbol{y}))} \tag{13}$$

Here, the *dim* operator retrieves the dimensionality of a vector space. Normalization of the Manhattan norm with this dimensionality is necessary since we compare models with different sizes which could have different binary vector dimensions. In order to measure the central tendency of the binary misalignment rate across the FMTOD data set's test partition, we compute this distance metric for all instances and correspondingly compute the mean binary misalignment rate; which we denote here as $\overline{\delta_b}$. A low mean binary misalignment rate would imply that the TauSTE binary vector distributions were close together. Therefore, the binary misalignment could be used as another indicator to measure the distance between SoPa++ and RE proxy model pairs.

## 3.7 RQ3: Interesting and relevant explanations

Finally, we arrive at the methodologies used to answer our third research question related to showing interesting and relevant explanations that the SoPa++ and RE proxy models can provide on the FMTOD data set. To approach this research question, we pursue two key strategies of visualizing relative linear weights and sampling REs from the RE lookup layer corresponding to salient TauSTE neurons.

### 3.7.1 Relative linear weights

A major advantage of the linear layer in both SoPa++ and RE proxy models is its interpretability or transparency compared to the MLP in the SoPa model. To visualize the relative linear weights applied to the TauSTE neurons, we apply a softmax operation over weights in the linear layer assigned to each TauSTE neuron and visualize these relative weights to observe how SoPa++ and its RE proxy models distribute feature importance across TauSTE neurons.

### 3.7.2 Samples from RE lookup layer

Based on the aforementioned visualization of relative linear weights, we identify salient TauSTE neurons which receive disproportionately large relative linear weights for the alarm, reminder and weather domains. Correspondingly, we probe the RE lookup layer corresponding to these salient TauSTE neurons and sample ten "activating" regular expressions for each of these salient TauSTE neurons. Theoretically, this could provide us with an insight as to which regular expressions were of particular importance for the classification of the three domains. Furthermore, a deeper analysis of these regular expressions could provide us with insights into possible inductive biases incorporated by the SoPa++ and the RE proxy models.

# Chapter 4

# Results

In this chapter, we focus on summarizing the results of our aforementioned methodologies which we described to answer our three research questions. We break this results chapter into three sections, with each section addressing one research question. As a note, we do not dive deep into interpreting our results but instead use this chapter to solely report on our results. We interpret the results in greater detail in the next Discussion chapter.

## 4.1 RQ1: Evaluating performance of SoPa++

Following our methodologies, we conducted a grid-search training scheme where we varied our patterns $P$ and $\tau$-threshold parameters. The variation in the patterns hyperparameter $P$ resulted in three different SoPa++ model sizes; specifically the small, medium and large model sizes as reported in Table 5. Along with ten random seed iterations for each grid instance in our grid-search scheme, we ran a total of 150 models runs.

We start off by describing the training process for our SoPa++ models. Figure 13 shows a visualization of the validation accuracy against training updates in our grid-search training runs. Firstly, we observe a trend that the larger models tend to have a higher validation accuracy profiles compared to smaller models. Next, we observe that increasing the $\tau$ value from 0.00 to 1.00 tends to decrease the overall validation accuracy profile across all model sizes, albeit more drastically for smaller models than larger models. Finally, we observe that the larger models tend to have an earlier convergence window compared to the smaller models.

In regards to the evaluation of SoPa++ performance, we refer to Table 6 for a tabular summary of test accuracies across the small, medium and large model sizes grouped by the various $\tau$-threshold hyperparameters. Here, we can observe two trends. Firstly, the larger models tend to have a higher mean test accuarcy. Secondly, the mean test accuarcy for all models tends to decrease as the $\tau$-threshold increases. These trends are also consistent with the validation accuracy trends in the previous section. The best performing models for each model size are those where $\tau$=0.00.

## 4.2 RQ2: Evaluating explanations by simplification

Following our methodologies for evaluating explanations by simplification for the SoPa++ model, we compare test set accuarcies and distance metrics between SoPa++ and RE proxy model pairs. Figure 14 and Table 7 provide both a visualization and tabular summary of all the aforementioned comparisons respectively. Overall, we can observe three general trends. Firstly, we observe that SoPa++ and RE proxy

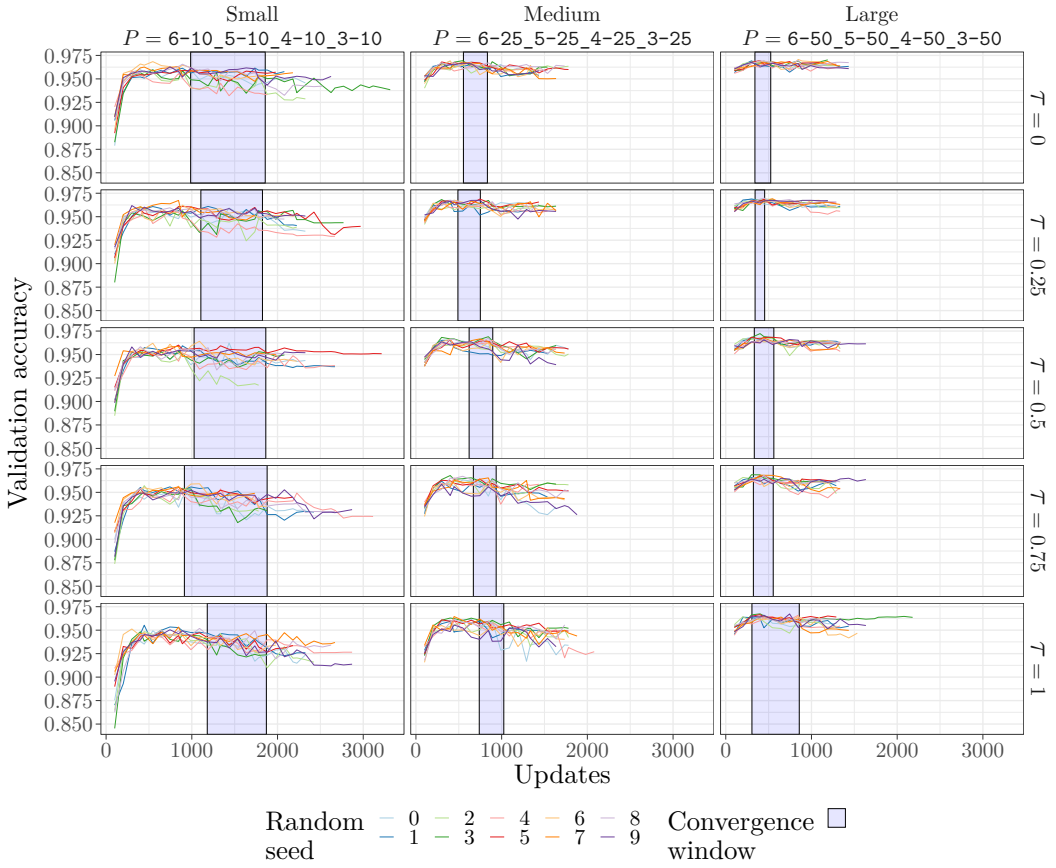


FIGURE 13: Visualization of validation accuracy against number of training updates grouped by patterns $P$ and $\tau$-threshold hyperparameters; the different coloured lines represent random seed iterations with a specific initial random seed

| | | Accuracy in % with mean ± standard-deviation | | | | |
|---|---|---|---|---|---|---|
| Size | Parameters | $\tau$=0.00 | $\tau$=0.25 | $\tau$=0.50 | $\tau$=0.75 | $\tau$=1.00 |
| Small | 1,260,292 | **97.6 ± 0.2** | 97.6 ± 0.2 | 97.3 ± 0.2 | 97.0 ± 0.3 | 96.9 ± 0.3 |
| Medium | 1,351,612 | **98.3 ± 0.2** | 98.1 ± 0.1 | 98.0 ± 0.2 | 97.9 ± 0.1 | 97.7 ± 0.1 |
| Large | 1,503,812 | **98.3 ± 0.2** | 98.3 ± 0.2 | 98.2 ± 0.2 | 98.1 ± 0.2 | 98.0 ± 0.2 |

TABLE 6: Test accuracies of the SoPa++ models grouped by model sizes and $\tau$-threshold hyperparameters; mean accuracies and standard deviations were calculated across random seed iterations; bold scores show best performing model for each model size

test accuracies tend to become converge towards one another as we increase the $\tau$-threshold. Next, we can observe that RE proxy accuracies tend to be lower than their SoPa++ counterparts during this convergence process; with a single exception of the large model at $\tau = 1.00$ where the RE proxy mean accuracy exceeds that of SoPa++. Finally, we observe the general trend that both $\overline{\delta_\sigma}$ and $\overline{\delta_b}$ become closer to zero as we increase the $\tau$-threshold. This trend is consistently observed for $\overline{\delta_b}$ but is not always the case for $\overline{\delta_\sigma}$; which is minimized at $\tau$-threshold values of 0.50-0.75.

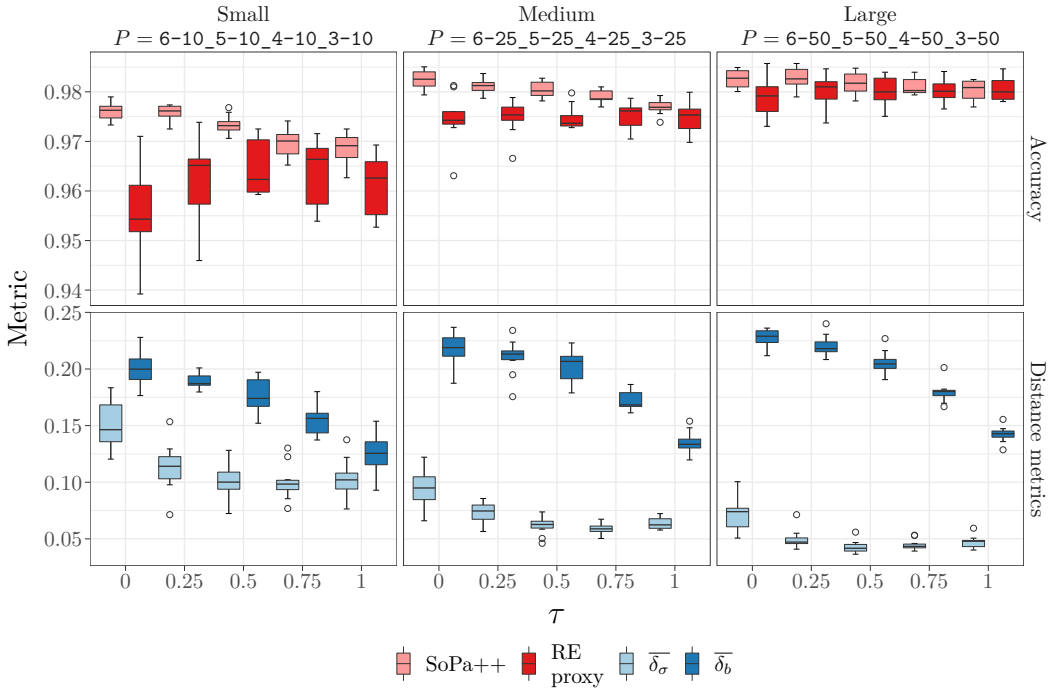


FIGURE 14: Visualization of accuracy and model-pair distance metrics against the $\tau$-threshold hyperparameter grouped by the patterns hyperparameter $P$

| | | Accuracy in % with mean ± standard-deviation | | | | |
|---|---|---|---|---|---|---|
| Size | Model | $\tau$=0.00 | $\tau$=0.25 | $\tau$=0.50 | $\tau$=0.75 | $\tau$=1.00 |
| Small | SoPa++ | 97.6 ± 0.2 | 97.6 ± 0.2 | 97.3 ± 0.2 | **97.0 ± 0.3** | 96.9 ± 0.3 |
| | RE proxy | 95.5 ± 1.0 | 96.2 ± 0.8 | 96.5 ± 0.6 | **96.3 ± 0.7** | 96.1 ± 0.6 |
| Medium | SoPa++ | 98.3 ± 0.2 | 98.1 ± 0.1 | 98.0 ± 0.2 | 97.9 ± 0.1 | **97.7 ± 0.1** |
| | RE proxy | 97.4 ± 0.5 | 97.5 ± 0.3 | 97.5 ± 0.2 | 97.5 ± 0.3 | **97.5 ± 0.3** |
| Large | SoPa++ | 98.3 ± 0.2 | 98.3 ± 0.2 | 98.2 ± 0.2 | 98.1 ± 0.2 | **98.0 ± 0.2** |
| | RE proxy | 97.9 ± 0.4 | 98.0 ± 0.3 | 98.0 ± 0.3 | 98.0 ± 0.2 | **98.1 ± 0.2** |
| | | Metric in % with mean ± standard-deviation | | | | |
| Size | Metric | $\tau$=0.00 | $\tau$=0.25 | $\tau$=0.50 | $\tau$=0.75 | $\tau$=1.00 |
| Small | $\overline{\delta_\sigma}$ | 15.0 ± 2.3 | 11.3 ± 2.2 | 10.0 ± 1.6 | **10.0 ± 1.6** | 10.3 ± 1.8 |
| | $\overline{\delta_b}$ | 20.1 ± 1.5 | 18.9 ± 0.7 | 17.8 ± 1.5 | 15.5 ± 1.3 | **12.4 ± 1.8** |
| Medium | $\overline{\delta_\sigma}$ | 9.5 ± 1.7 | 7.3 ± 0.9 | 6.1 ± 0.8 | **5.8 ± 0.5** | 6.3 ± 0.5 |
| | $\overline{\delta_b}$ | 21.7 ± 1.5 | 21.0 ± 1.6 | 20.3 ± 1.5 | 17.2 ± 0.9 | **13.5 ± 1.0** |
| Large | $\overline{\delta_\sigma}$ | 7.1 ± 1.5 | 5.0 ± 0.8 | **4.3 ± 0.6** | 4.5 ± 0.5 | 4.7 ± 0.5 |
| | $\overline{\delta_b}$ | 22.7 ± 0.9 | 22.0 ± 1.0 | 20.6 ± 1.0 | 18.0 ± 0.9 | **14.2 ± 0.7** |

TABLE 7: Test accuracies and model-pair distances metrics of the SoPa++ and RE proxy models grouped by model sizes and $\tau$-thresholds; accuracies and standard deviations were calculated across random seed iterations; bold test accuracies show model pairs where mean accuracies are closest to one another; bold distance metrics indicate mean metrics that are closest to zero

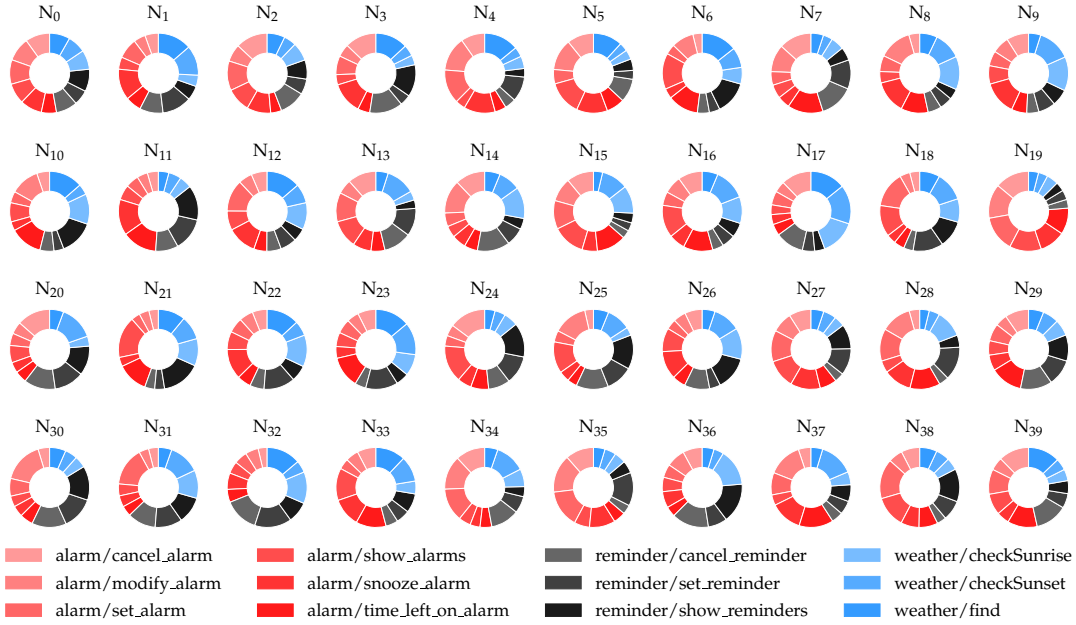


FIGURE 15: Relative linear layer weights applied to TauSTE neurons for the best performing small RE proxy model with a test accuracy of 97.4%

## 4.3 RQ3: Interesting and relevant explanations

Following our methologies to find interesting and relevant explanations on the FMTOD data set, we start off by selecting a particular RE proxy model and analyzing the relative linear weights applied to each TauSTE neuron. For simplicity and to minimize the number of TauSTE neurons to visualize, we select the best performing small RE proxy model with 40 TauSTE neurons and a test set accuracy of 97.4%. Figure 15 visualizes the relative linear weights applied to each TauSTE neuron in this RE proxy model. We can observe that the relative linear weights are generally smoothly and evenly distributed for each TauSTE neuron among all the 12 FMTOD classes. That being said, there exist salient TauSTE neurons on which there are disproportionatey large relative weights placed for certain domains compared to others. Examples of these include neuron 19, 25 and 17 which place disproportionately large relative weights on the alarm, reminder and weather domains respectively.

Next, we isolate the aforementioned salient TauSTE neurons 19, 25 and 17 and we sample ten "activating" regular expressions from the RE lookup layer corresponding to these three neurons. We visualize these regular expressions as strict linear-chain NFA with $\omega$-transitions as shown in Figures 16, 17 and 18 respectively. We can observe that TauSTE neurons 17 and 19 are used for capturing sub-strings of length 3, while TauSTE neuron 25 is used for capturing sub-strings of length 4. Next, we can observe the segmentation of domain related information among the three neurons. Finally, we can observe certain main-path transitions which show high degrees of branching; which implies that these transitions were frequently utilized. While we only sampled ten regular expressions per neuron from the RE lookup layer, it should be noted that there exist many more regular expressions to explore. Therefore, the following regular expressions only function as samples to provide us with some notion of what kinds of regular expressions the SoPa++ and RE proxy models captured as important.

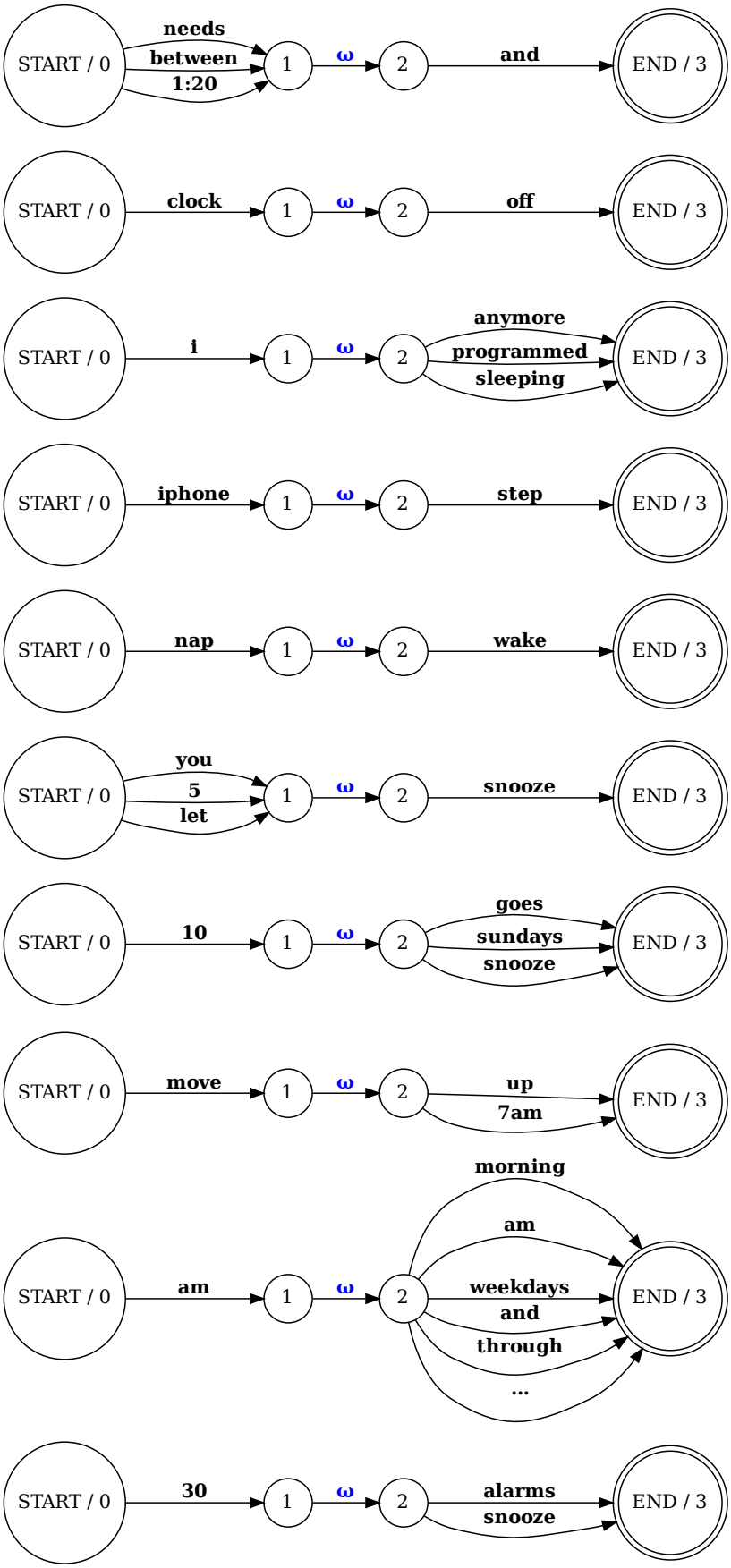


FIGURE 16: Ten sampled regular expressions from the RE lookup layer corresponding to TauSTE neuron 19 for the best performing small RE proxy model; REs are presented as equivalent NFAs; black signifies a main-path transition while blue signifies a $\omega$-transition

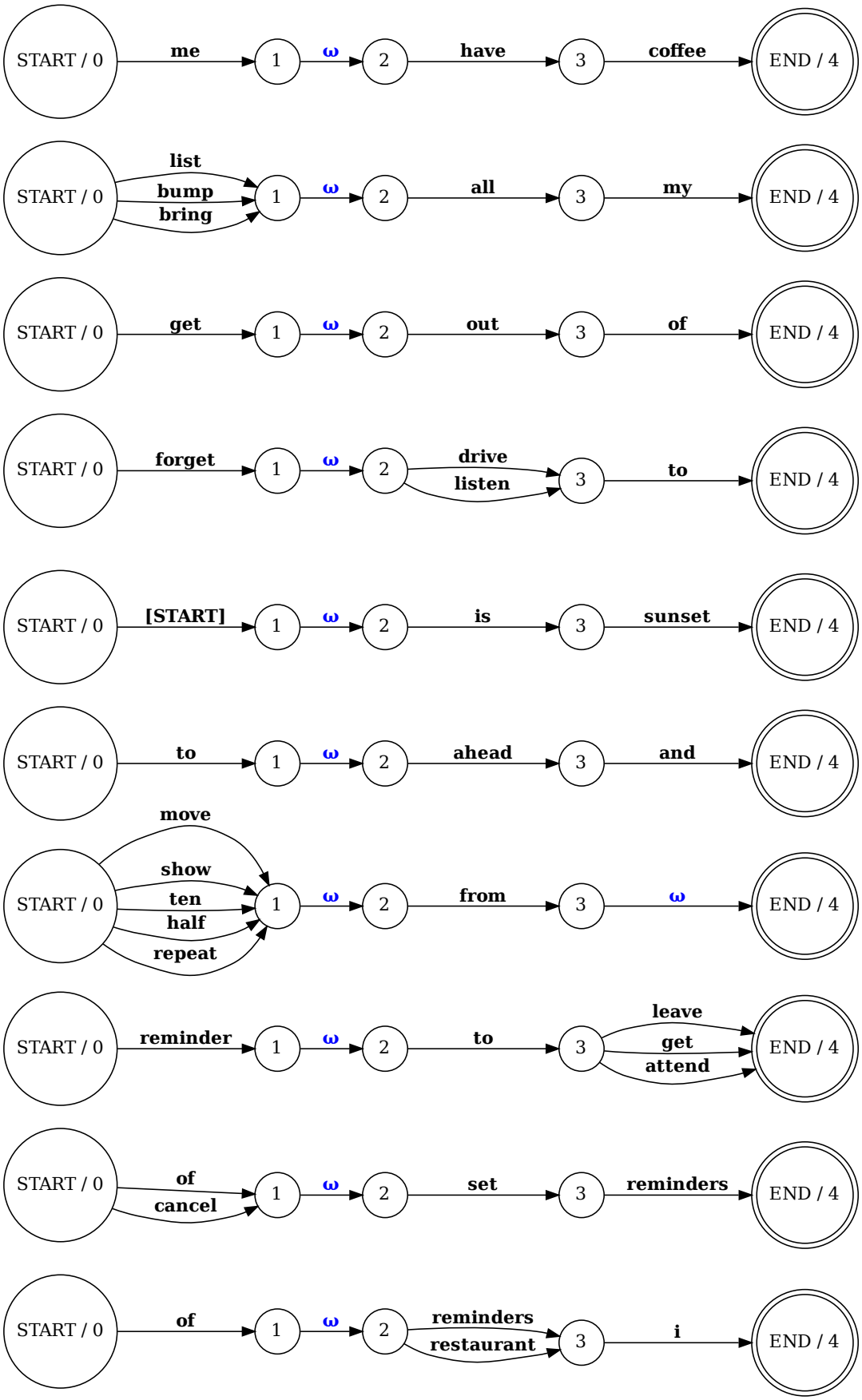


FIGURE 17: Ten sampled regular expressions from the RE lookup layer corresponding to TauSTE neuron 25 for the best performing small RE proxy model; REs are presented as equivalent NFAs; black signifies a main-path transition while blue signifies a $\omega$-transition

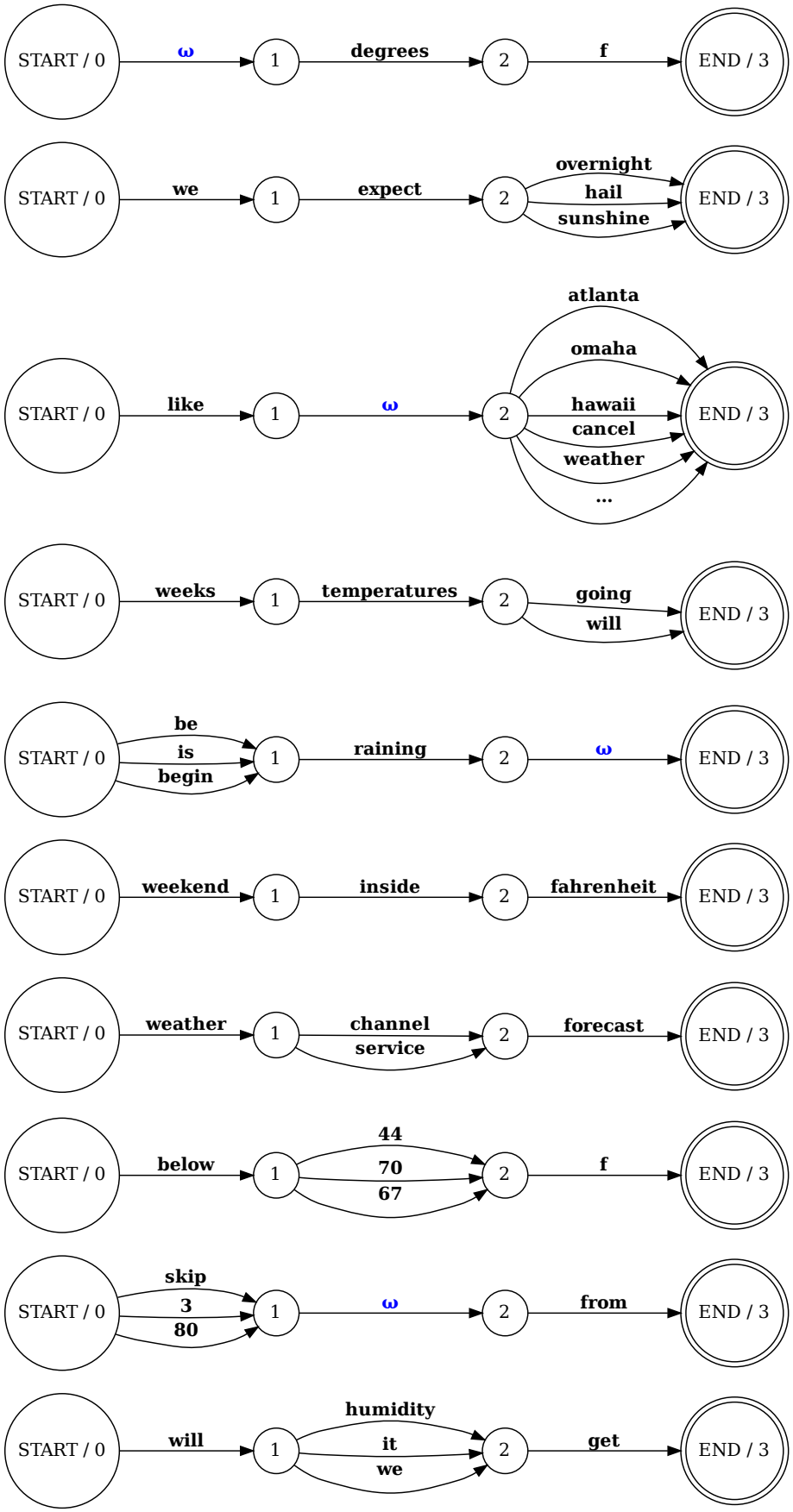


FIGURE 18: Ten sampled regular expressions from the RE lookup layer corresponding to TauSTE neuron 17 for the best performing small RE proxy model; REs are presented as equivalent NFAs; black signifies a main-path transition while blue signifies a $\omega$-transition

# Chapter 5

# Discussion

In this chapter, we interpret the results described from the previous chapter and discuss their implications in order to answer our research questions. Similar to the previous chapter, we break this chapter into three sections with each section addressing one research question. Aside from answering the research questions, we also gather interesting observations and propose hypotheses which could motivate future research.

## 5.1 RQ1: Evaluating performance of SoPa++

To answer our first research question on whether SoPa++ can deliver competitive performance on the FMTOD English language intent classification task, we compare the mean performance metrics of our SoPa++ models against those from other recent studies as mentioned in Section 3.1.3. Referring to our accuracy ranges from Table 6, we observe that the SoPa++ models show an accuracy range from 97.6-98.3% for the best performing models given their respective sizes. This falls into the general accuracy range of 96.6-99.5% observed in other studies as per Table 2; albeit in the lower end of this spectrum. We can therefore conclude that SoPa++ offers competitive performance on the FMTOD's English language intent classification task.

While SoPa++'s performance range falling in the lower end of the aforementioned spectrum can be seen as disadvantageous, it is worth noting that the models SoPa++ is being compared against are vastly different. For one, the BERT-based models shown in Table 3 had parameter counts ranging from ∼110-340 million parameters (Devlin et al., 2019); which are ∼100-300 times larger than our SoPa++ models. In addition, models from Zhang, Lyu, and Callison-Burch (2020) showed an exceptionally high accuracy of 99.5% mainly because of pre-training on the external WikiHow data set for general intent classification tasks. Finally, many of the models described in Table 3 were jointly trained on both FMTOD intent classification and slot filling tasks; which could have contributed to certain joint-task performance benefits. These significant differences between SoPa++ and the aforementioned models should be taken into account when comparing SoPa++'s performance with other studies.

## 5.2 RQ2: Evaluating explanations by simplification

To answer our second research question on whether SoPa++ provides effective explanations by simplification, we summarize the minimum differences in SoPa++ and RE proxy model-pair performance metrics; as well as the minimum distance metrics observed as per Table 7. Regarding performance metrics, we can observe that lowest accuracy score differences between SoPa++ and RE proxy model-pairs to be 7%

for small-sized models, 2% for medium-sized models and 1% for large-sized models. Regarding distance metrics, we observe the lowest $\overline{\delta_\sigma}$ and $\overline{\delta_b}$ to be 10.0% and 12.4% for small-sized models, 5.8 % and 13.5% for medium-sized models and 4.3% and 14.2% for large-sized models respectively. These minimum performance metric differences and distance metrics are typically observed with larger $\tau$-thresholds ranging from 0.50-1.00. Unlike the case for RQ1, we do not have an objective range of competitive accuracy differences or distance metrics to compare against with other studies. As a result, our interpretation of the effectiveness of the explanations by simplification post-hoc explainability technique will be subjective. That being said, we still believe that accuracy differences as low as 1% and softmax distance norms as small as 4.3% provide significant evidence towards a high degree of resemblance between SoPa++ and RE proxy models. In summary, we find that the explanations by simplification post-hoc explainability technique in SoPa++ is effective, in particular for medium and large-sized models with $\tau$-thresholds ranging from 0.50-1.00.

In the interest of objectivity, we would like to provide some perspectives in which the explanations by simplification technique is not effective. For one, explanations by simplification as a post-hoc explainability technique as per Definition 6 explicitly requires the simplified proxy model to be more transparent than its antecedent counterpart. While we made a case for the transparency of the RE proxy model in Section 3.3.5, one could also provide arguments for the RE proxy model being non-transparent; especially when the RE lookup layer contains too many regular expressions for a human to comprehend. This could indeed be the case for medium and large-sized RE proxy models which have RE lookup layers containing tens of thousands of "activating" regular expressions. In cases such, it would not be possible for a human to understand all of the regular expressions; which could ultimately render the RE proxy model as yet another black-box model. In such cases, the explanations by simplification post-hoc explainability technique would likely be ineffective.

With these arguments set aside, we now proceed to discuss some interesting observations in regards to our results for RQ2. Firstly, we can observe the performance-interpretability tradeoff from Section 2.1.5 in Table 7 with the more transparent RE proxy models consistently performing worse than their black-box SoPa++ counterparts; with the exception of the large-sized model-pair with $\tau$=1.00 where the RE proxy generally outperforms the SoPa++ antecedent model. Next, as per Table 7; we observe that RE proxy models tend to perform better as the $\tau$-threshold increases. We hypothesize that this occurs mainly because larger $\tau$-threshold forces the memorization of higher scoring paths which ultimately reduces the chances of the RE proxy model memorizing superfluous or unimportant regular expressions in the RE lookup layer. Finally as per Figure 14, we observe that the $\overline{\delta_b}$ metric continues to decrease as the $\tau$-threshold increases, while the $\overline{\delta_\sigma}$ metric plateaus beforehand and then slighly increases. This could be seen as counter-intuitive, since more similar TauSTE binary vectors should imply more similar softmax distributions. It would be interesting to further explore these trends with even higher $\tau$-thresholds.

## 5.3 RQ3: Interesting and relevant explanations

To answer our third research question on interesting and relevant explanations derived from SoPa++ on the FMTOD data set, we refer back to our results for this research question and attempt to interpret them. Since this research question is more

open-ended than the previous two, our approach to answer it will also be opinionated and subjective. One interesting observation is in the relative linear weights applied to the TauSTE neurons in Figure 15. We can observe that weights are generally continously distributed across all neurons; with some exceptions such as neurons 19, 25 and 17 where the weights are more skewed towards the alarm, reminder and weather domains respectively. This implies that SoPa++ and RE proxy models still distribute feature importances across TauSTE neurons in a highly connective sense; which also implies that each TauSTE neuron has a non-negligible impact on all classification decisions.

With the identification of the salient TauSTE neurons 19, 25 and 17 specializing in the alarm, reminder and weather domains respectively; we draw out ten regular expression samples from RE lookup layer corresponding to each of these neurons as reflected in Figures 16, 17 and 18 respectively. To extract interesting and relevant explanations, we attempt to interpret these sampled regular expressions. Firstly, we can observe a segmentation of lexical information between the regular expressions corresponding to these neurons. For example, many of the regular expressions corresponding to neuron 19 use words related to alarms such as *"snooze"* and *"clock"*; while those corresponding to neuron 17 use words related to weather such as *"fahrenheit"* and *"forecast"*. Next, we can observe transition clustering or branching in sampled regular expressions across all three TauSTE neurons. This branching phenomenon is interesting because words in these branches can sometimes have very similar lexical semantics. For example, in the third regular expression from the bottom in Figure 18, we observe branching with three different digital tokens *"44"*, *"70"* and *"67"* which represent the temperatures encountered in the training data. Similarly, the third regular expression from the top in Figure 18 shows branching with the tokens *"atlanta"*, *"omaha"* and *"hawaii"*, which all represent locations in the USA encountered in the training data. Finally, we can observe interesting positional, or possibly syntactic, features in the regular expressions in Figures 16 and 17; which all have a $\omega$-transition in the same transition.

Finally, the sampled regular expressions allow us to identify various inductive biases incorporated by SoPa++ and its RE proxy models from the training data. Going back to the digital and location-based tokens mentioned in the previous paragraph, we can observe how the training data induces USA-centric biases pertaining to locations such as *"atlanta"* and hard-coded Fahrenheit temperatures such as *"70"*. As a result, we can extrapolate that the SoPa++ and RE proxy models will likely only perform well on unseen data based in USA-centric domains since they likely would not have encountered tokens from non-USA-centric domains. An advantageous aspect of the RE proxy model is that these inductive biases can be easily identified and also corrected. In the case of correcting USA-centric locations, we could manually add more non-USA-based locations in the branching transition of the third regular expression from the top in Figure 18. Another possible inductive bias could be in the third regular expression from the top in Figure 16, where the first transition only allows for the pronoun *"i"*. This inductive bias could be corrected in the RE proxy model by augmenting it with all other pronouns available in the English language. Finally, we can propagate these manual corrections in the RE proxy model back to SoPa++ by copying the word embeddings and transition matrix diagonals of the biased word to now represent those of the manually added new words.

# Chapter 6

# Conclusions

We now arrive at our Conclusions chapter where we summarize and conclude the main contents of this thesis. We started off this thesis by emphasizing the importance of XAI research and correspondingly laid out clear definitions of XAI-related concepts adapted from Arrieta et al. (2020). We then drew inspiration from the SoPa model in Schwartz, Thomson, and Smith (2018) and focused on improving the SoPa model to ultimately create our SoPa++ model. The most significant changes in SoPa++ included the modification of linear-chain WFAs to strict linear-chain WFA-$\omega$'s, the replacement of the MLP with layer normalization, TauSTE and linear layers and the introduction of the explanations by simplification post-hoc explainability technique to create RE proxy models. With these changes, we proceeded to answer our three research questions listed in Section 1.2.

Regarding our first research question, we observe that SoPa++'s best accuracy range on the FMTOD data set of 97.6-98.3% falls into the competitive accuracy range of 96.6-99-5% based on other recent studies. In this respect, we conclude that SoPa++ offers competitive performance on the FMTOD English language intent classification task.

Regarding our second research question, we compare the accuracy scores and distance metrics between SoPa++ and RE proxy model pairs and observe accuracy differences as low as $\sim$1% and softmax distance norms as small as $\sim$4% for medium and large-sized models with $\tau$-thresholds ranging from 0.50-1.00. We therefore conclude that the explanations by simplification post-hoc explainability technique is effective on the FMTOD English language intent classification task given larger model sizes and $\tau$-thresholds.

Regarding our third and final research question, we identify salient TauSTE neurons which received disproportionately large relative linear weights in our best performing small RE proxy model. Next, we analyze regular expression samples in the RE lookup layer from the aforementioned RE proxy model corresponding to these salient neurons. Based on an analysis of these sampled regular expressions, we observe several interesting phenomena such as lexical segmentation, branching transitions with tokens having similar lexical semantics and also the presence of USA-centric inductive biases captured from the training data.

# Chapter 7

# Further work

In this final chapter, we address the limitations of our methodologies and discuss possible future research directions which could be pursued to mitigate these limitations.

## 7.1 Efficiency

In this thesis, we showed the effectiveness of simplifying SoPa++ models into RE proxy models. While SoPa++ models ran very efficiently because of tensor-based parallelizations and GPU hardware-acceleration derived from `PyTorch`'s mature computational ecosystem, we observed that the RE proxy models ran significantly slower with the main bottleneck being the slow lookup process in the RE lookup layer; as was mentioned in Section 3.3.4. To overcome this low efficiency in RE proxy models, we could recommend three approaches. One approach could be to save the RE lookup layer as a data base and utilize indexed searches to make regular expression searches and lookups much faster. Another approach could be to attempt CPU-based multi-threading on the RE lookup process; which is likely to be a complex task and would require experimental tweaking to attain the best efficiencies. The final approach could be to utilize GPU-accelerated regular expression matching algorithms (Wang et al., 2011; Zu et al., 2012; Yu and Becchi, 2013) to parallelize the overall RE lookup layer and its matching functionalities.

## 7.2 Generalization

Returning back to the last two paragraphs in Section 5.3, we recall how certain transitions in the RE lookup layer tend to be populated with tokens which have similar lexical semantics. We specifically draw attention to the example of digital temperature tokens mentioned in the aforementioned paragraphs and how both SoPa++ and the RE proxy model tend to memorize specific digital tokens such as *"44"*, *"70"* and *"67"*. While it is clear that these tokens could be replaced by any other finite digital tokens, both SoPa++ and the RE proxy model overfit on these particular tokens. It would therefore be of interest to explore generalizations on these types of tokens. In the case of the aforementioned digital tokens, we could replace these transitions in the RE lookup layer with a generic Perl-compatible regular expression `\-?[\d]+\.?[\d]*` which would match digital tokens with arbitrary lengths, period-separated decimal precision and a possible minus sign. As a result, this transition would now be able to accomodate various digital tokens with different formats and scales. It would be interesting to explore similar generalizations on other branching transitions such as those for commmunicating time. Naturally, this process would be

much more difficult for tokens that do not have consistent formatting; which could include synonyms and proper nouns.

## 7.3 Modeling

Currently, the SoPa++ model conducts classification decisions by max-pooling document scores from its constitutent strict-linear chain WFA-$\omega$'s. These document scores correspondingly reflect substrings in the document but do not necessarily contain positional information regarding them, such as substring X occurring after substring Y. While this was not a shortcoming in the generally short-sequence FMTOD data set, it could become a hindrance when applying SoPa++ to longer documents. As a result, it would be interesting to incorporate positional information alongside the Viterbi algorithm to allow for classification on longer documents. Another possible extension could be to extend SoPa++'s weighted finite-state automata to finite-state transducers; which are highly similar but return scored seqeuences instead of scored paths. In this way, SoPa++ with finite-state transducers could even be used for sequence-to-sequence tasks; making it viable for other applications in NLP such as Machine Translation (MT).

## 7.4 Explainability

In regards to explainability, we largely focused on the technical requirements for showing effective explanations by simplification for the SoPa++ model. To evaluate the quality of explanations of SoPa vs. SoPa++, we only used the three guidelines for good explanations as per Section 2.1.6. While useful, these are ultimately only guidelines and do not necessarily reflect the evaluation of explanations perceived by the target audiences of SoPa and SoPa++; which are average end-users and exper-users respectively. As a result, it would be interesting to conduct a survey on how satisfactory the provided explanations from SoPa and SoPa++ were to these target audiences. This survey could, for example, be done in a University's department-wide setting using one of the many well-developed web-based survey tools where participants provide a basic positive or negative rating on the provided explanations from their assigned models. Although subjective, this might help to provide a rating of the explainability techniques of SoPa and SoPa++ through their respective target audiences.

# Bibliography


Arrieta, Alejandro Barredo, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, et al. (2020). "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI". In: *Information Fusion* 58, pp. 82–115.

Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E. Hinton (2016). *Layer Normalization*. arXiv: 1607.06450 [stat.ML].

Bastani, Osbert, Carolyn Kim, and Hamsa Bastani (2017). "Interpretability via Model Extraction". In: *CoRR* abs/1706.09773. arXiv: 1706.09773. URL: http://arxiv.org/abs/1706.09773.

Baum, Leonard E and Ted Petrie (1966). "Statistical inference for probabilistic functions of finite state Markov chains". In: *The annals of mathematical statistics* 37.6, pp. 1554–1563.

Bengio, Yoshua, Nicholas Léonard, and Aaron C. Courville (2013). "Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation". In: *CoRR* abs/1308.3432. arXiv: 1308.3432. URL: http://arxiv.org/abs/1308.3432.

Bird, Steven and Edward Loper (July 2004). "NLTK: The Natural Language Toolkit". In: *Proceedings of the ACL Interactive Poster and Demonstration Sessions*. Barcelona, Spain: Association for Computational Linguistics, pp. 214–217. URL: https://www.aclweb.org/anthology/P04-3031.

Courbariaux, Matthieu and Yoshua Bengio (2016). "BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1". In: *CoRR* abs/1602.02830. arXiv: 1602.02830. URL: http://arxiv.org/abs/1602.02830.

Danilevsky, Marina, Kun Qian, Ranit Aharonov, Yannis Katsis, Ban Kawas, and Prithviraj Sen (Dec. 2020). "A Survey of the State of Explainable AI for Natural Language Processing". In: *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*. Suzhou, China: Association for Computational Linguistics, pp. 447–459. URL: https://www.aclweb.org/anthology/2020.aacl-main.46.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (June 2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://www.aclweb.org/anthology/N19-1423.

Doran, Derek, Sarah Schulz, and Tarek R. Besold (2017). "What Does Explainable AI Really Mean? A New Conceptualization of Perspectives". In: *CoRR* abs/1710.00794. arXiv: 1710.00794. URL: http://arxiv.org/abs/1710.00794.

Eisner, Jason (2002). "Parameter estimation for probabilistic finite-state transducers". In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pp. 1–8.

Jiang, Chengyue, Yinggong Zhao, Shanbo Chu, Libin Shen, and Kewei Tu (2020). "Cold-start and Interpretability: Turning Regular Expressions into Trainable Recurrent Neural Networks". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 3193–3207.

Kingma, Diederik P. and Jimmy Ba (2015). "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. URL: http://arxiv.org/abs/1412.6980.

Konig, Rikard, Ulf Johansson, and Lars Niklasson (2008). "G-REX: A versatile framework for evolutionary data mining". In: *2008 IEEE International Conference on Data Mining Workshops*. IEEE, pp. 971–974.

Kuich, Werner and Arto Salomaa (1986). "Linear Algebra". In: *Semirings, automata, languages*. Springer, pp. 5–103.

Lundberg, Scott M. and Su-In Lee (2017). "A Unified Approach to Interpreting Model Predictions". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 4768–4777. ISBN: 9781510860964.

Maletti, Andreas (2017). "Survey: Finite-state technology in natural language processing". In: *Theoretical Computer Science* 679, pp. 2–17.

McNaughton, Robert and Hisao Yamada (1960). "Regular expressions and state graphs for automata". In: *IRE transactions on Electronic Computers* 1, pp. 39–47.

Miller, Tim (2019). "Explanation in artificial intelligence: Insights from the social sciences". In: *Artificial Intelligence* 267, pp. 1–38. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2018.07.007. URL: https://www.sciencedirect.com/science/article/pii/S0004370218305988.

Peng, Hao, Roy Schwartz, Sam Thomson, and Noah A. Smith (2018). "Rational Recurrences". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, pp. 1203–1214. DOI: 10.18653/v1/D18-1152. URL: https://www.aclweb.org/anthology/D18-1152.

Pennington, Jeffrey, Richard Socher, and Christopher D Manning (2014). "Glove: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543.

Peñarrubia, Jorge, Yin-Zhe Ma, Matthew G. Walker, and Alan McConnachie (July 2014). "A dynamical model of the local cosmic expansion". In: *Monthly Notices of the Royal Astronomical Society* 443.3, pp. 2204–2222. ISSN: 0035-8711. DOI: 10.1093/mnras/stu879. eprint: https://academic.oup.com/mnras/article-pdf/443/3/2204/4932191/stu879.pdf. URL: https://doi.org/10.1093/mnras/stu879.

Ribeiro, Marco Tulio, Sameer Singh, and Carlos Guestrin (2016). ""Why Should I Trust You?": Explaining the Predictions of Any Classifier". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pp. 1135–1144.

Sanh, Victor, Lysandre Debut, Julien Chaumond, and Thomas Wolf (2019). "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter". In: *NeurIPS EMC$^2$ Workshop*.

Schuster, Sebastian, Sonal Gupta, Rushin Shah, and Mike Lewis (June 2019). "Cross-lingual Transfer Learning for Multilingual Task Oriented Dialog". In: *Proceedings*

*of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 3795–3805. DOI: 10.18653/v1/N19-1380. URL: https://www.aclweb.org/anthology/N19-1380.

Schwartz, Roy, Sam Thomson, and Noah A. Smith (July 2018). "Bridging CNNs, RNNs, and Weighted Finite-State Machines". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, pp. 295–305. DOI: 10.18653/v1/P18-1028. URL: https://www.aclweb.org/anthology/P18-1028.

Sipser, Michael (1996). "Introduction to the Theory of Computation". In: *ACM Sigact News* 27.1, pp. 27–29.

Suresh, Ananda Theertha, Brian Roark, Michael Riley, and Vlad Schogol (Sept. 2019). "Distilling weighted finite automata from arbitrary probabilistic models". In: *Proceedings of the 14th International Conference on Finite-State Methods and Natural Language Processing*. Dresden, Germany: Association for Computational Linguistics, pp. 87–97. DOI: 10.18653/v1/W19-3112. URL: https://www.aclweb.org/anthology/W19-3112.

Tan, Sarah, Rich Caruana, Giles Hooker, and Yin Lou (2018). "Distill-and-compare: Auditing black-box models using transparent model distillation". In: *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, pp. 303–310.

Thompson, Ken (1968). "Programming techniques: Regular expression search algorithm". In: *Communications of the ACM* 11.6, pp. 419–422.

Townsend, Joseph, Thomas Chaton, and João M Monteiro (2019). "Extracting relational explanations from deep neural networks: A survey from a neural-symbolic perspective". In: *IEEE transactions on neural networks and learning systems* 31.9, pp. 3456–3470.

Tsuruoka, Yoshimasa, Jun'ichi Tsujii, and Sophia Ananiadou (2009). "Fast full parsing by linear-chain conditional random fields". In: *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, pp. 790–798.

Viterbi, Andrew (1967). "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm". In: *IEEE transactions on Information Theory* 13.2, pp. 260–269.

Wang, Cheng and Mathias Niepert (2019). "State-Regularized Recurrent Neural Networks". In: ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. Long Beach, California, USA: PMLR, pp. 6596–6606. URL: http://proceedings.mlr.press/v97/wang19j.html.

Wang, Lei, Shuhui Chen, Yong Tang, and Jinshu Su (2011). "Gregex: Gpu based high speed regular expression matching engine". In: *2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*. IEEE, pp. 366–370.

Yin, Penghang, Jiancheng Lyu, Shuai Zhang, Stanley J. Osher, Yingyong Qi, and Jack Xin (2019). "Understanding Straight-Through Estimator in Training Activation Quantized Neural Nets". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=Skh4jRcKQ.

Yu, Xiaodong and Michela Becchi (2013). "GPU acceleration of regular expression matching for large datasets: exploring the implementation space". In: *Proceedings of the ACM International Conference on Computing Frontiers*, pp. 1–10.

Zeiler, Matthew D and Rob Fergus (2014). "Visualizing and understanding convolutional networks". In: *European conference on computer vision*. Springer, pp. 818–833.

Zhang, Li, Qing Lyu, and Chris Callison-Burch (Dec. 2020). "Intent Detection with WikiHow". In: *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*. Suzhou, China: Association for Computational Linguistics, pp. 328–333. URL: https://www.aclweb.org/anthology/2020.aacl-main.35.

Zhang, Zhichang, Zhenwen Zhang, Haoyuan Chen, and Zhiman Zhang (2019). "A joint learning framework with bert for spoken language understanding". In: *IEEE Access* 7, pp. 168849–168858.

Zu, Yuan, Ming Yang, Zhonghu Xu, Lin Wang, Xin Tian, Kunyang Peng, and Qunfeng Dong (2012). "GPU-based NFA implementation for memory efficient high speed regular expression matching". In: *Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming*, pp. 129–140.